

NebulaGraph

Nebula Graph Database 手册

v3.1.0

吴敏，周瑶，梁振亚，杨怡璇，黄凤仙

2022 vesoft Inc.

Table of contents

1. 欢迎阅读 Nebula Graph 3.1.0 文档 7
1.1 快速开始 7
1.2 其他资料 7
1.3 图例说明 7
1.4 修改文档中的错误 8
2. 简介 9
2.1 图 9
2.2 图数据库的市场概况 26
2.3 相关技术 37
2.4 什么是 Nebula Graph 50
2.5 数据模型 53
2.6 路径 55
2.7 点 VID 57
2.8 服务架构 59
3. 快速入门 75
3.1 快速入门 75
3.2 步骤 1：安装 Nebula Graph 76
3.3 步骤 2：启动 Nebula Graph 服务 79
3.4 步骤 3：连接 Nebula Graph 84
3.5 注册 Storage 服务 86
3.6 步骤 4：使用常用 nGQL（CRUD 命令） 87
3.7 nGQL 命令汇总 97
4. nGQL 指南 117
4.1 nGQL 概述 117
4.2 数据类型 132
4.3 变量和复合查询 151
4.4 运算符 156
4.5 函数和表达式 169
4.6 通用查询语句 195
4.7 子句和选项 232
4.8 图空间语句 260
4.9 Tag 语句 268
4.10 Edge type 语句 276
4.11 点语句 282
4.12 边语句 289

2022 vesoft Inc.

4.13 原生索引 296
4.14 全文索引 307
4.15 子图和路径 315
4.16 查询调优与终止 322
4.17 作业管理 326
5. 安装部署 330
5.1 准备编译、安装和运行 Nebula Graph 的环境 330
5.2 编译与安装 336
5.3 存算合并版 Nebula Graph 355
5.4 设置 Nebula Graph 企业版 License 357
5.5 管理 Nebula Graph 服务 360
5.6 连接 Nebula Graph 服务 365
5.7 管理 Storage 主机 367
5.8 升级版本 368
5.9 卸载 Nebula Graph 374
6. 配置与日志 376
6.1 配置 376
6.2 日志 393
7. 监控 398
7.1 查询 Nebula Graph 监控指标 398
7.2 RocksDB 统计数据 406
8. 数据安全 407
8.1 验证和授权 407
8.2 SSL 加密 416
9. 备份与恢复 418
9.1 Nebula BR 418
9.2 管理快照 427
10. 同步与迁移 429
10.1 BALANCE 429
10.2 集群间数据同步 430
11. 最佳实践 436
11.1 Compaction 436
11.2 Storage 负载均衡 438
11.3 图建模设计 441
11.4 系统设计建议 445
11.5 执行计划 446
11.6 超级顶点（稠密点）处理 447
11.7 实践案例 449

2022 vesoft Inc.

12. 客户端 450
12.1 客户端介绍 450
12.2 Nebula Console 451
12.3 Nebula CPP 455
12.4 Nebula Java 457
12.5 Nebula Python 459
12.6 Nebula Go 460
13. Nebula Graph Cloud 461
13.1 什么是 NebulaGraph Cloud 461
13.2 Nebula Graph Cloud 阿里云版 462
13.3 Nebula Graph Cloud AWS 版 469
13.4 Nebula Graph Cloud Azure 版 476
14. Nebula Studio 481
14.1 Studio 版本更新说明 481
14.2 认识 Nebula Studio 482
14.3 安装与登录 485
14.4 快速开始 499
14.5 故障排查 520
15. Nebula Dashboard（社区版） 523
15.1 什么是 Nebula Dashboard（社区版） 523
15.2 部署 Dashboard 525
15.3 连接 Dashboard 529
15.4 Dashboard 页面介绍 531
15.5 监控指标说明 535
16. Nebula Dashboard（企业版） 543
16.1 什么是 Nebula Dashboard（企业版） 543
16.2 部署 Dashboard 企业版 545
16.3 Nebula Dashboard 企业版 License 552
16.4 创建及导入集群 554
16.5 集群管理 562
16.6 任务中心 593
16.7 权限管理 594
16.8 安装包管理 597
16.9 全局配置 599
16.10 监控指标说明 601
16.11 常见问题 FAQ 609
17. Nebula Explorer 611
17.1 什么是 Nebula Explorer 611

2022 vesoft Inc.

17.2 安装与登录 612
17.3 页面概览 619
17.4 数据库管理 622
17.5 图探索 625
17.6 可视化查询 634
17.7 画布功能 639
17.8 工作流 648
17.9 基本操作和快捷键 675
18. Nebula Importer 676
18.1 Nebula Importer 676
18.2 有表头配置说明 682
18.3 无表头配置说明 685
19. Nebula Exchange 688
19.1 认识 Nebula Exchange 688
19.2 获取 Nebula Exchange 693
19.3 参数说明 695
19.4 使用 Nebula Exchange 704
19.5 Exchange 常见问题 778
20. Nebula Operator 782
20.1 什么是 Nebula Operator 782
20.2 使用流程 784
20.3 部署 Nebula Operator 785
20.4 部署 Nebula Graph集群 791
20.5 配置 Nebula Graph 集群 804
20.6 升级 Nebula Graph 集群 813
20.7 通过 Nebular Operator 连接 Nebula Graph 数据库 816
20.8 故障自愈 820
20.9 常见问题 821
21. 图计算 822
21.1 算法简介 822
21.2 Nebula Algorithm 832
21.3 Nebula Analytics 838
21.4 Nebula Explorer Workflow 844
21.5 Dag Controller 845
22. Nebula Spark Connector 848
22.1 适用场景 848
22.2 特性 848
22.3 更新说明 848

2022 vesoft Inc.

22.4 获取 Nebula Spark Connector 849
22.5 使用方法 849
23. Nebula Flink Connector 854
23.1 适用场景 854
23.2 更新说明 854
24. Nebula Bench 855
24.1 适用场景 855
24.2 更新说明 855
25. 附录 856
25.1 Nebula Graph 3.1.0 release notes 856
25.2 Nebula Graph 学习路径 858
25.3 常见问题 FAQ 865
25.4 生态工具概览 873
25.5 导入工具选择 877
25.6 如何贡献代码和文档 878
25.7 Nebula Graph 年表 882
25.8 思维导图 887
25.9 错误码 891

2022 vesoft Inc.

1. 欢迎阅读 Nebula Graph 3.1.0 文档

Caution

该版本文档包括 Nebula Graph 3.1.0 企业版、社区版内容，和对应版本的周边工具内容。详情查看版本发布说明。

Note

本文档更新时间2023-6-20，GitHub commit 5b2b40d1。

Nebula Graph 是一款开源的、分布式的、易扩展的原生图数据库，能够承载数千亿个点和数万亿条边的超大规模数据集，并且提供毫秒级查询。

1.1 快速开始

- 学习路径与获取认证
- 什么是 Nebula Graph
- 快速开始
- 部署要求
- nGQL 命令汇总
- FAQ
- 生态工具

1.2 其他资料

- 《开源分布式图数据库 Nebula Graph 完全指南》
- 发布说明
- 论坛
- 项目主页
- 系列视频
- 英文文档

1.3 图例说明

Note

额外的信息或者操作相关的提醒等。

Caution

需要严格遵守的注意事项。不遵守 caution 可能导致系统故障、数据丢失、安全问题等。

2022 vesoft Inc.

Danger

会引发危险的事项。不遵守 danger 必定会导致系统故障、数据丢失、安全问题等。

Performance

性能调优时需要注意的事项。

Faq

常见问题。

Compatibility

nGQL 与 openCypher 的兼容性或 nGQL 当前版本与历史版本的兼容性。

Enterpriseonly

描述社区版和企业版的差异。

1.4 修改文档中的错误

Nebula Graph 文档以 Markdown 语言编写。单击文档标题右上侧的铅笔图标即可提交修改建议。

最后更新: June 20, 2023

2022 vesoft Inc.

2. 简介

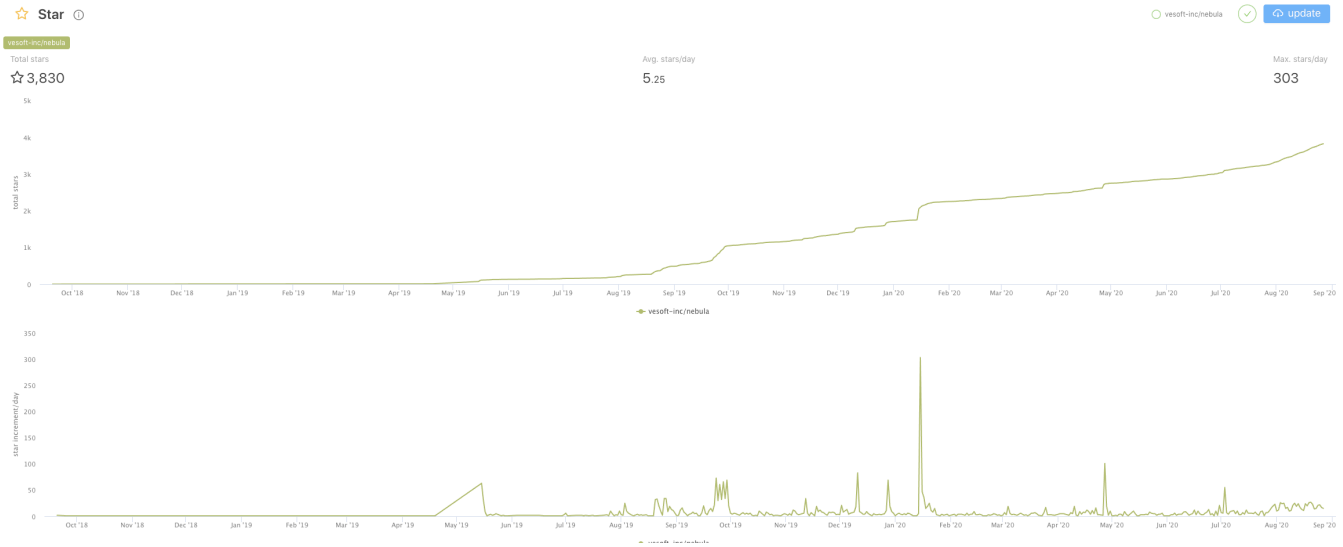
2.1 图

当前，从计算机行业巨头（例如 Amazon 和 Facebook）到小型研究团队，都投入了大量的资源探索图数据库在解决各种数据关系问题上的潜力。当然你也可以选择像他们这样进行尝试，现在可供选择的数据库有很多。那么图数据库究竟是什么？它可以做些什么？作为一类数据库，它在数据库领域里处于什么位置呢？要回答这些问题，我们首先得了解图。

图是计算机科学研究的主要领域之一。图能够高效地解决目前存在的诸多问题。本章将从图说起，继而说明图数据库的优点及其在现代应用程序开发中的巨大潜力，然后介绍分布式图数据库的区别和几种其他类型的数据库。

2.1.1 图、图片与图论

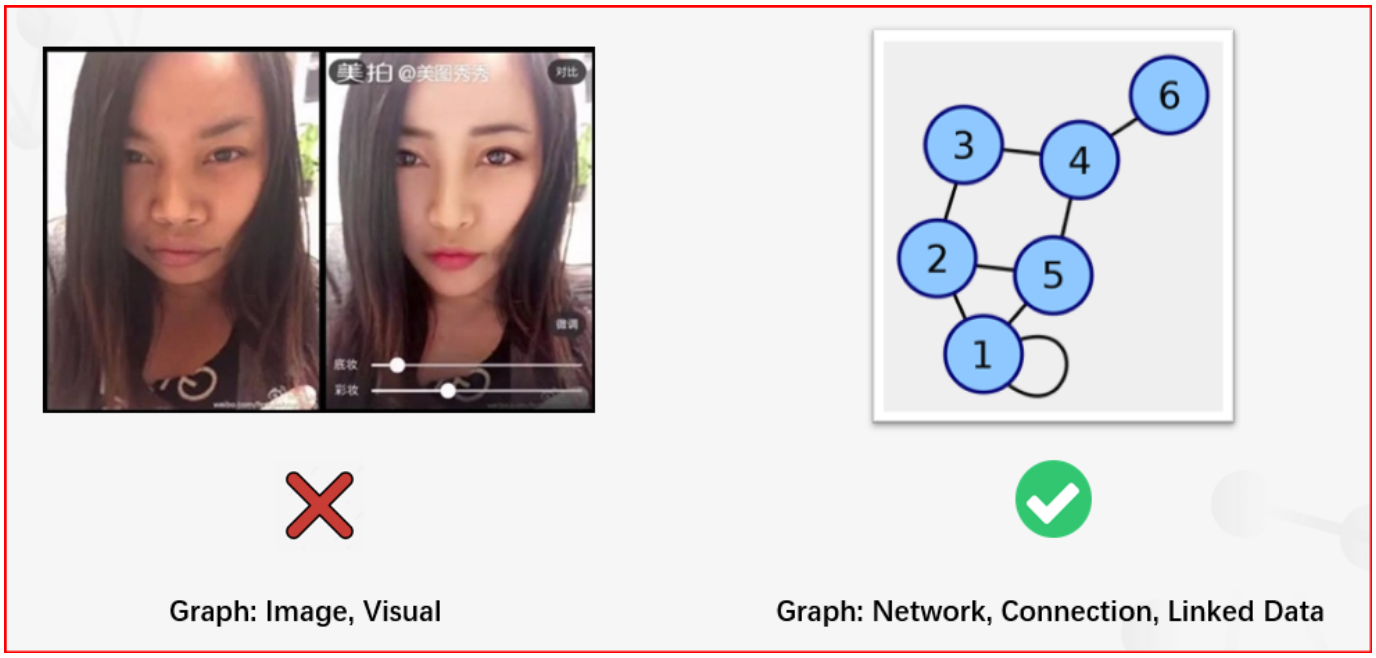
图无处不在。当听到图这个词时，很多人都会想到条形图或折线图，因为有时候我们确实会把它们称作图。从传统意义上来说，图是用来展示两个或多个数据系统之间的联系的。最简单的例子如下图，下图展示了 Nebula Graph GitHub 仓库星星数量随时间推移的变化。



这是相对比较简单的一种图，横轴为时间，纵轴为星星数量。可以看到，星星数量是随着时间推移而上升的。这种类型的图通常称为折线图。折线图可以显示随时间（根据常用比例设置）而变化的连续数据。此处我们只给出了折线图的例子。当然图的形式有多种，比如饼图、条形图等。

还有一种“图”在日常口语中会更多的被提及，例如，“图像识别”，“美图秀”，“修图”等。例如下“图”的左边。

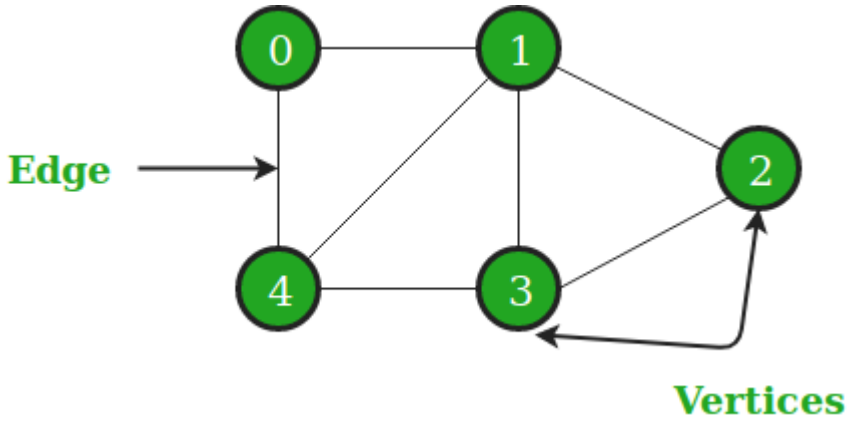
2022 vesoft Inc.



但是——总会有但是——我们在本书中讨论的图是另外一个概念——“图论”中的图。

在数学的分支图论中，图是基本研究对象，用于表示实体与实体之间的关系。一张图由一些小圆点（称为顶点或节点，即 Vertex）和连接这些圆点的直线或曲线（称为边，即 Edge）组成。“图（Graph）”这一名词最早由西尔维斯特在 1878 年提出。

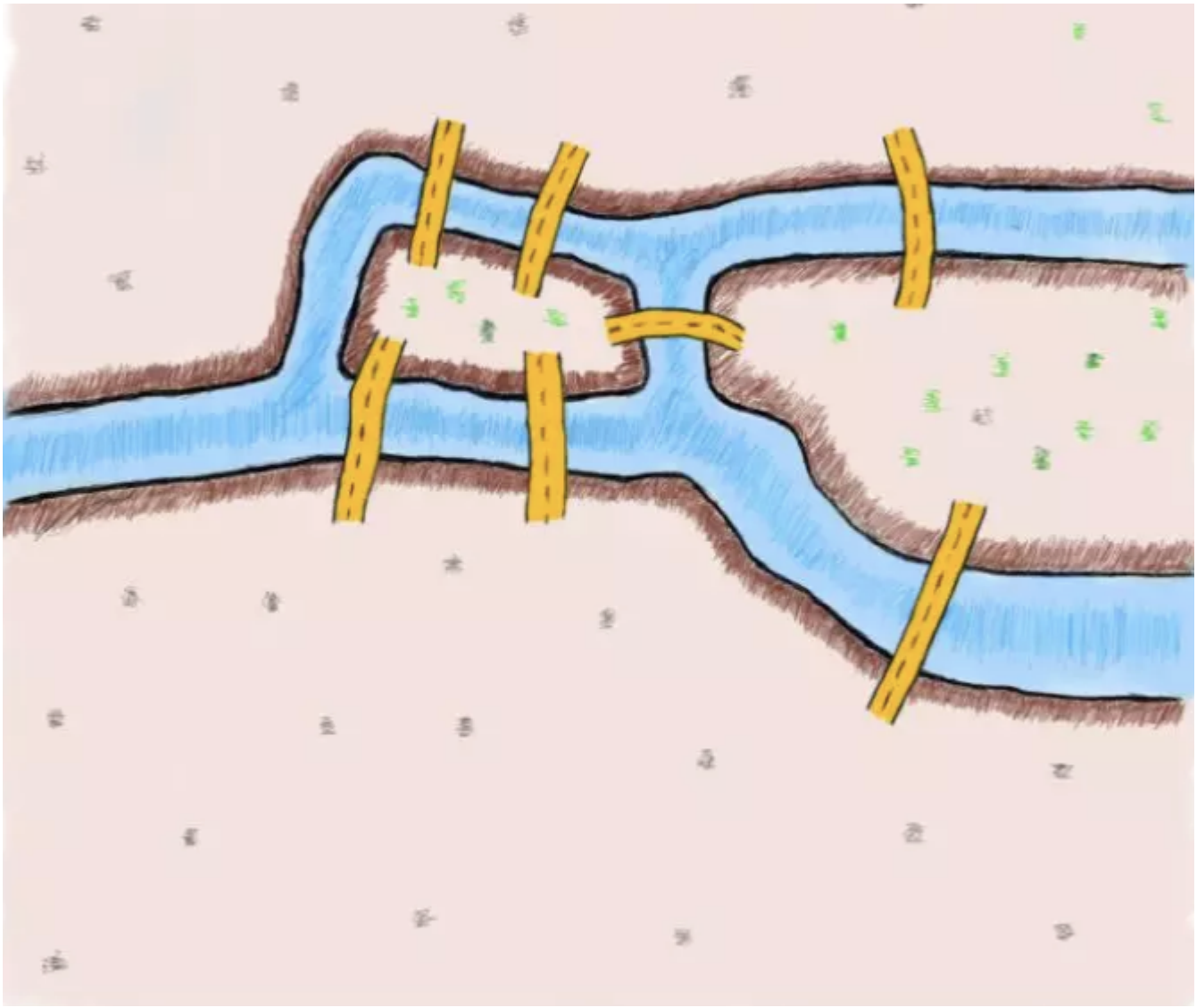
通常，在英文中，为了区分这两种不同的图，前者会称为 Image，后者称为 Graph。在中文中，前者会强调为“图片”，后者会强调为“拓扑图”、“网络图”等。



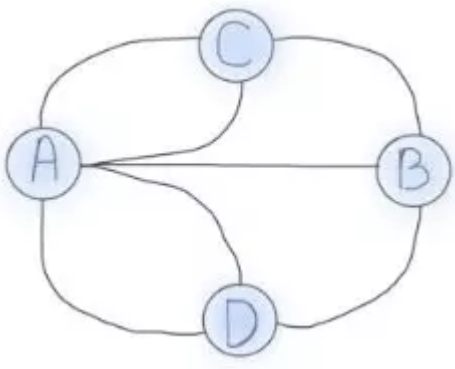
这才是本书所说的图。

简单来说，图论就是研究图的学问。图论始于 18 世纪初期的柯尼斯堡七桥问题。柯尼斯堡当时是普鲁士的城市（现在属于俄罗斯，更名为加里宁格勒）。普雷格尔河穿过柯尼斯堡，不仅把柯尼斯堡分成了两部分，而且还在河中间形成了两个小岛。这就将整个城市分割成了四个区域，各区域由七座桥连接。当时有一个与柯尼斯堡相关的小游戏，即如何只穿过每座桥一次，浏览整个城市的四个区域。下图为柯尼斯堡七座桥的简化图。 有兴趣的话可以试试寻找这个小游戏的答案[1]。

2022 vesoft Inc.



大数学家欧拉为了解决了这一问题。通过将城市的四个区域抽象成点，将连接城市的七座桥抽象成连接点的边，欧拉证明了这一问题是无法解决的。简化的抽象图如下[2]：



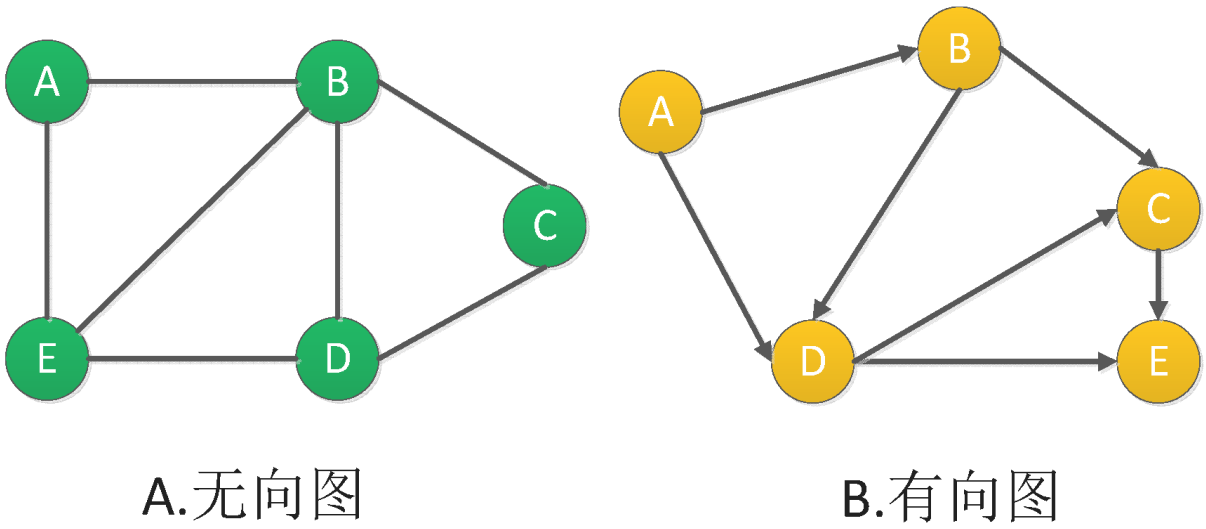
图中四个点代表柯尼斯堡的四个区域，点之间的线代表连接四个区域的七座桥。从图中不难看出，偶数座桥连接的区域可以轻松通过，因为来去可以选择不同的路线。奇数座桥连接的区域只能作为起点或者终点，因为同样的路线只能走一次。和节点相关联的边的条数称为节点度。现在可以证明，只有两个节点有奇数度，另外节点有偶数度时，也即两个区域必须有偶数座桥，剩下的区域有奇数座桥时，柯尼斯堡问题才能解决。然而由上图得知，柯尼斯堡的任何区域都没有偶数座桥，所以这个谜题无解。

2022 vesoft Inc.

2.1.2 属性图

从数学角度来说，图论是研究建模对象之间关系结构的学科。但是从工业界使用的角度，通常会对基础的图模型进行扩展，称为属性图模型。属性图通常由以下几部分组成：

- 节点，即对象或实体。在本书中，通常简称为点（Vertex）。
- 节点之间的关系，在本书中，通常简称为边（Edge）。通常边是有方向或者无方向的，以表示两个实体之间有持续的关系。

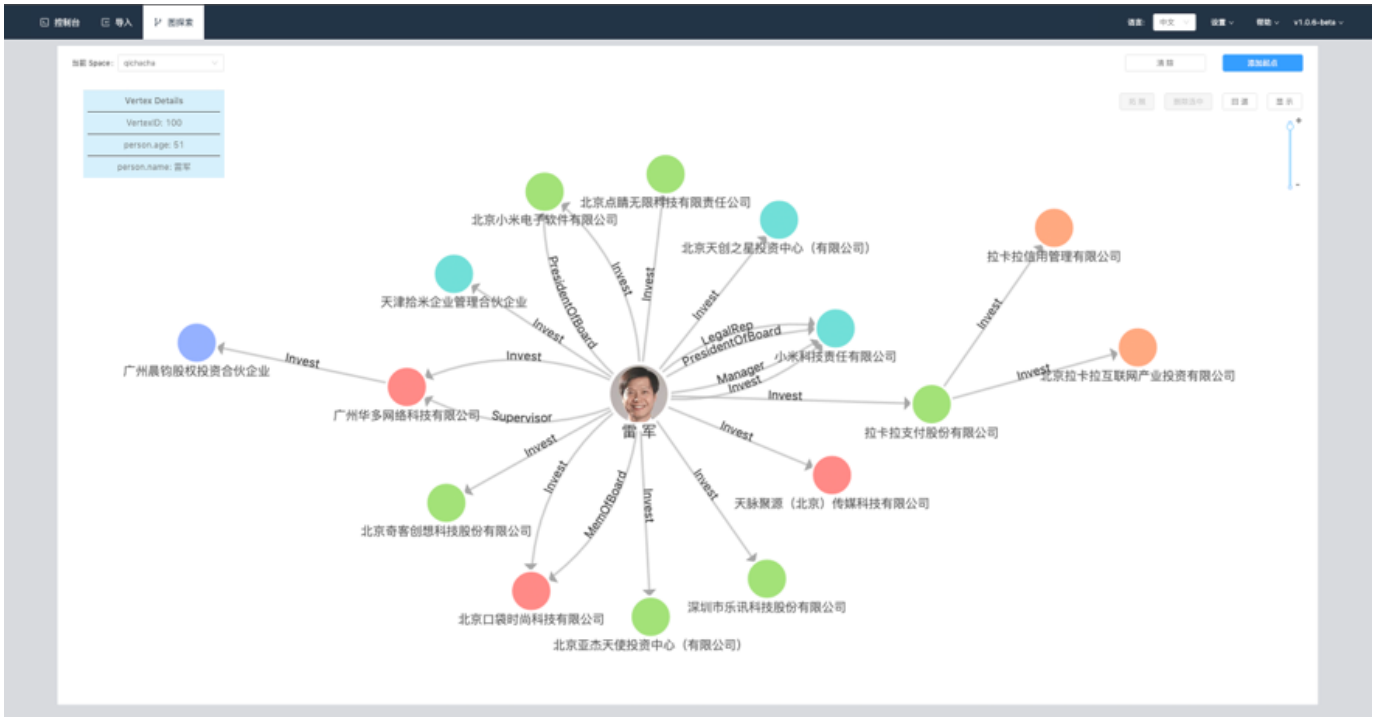


A.无向图　　B.有向图

- 此外，在节点和边上，还可以有属性（properties）。

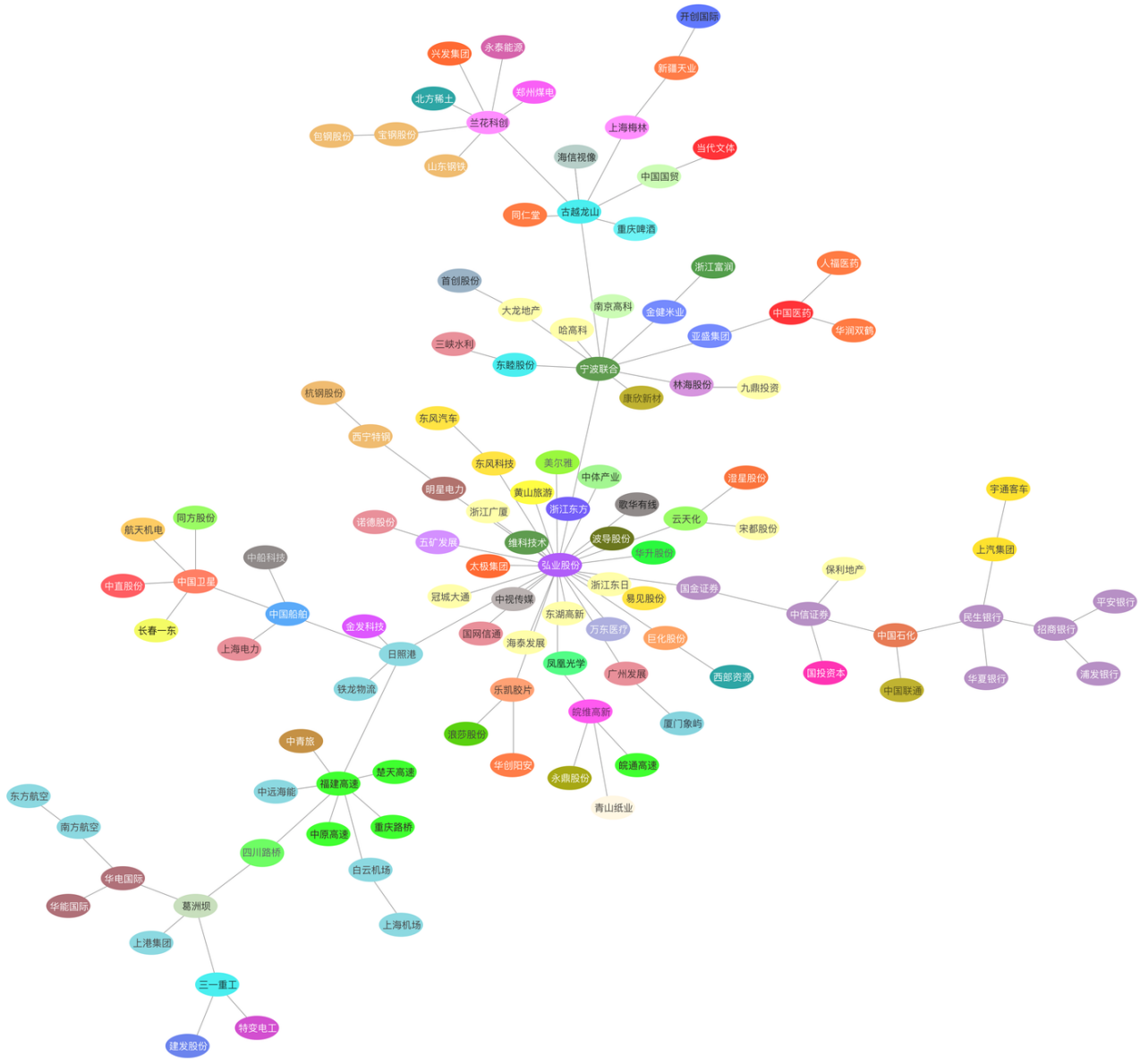
在现实生活中，有很多属性图的例子。

例如企查查或者 BOSS 直聘这类的公司，用图来建模商业股权关系网络。这个网络中，点通常是一个自然人或者是一家企业，边通常是某自然人与某企业之间的股权关系。点上的属性可以是自然人姓名、年龄、身份证号等。边上的属性可以是投资金额、投资时间、董监高等职位关系。



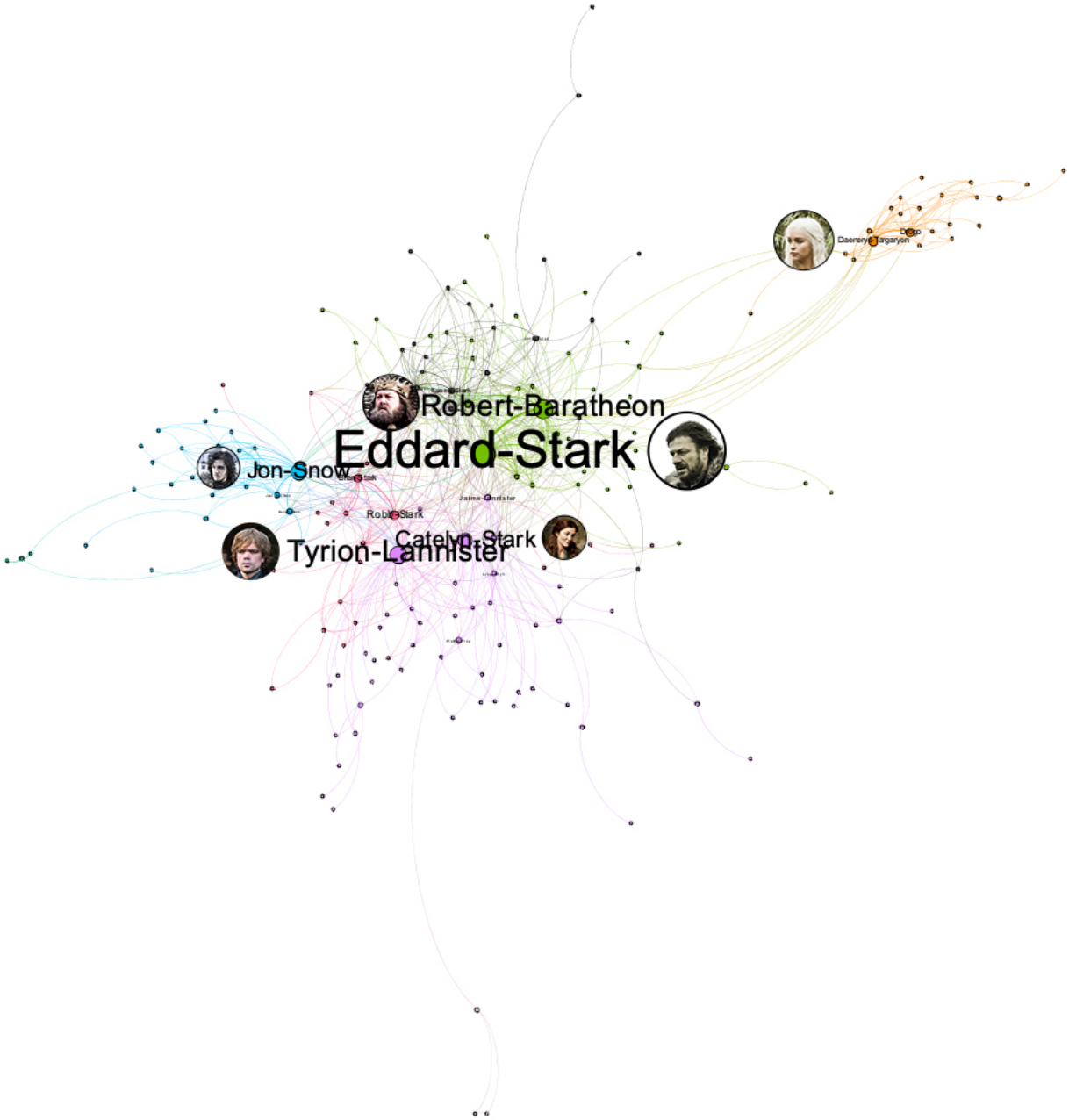
在一个股票市场里面，点可以是一家上市公司，边可以是上市公司之间的相关性。点的属性可以为股票代码、简称、市值、板块等；边的属性可以为股价的时间序列相关性系数[3]。

2022 vesoft Inc.



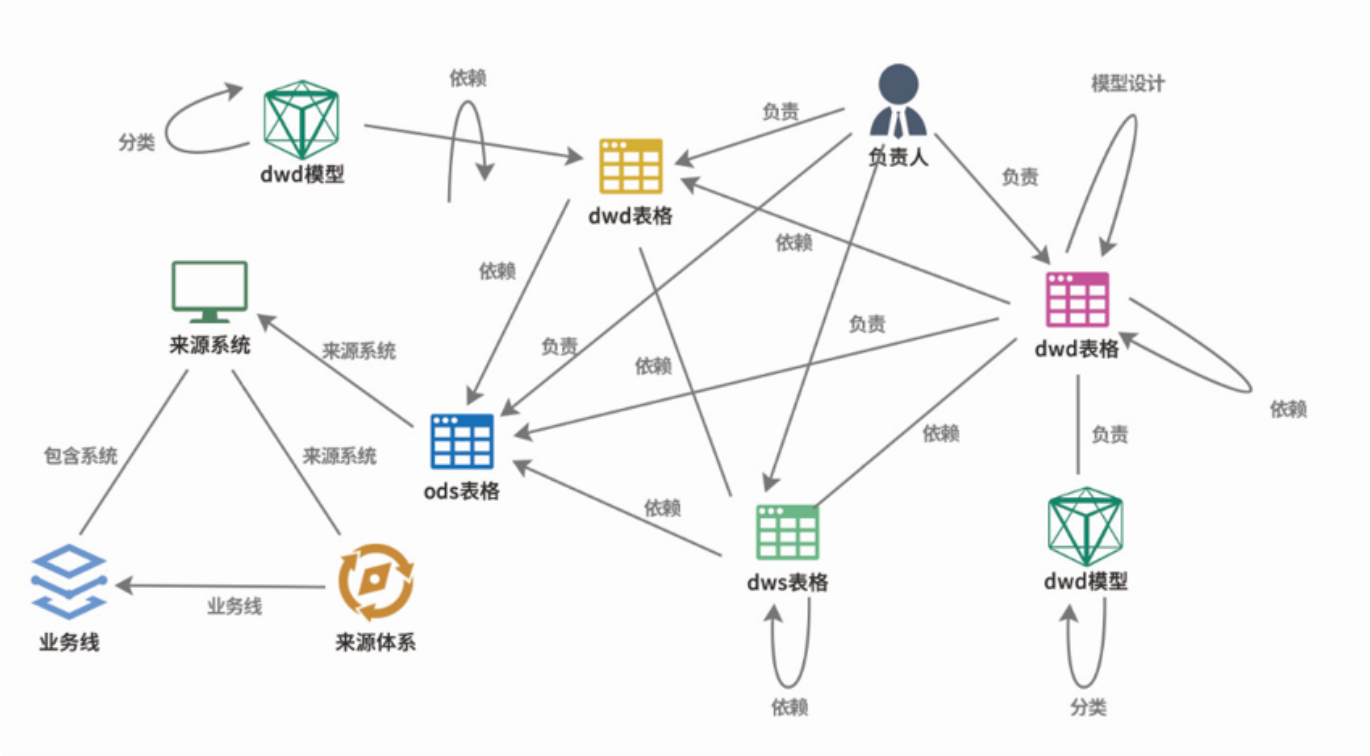
图关系还可以是类似《权力的游戏》这样电视剧中的人物关系网[4]：点为人物，边为人物之间的互动关系；点的属性为人物姓名、年龄、阵营等，边的属性（距离）为两个人物之间的互动次数，互动越频繁距离越近。

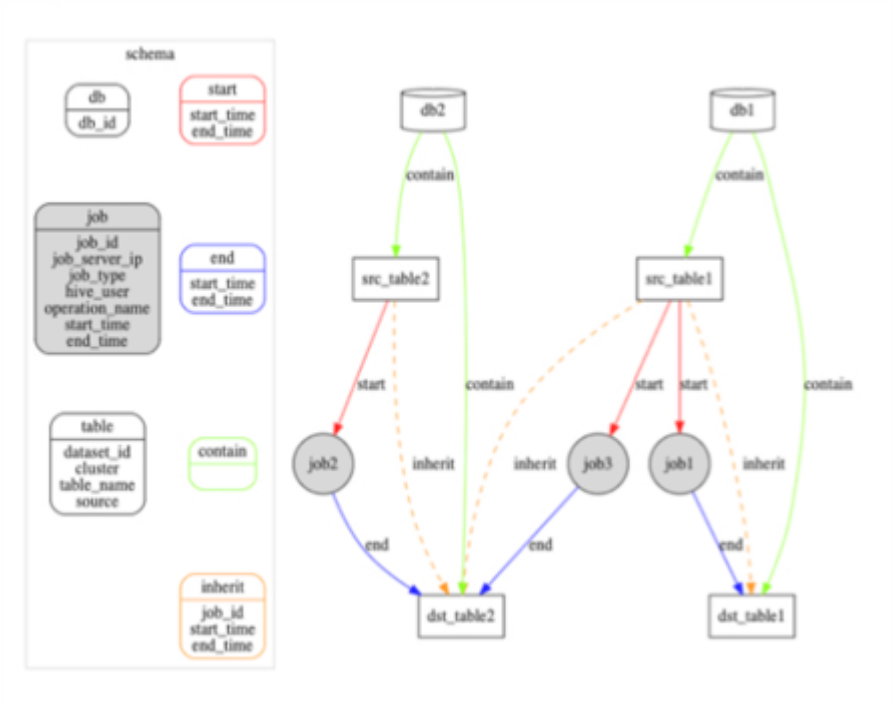
2022 vesoft Inc.



图也可以用于 IT 系统内部的治理。例如，对于像微众银行这样的公司，通常有着非常庞大的数据仓库，以及相应的数仓管理工具。这些管理工具记录了数仓内 Hive 表之间通过 Job 实现的 ETL 关系[5]，这样的 ETL 关系，可以非常方便的用图的形式呈现和管理，当出现问题时也可以非常方便地追溯根源。

2022 vesoft Inc.



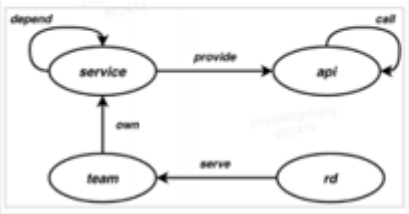


图也可以用于记录一个大型 IT 系统内部错综复杂的微服务之间的调用关系[6]，运维团队用其进行服务治理。这里每个点表示一个微服务，边表示两个微服务之间的调用关系；这样，运维人员可以方便地寻找可用性低于阈值 (99.99%) 的调用链路，或者发现那些出故障会影响面特别大的微服务节点。

2022 vesoft Inc.

服务治理
- 图谱数据
  - 将RPC服务调用关系写入图谱
  - 包含service、api、team等4类实体及5类关系
  - 点边数量在百万级别，实时写入
- 用于服务链路治理和告警优化

```
//查找API com.sankuai.ia.search.api:SearchControllerV2.search过去七天可用率低于99.99%的链路的thrift调用，最大图遍历深度为10
GO 1 TO 10 STEPS FROM
hash("com.sankuai.ia.search.api:SearchControllerV2.search") OVER call WHERE call.availability>0 AND call.availability<1000000 AND
$$.api.type=="mtthrift" YIELD call._src,call._dst
```



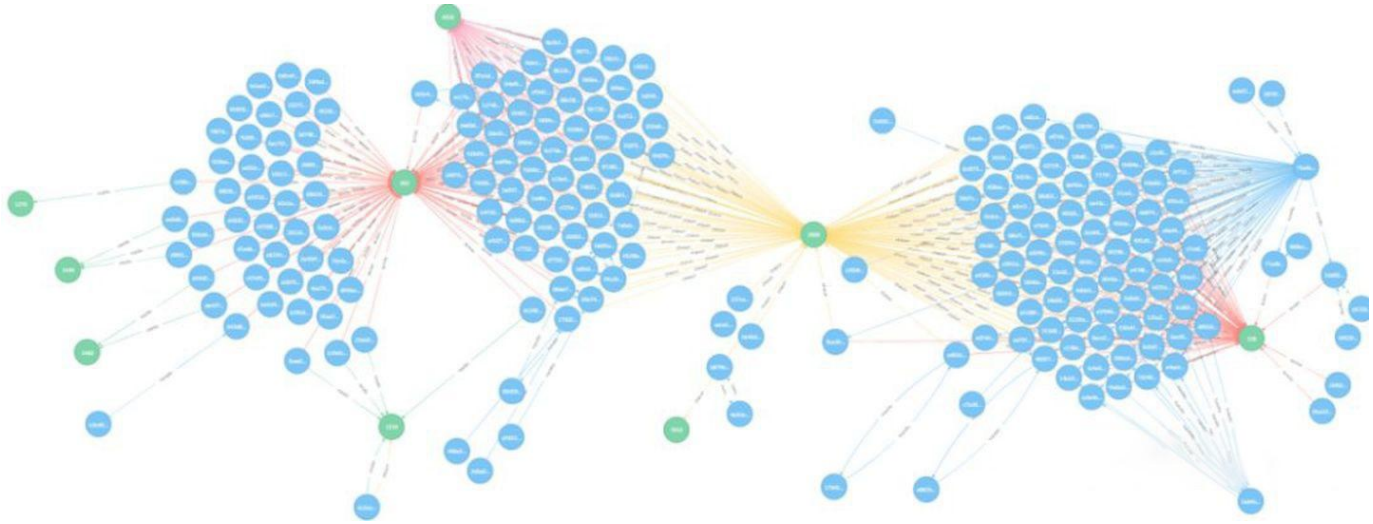

```
//查找所有java类型服务提供的API，并统计其会影响的上游API的数量，从高到低排序看影响次数大小（调用的可用率小于4个9）
LOOKUP ON service WHERE service.type=="java"
| GO FROM $-.VertexID OVER provider YIELD
provider._dst AS java_api_id
| GO FROM $-.java_api_id OVER call REVERSELY WHERE
call.availability>0 AND call.availability<1000000
YIELD call._src AS api_src, call._dst AS api_dst
| GROUP BY $-.api_src YIELD $-.api_src AS api_id,
count(1) AS call_cnt
| ORDER BY call_cnt DESC
| FETCH PROP ON api $-.api_id YIELD
api.appkey,api.method,$-.call_cnt
```

图也可以用于提升代码开发效率。用图存放代码之间的函数调用关系[6]，可以提升研发团队审查和测试代码的效率。在这样的图中，每个点是代码中的一个函数或者变量，每个边是函数或者变量之间的调用关系。当有新提交的代码之时，人们可以更方便的看到可能会受到影响到的其他接口，这样可以帮助测试人员更好的评估潜在的上线风险。

image

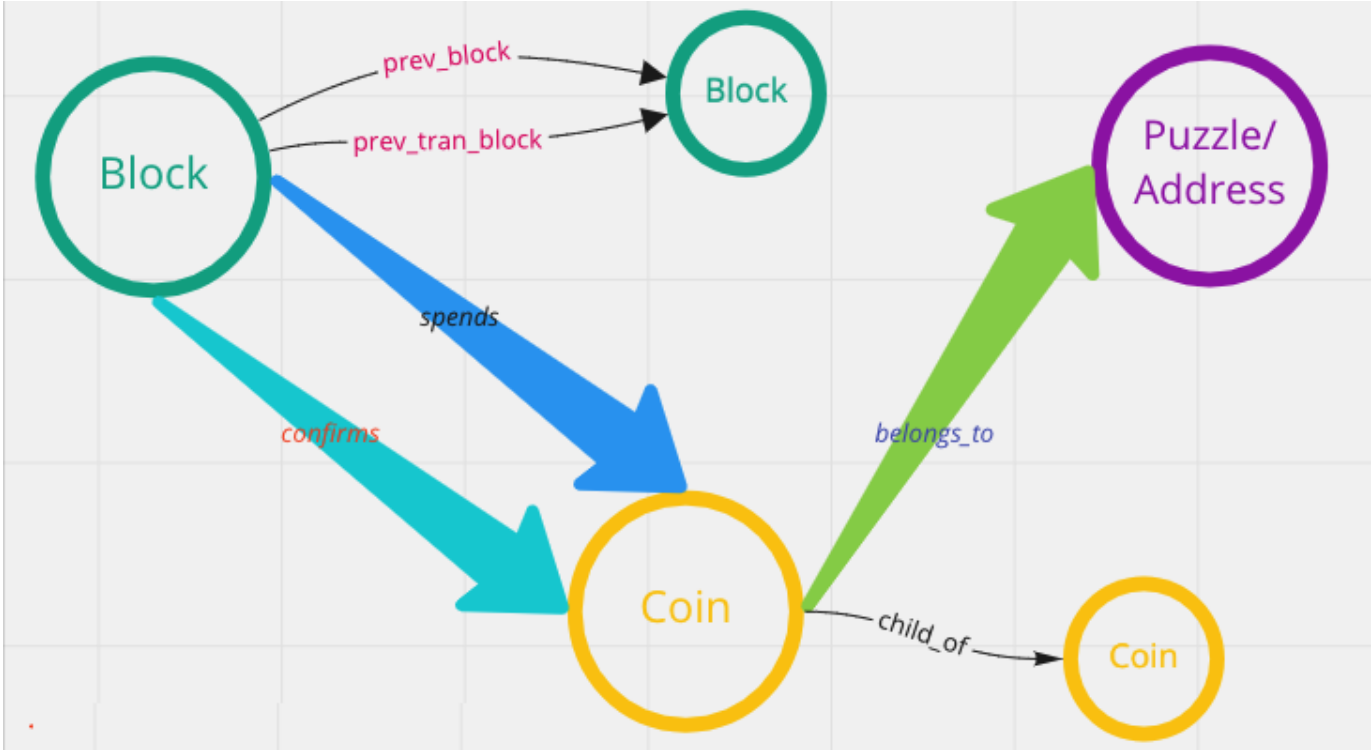
此外，相对于静态不发生变化的属性图，我们还可以通过增加一些时间信息，发掘出更多的使用场景。

例如，在一个银行间账户资金流向网络里面[7]，点是账户，边是账户之间的转账记录。边属性记录了转账的时间、金额等。同盾、邦盛、半云科技等公司采用图技术，可以方便地通过图的方式探索发现明显的资金挪用、“以贷还贷”、“团伙贷款”等现象。

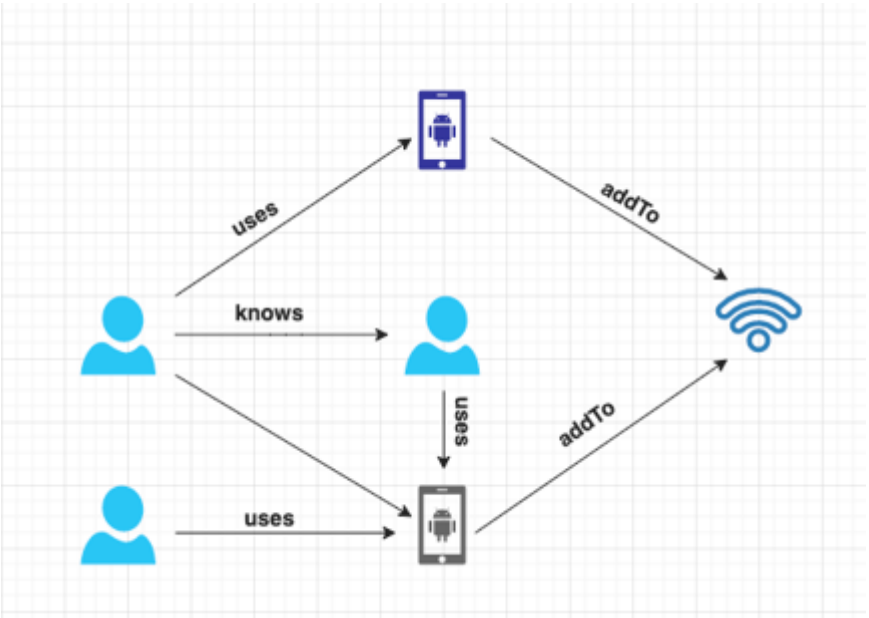


同样的方法也可以用于探索发现加密货币的流向。

2022 vesoft Inc.

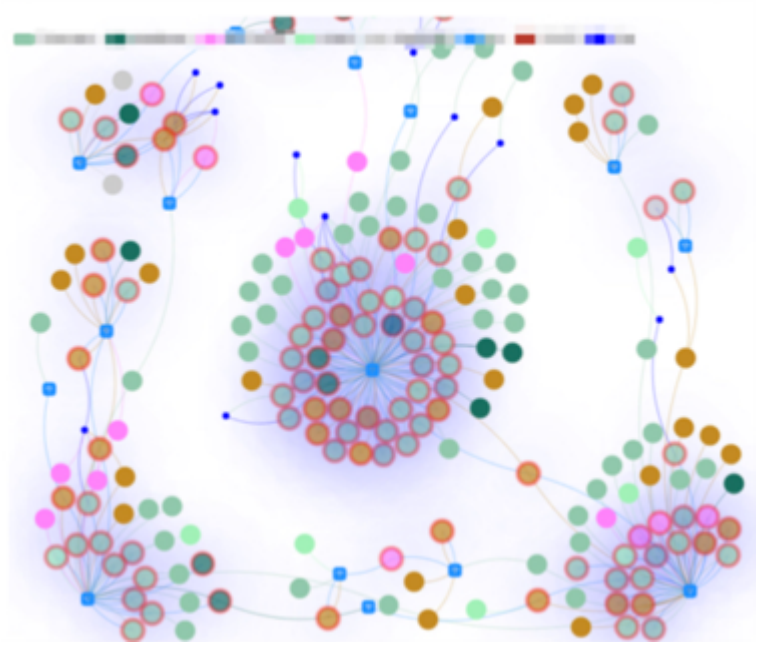


在一个黑产账户和设备网络中[8]，其中的点可以是账户、手机设备和 WIFI 网络，边是这些账户与手机设备之间的登录关系，以及手机设备和 WIFI 网络之间的接入关系。



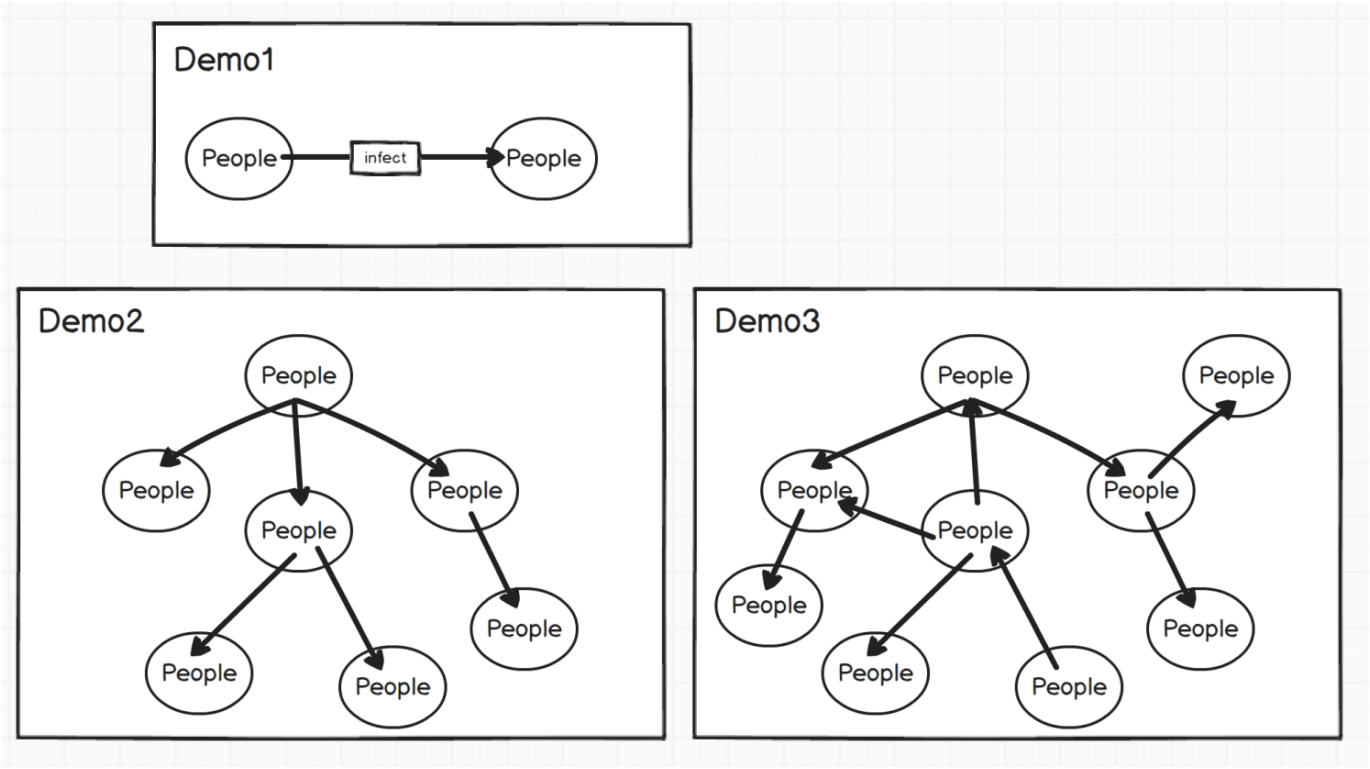
这些登录记录的网络构成了黑产群体网络的团伙作案特征。360 数科[8]、快手[9]、微信[10]、知乎[11]、携程金融这些公司都通过图技术实时（毫秒级的）识别超过百万个的黑产社群。

2022 vesoft Inc.



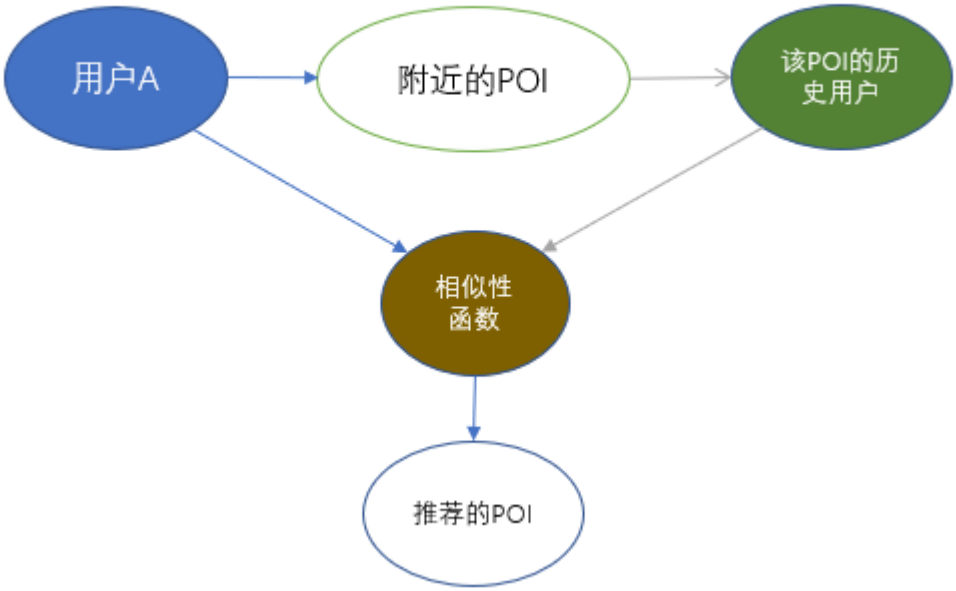
更进一步，除了时间这个维度外，我们通过添加一些地理位置信息，还能发现属性图更多的应用场景。

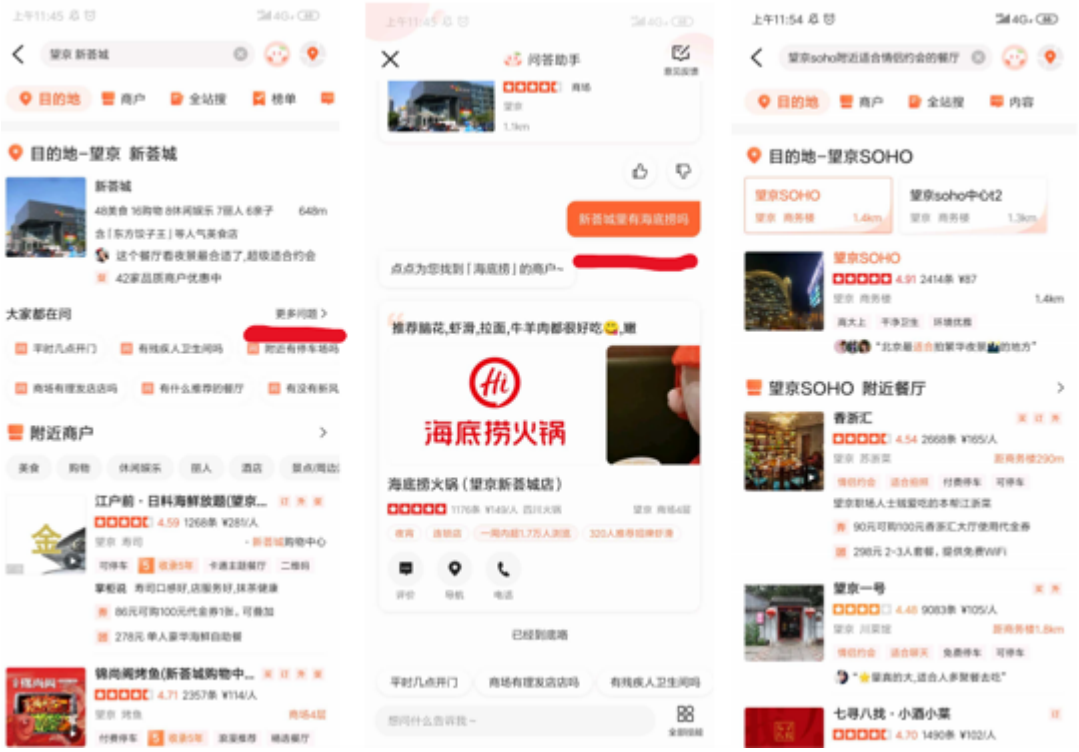
例如新冠病毒的流行病学溯源[12]，点是人物，边是人与人之间的接触；点属性为人物的身份证、发病时间等信息，边属性为人物之间发生密切接触的时间和地理位置等。为卫生防疫部门快速识别高风险人群和其行为轨迹提供帮助。



地理位置与图的结合也可以用于一些 O2O 的场景，例如基于 POI（Point-of-Interest）的实时美食推荐[13]，使得美团这类本地生活服务平台公司能在消费者在打开 APP 的时候，实时推荐出更为合适的商家。

2022 vesoft Inc.





图还可以用于更深度的知识推理，华为、vivo、OPPO、微信、美团等公司，将图用于表征底层知识关系的数据模型。

2022 vesoft Inc.

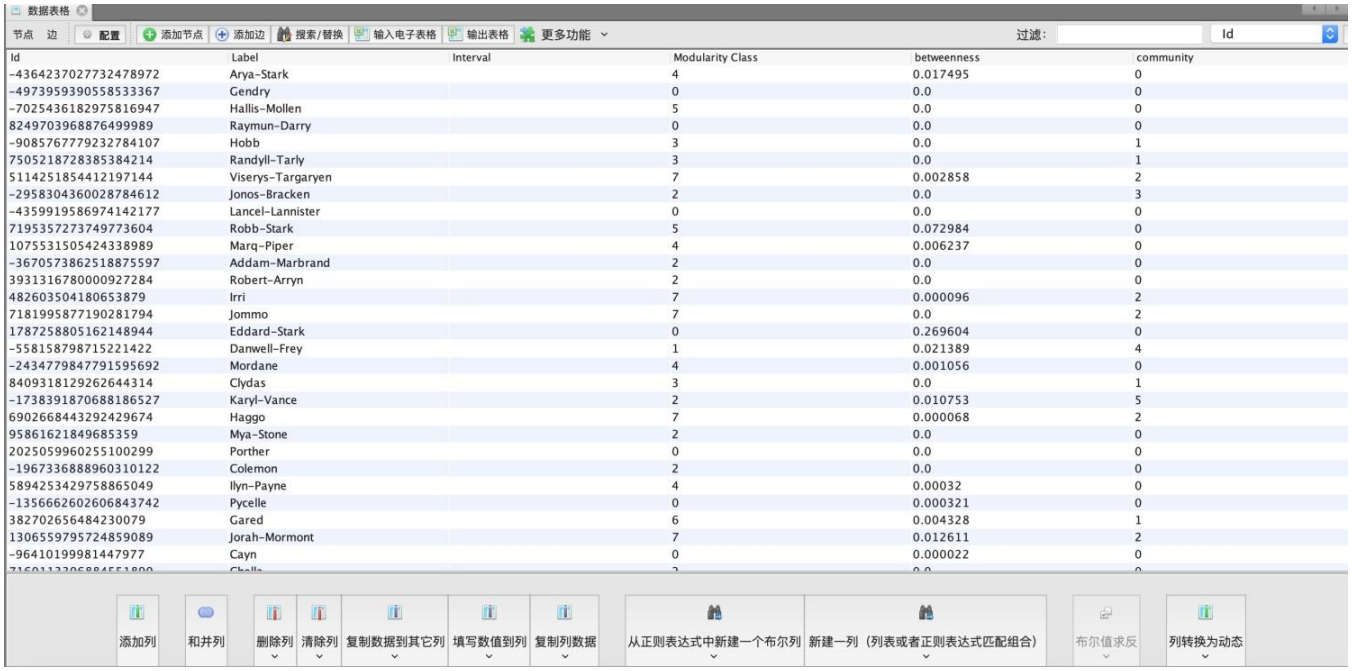
2.1.3 为什么要使用图数据库

虽然关系型数据库与 XML/JSON 等半结构类型的数据库，都可以用来描述图结构的数据模型，但是，图（数据库）不仅可以描述图结构与存储数据本身，更着眼于处理数据之间的关联（拓扑）关系。具体来说，图（数据库）有这么几个优点：

- 图是一种更直观、更符合人脑思考直觉的知识表示方式。这使得我们在抽象业务问题时，可以着眼于“业务问题本身”，而不是“如何将问题描述为数据库的某种特定结构（例如表格结构）”。

2022 vesoft Inc.

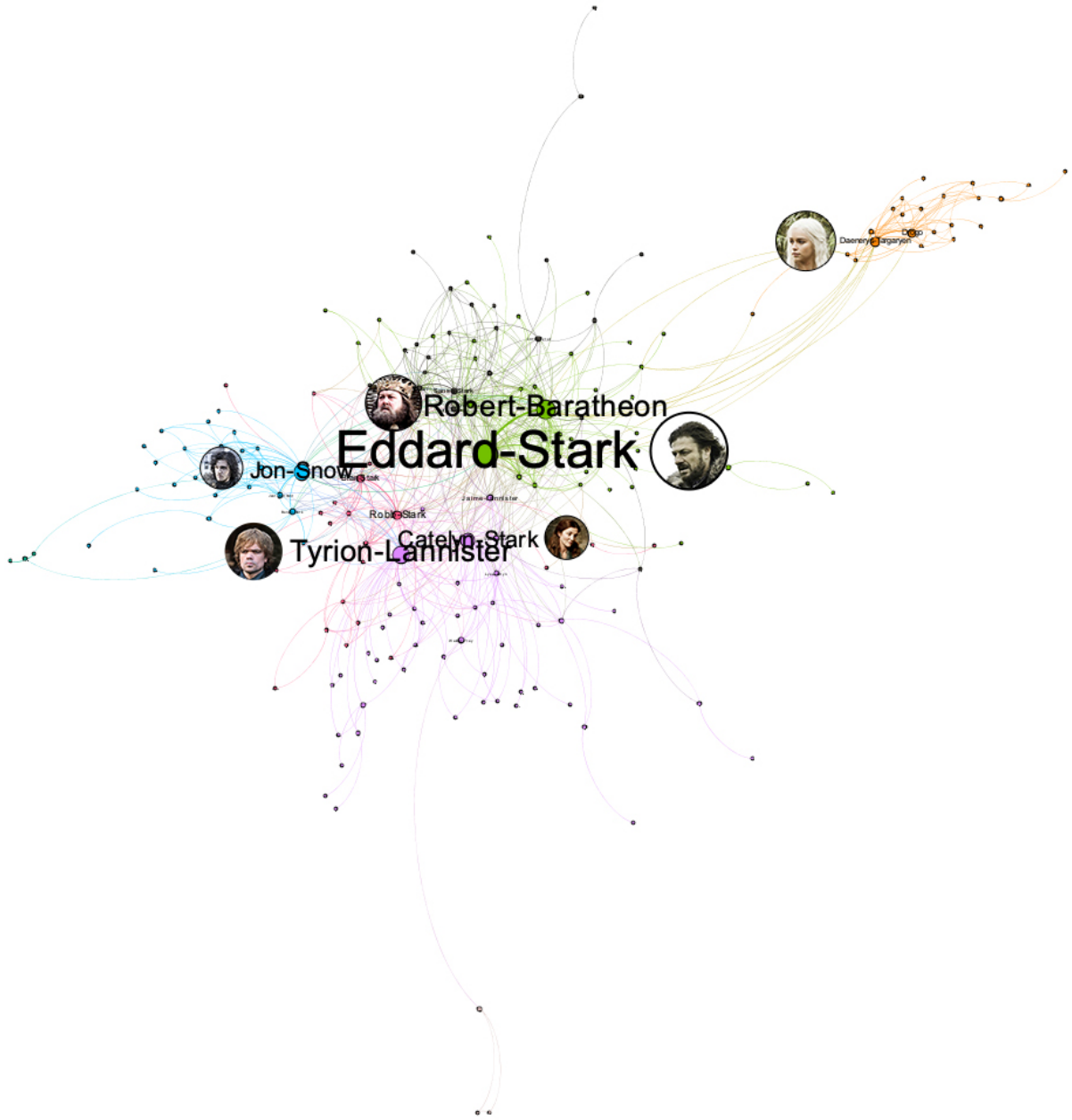
- 图更容易展现数据的特征，例如转账的路径、近邻的社区。例如，如果要分析《权力的游戏》中的人物派别关系和人物重要性，表的组织方式如下：



Id	Label	Interval	Modularity Class	betweenness	community
-4364237027732478972	Arya-Stark		4	0.017495	0
-4973959390558533367	Gendry		0	0.0	0
-7025436182975816947	Hallis-Mollen		5	0.0	0
8249703968876499989	Raymun-Darry		0	0.0	0
-9085767779232784107	Hobb		3	0.0	1
7505218728385384214	Randyll-Tarly		3	0.0	1
5114251854412197144	Viserys-Targaryen		7	0.002858	2
-2958304360028784612	Jonos-Bracken		2	0.0	3
-4359919586974142177	Lancel-Lannister		0	0.0	0
71953572737499773604	Robb-Stark		5	0.072984	0
10755315054243389 89	Marq-Piper		4	0.006237	0
-3670573862518875597	Addam-Marbrand		2	0.0	0
3931316780000927284	Robert-Arryn		2	0.0	0
4826035041806538 79	Irri		7	0.000096	2
71819958771902817 94	Jommo		7	0.0	2
17872588051621489 44	Eddard-Stark		0	0.269604	0
-558158798715221422	Danwell-Frey		1	0.021389	4
-2434779847791595692	Mordane		4	0.001056	0
84093181292626443 14	Clydas		3	0.0	1
-1738391870688186527	Karyl-Vance		2	0.010753	5
69026684432924296 74	Haggo		7	0.000068	2
958616218496853 59	Mya-Stone		2	0.0	0
20250599602551002 99	Porther		0	0.0	0
-1967336888960310122	Colemon		2	0.0	0
58942534297588650 49	Ilyn-Payne		4	0.00032	0
-1356662602606843742	Pycelle		0	0.000321	0
3827026564842300 79	Gared		6	0.004328	1
13065597957248590 89	Jorah-Mormont		7	0.012611	2
-96410199981447977	Cayn		0	0.000022	0

这显然不如下方图的组织方式直观：

2022 vesoft Inc.



特别是当某些中心节点被删除:

2022 vesoft Inc.



或者，增加一条边，可以彻底地改变整个图拓扑：



虽然只是个别数据的细微改变，图可以比表更直观地表现其中的重要而系统的信息。

2022 vesoft Inc.

- 图查询语言是针对图结构访问设计的，可以更加直观。例如，下面是一个 LDBC 中的查询示例，要求：查找某人（Person）在社交网络上发布的帖子（Posts）；查找相应的回复（Message，回复本身还会被多次回复）;发帖时间、回帖时间都满足一定条件；根据回帖数量对结果排序。

  image

  如果使用 PostgreSQL 编写查询语句：

```
--PostgreSQL
WITH RECURSIVE post_all(psa_threadid
                      , psa_thread_creatorid, psa_messageid
                      , psa_creationdate, psa_messagetype
                       ) AS (
    SELECT m_messageid AS psa_threadid
         , m_creatorid AS psa_thread_creatorid
         , m_messageid AS psa_messageid
         , m_creationdate, 'Post'
      FROM message
     WHERE 1=1 AND m_c_replyof IS NULL -- post, not comment
       AND m_creationdate BETWEEN :startDate AND :endDate
  UNION ALL
    SELECT psa.psa_threadid AS psa_threadid
         , psa.psa_thread_creatorid AS psa_thread_creatorid
         , m_messageid, m_creationdate, 'Comment'
      FROM message p, post_all psa
     WHERE 1=1 AND p.m_c_replyof = psa.psa_messageid
     AND m_creationdate BETWEEN :startDate AND :endDate
)
SELECT p.p_personid AS "person.id"
     , p.p_firstname AS "person.firstName"
     , p.p_lastname AS "person.lastName"
     , count(DISTINCT psa.psa_threadid) AS threadCount
END) AS messageCount
     , count(DISTINCT psa.psa_messageid) AS messageCount
  FROM person p left join post_all psa on (
       1=1   AND p.p_personid = psa.psa_thread_creatorid
   AND psa_creationdate BETWEEN :startDate AND :endDate
   )
 GROUP BY p.p_personid, p.p_firstname, p.p_lastname
 ORDER BY messageCount DESC, p.p_personid
 LIMIT 100;
```

  如果使用为图专门设计的图语言 Cypher 编写查询语句：

```
--Cypher
MATCH (person:Person)<-[:HAS_CREATOR]-(post:Post)<-[:REPLY_OF*0..]-(reply:Message)
WHERE post.creationDate >= $startDate AND post.creationDate <= $endDate
  AND reply.creationDate >= $startDate AND reply.creationDate <= $endDate
RETURN
  person.id, person.firstName, person.lastName, count(DISTINCT post) AS threadCount,
  count(DISTINCT reply) AS messageCount
ORDER BY
  messageCount DESC, person.id ASC
LIMIT 100
```

- 由于存储引擎和查询引擎可以针对图的结构专门设计，图的遍历（对应 SQL 中的 join）要高效得多。下图是知名产品 Neo4j 所做的一个对比[13]。

深度	关系型数据库的执行时间(s)	Neo4j的执行时间(s)	返回的记录条数
2	0.016	0.01	~2500
3	30.267	0.168	~110000
4	1543.505	1.359	~600000
5	未完成	2.132	~800000

关系数据库 vs 图数据库(多跳查询)

- 图数据库具有广泛的适用场景。例如数据集成（知识图谱）、个性化推荐、欺诈与威胁检测、风险分析与合规、身份（与控制权）验证、IT 基础设施管理、供应链与物流、社交网络研究等。

- 根据文献[14] 的统计，使用图技术最多的领域，依次是：信息技术(IT)、学术界研究、金融、工业界实验室、政府、医疗健康、国防、制药业、零售与电子商务、交通运输、电信、保险。

2022 vesoft Inc.

- 2019 年，根据 Gartner 的问卷调研，27% 的客户（500 组）在使用图数据库，20% 有计划使用。

2.1.4 RDF

受篇幅所限，本章不讨论 RDF 数据模型。

---

1. 图片来源 https://medium.freecodecamp.org/i-dont-understand-graph-theory-1c96572a1401. ↩
2. 图片来源 https://medium.freecodecamp.org/i-dont-understand-graph-theory-1c96572a1401 ↩
3. https://nebula-graph.com.cn/posts/stock-interrelation-analysis-jgrapht-nebula-graph/ ↩
4. https://nebula-graph.com.cn/posts/game-of-thrones-relationship-networkx-gephi-nebula-graph/ ↩
5. https://nebula-graph.com.cn/posts/practicing-nebula-graph-webank/ ↩
6. https://nebula-graph.com.cn/posts/meituan-graph-database-platform-practice/ ↩↩
7. https://zhuanlan.zhihu.com/p/90635957 ↩
8. https://nebula-graph.com.cn/posts/graph-database-data-connections-insight/ ↩↩
9. https://nebula-graph.com.cn/posts/kuaishou-security-intelligence-platform-with-nebula-graph/ ↩
10. https://nebula-graph.com.cn/posts/nebula-graph-for-social-networking/ ↩
11. https://mp.weixin.qq.com/s/K2QinpR5Rplw1teHpHtf4w ↩
12. https://nebula-graph.com.cn/posts/detect-corona-virus-spreading-with-graph-database/ ↩
13. https://nebula-graph.com.cn/posts/meituan-graph-database-platform-practice/ ↩↩
14. https://arxiv.org/abs/1709.03188 ↩

最后更新: June 20, 2023

2022 vesoft Inc.

2.2 图数据库的市场概况

既然已经讨论了什么是图，接下来让我们进一步认识基于图论和属性图模型发展起来的图数据库。

不同的图数据库在术语方面可能会略有不同，但是归根结底都是在讲点、边和属性。至于更多的功能，例如标签、索引、约束、TTL、长任务、存储过程和UDF等这些高级功能，在不同图数据库中，会存在明显的差异。

图数据库用图来存储数据，而图是最接近高度灵活、高性能的数据结构之一。图数据库是一种专门用于存储和检索庞大信息网的存储引擎，它能够高效地将数据存储为点和边，并允许对这些点边结构进行高性能的检索和查询。我们也可以为这些点和边添加属性。

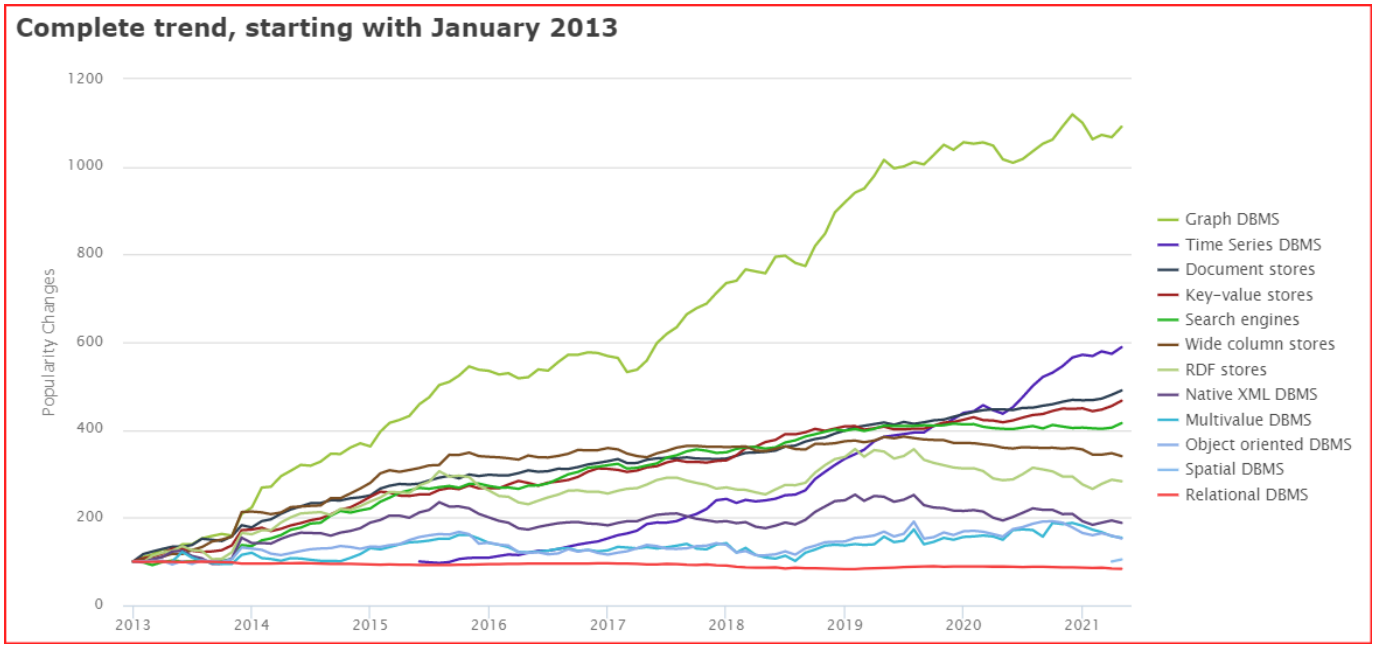
图数据库几乎适用于存储所有领域的数据。因为在几乎所有领域中，事物之间都是由某种相关联的。图数据库支持存储实体之间的丰富关系，并且能够将这些关系完美地呈现出来，而无需像其他建模方式那样，将关系也当成实体存储。因此图数据库能够以最接近对数据直观认知的形式存储数据。

2.2.1 三方机构的统计和预测

DB-Engines 的统计

根据世界知名的数据库排名网站DB-Engines.com的统计，图数据库至2013年以来，一直是“增速最快”的数据库类别[1]。

该网站根据一些指标来统计每种类别的数据库的流行度变化趋势，这些指标包括基于Google等搜索引擎的收录和趋势情况、主要IT技术论坛和社交网站上讨论的技术话题、招聘网站的职位变化等。该网站共收录了371种数据库产品，并分为12个类别。这12个类别中，图数据库这种类别的增速远远快于其他任何的类别。

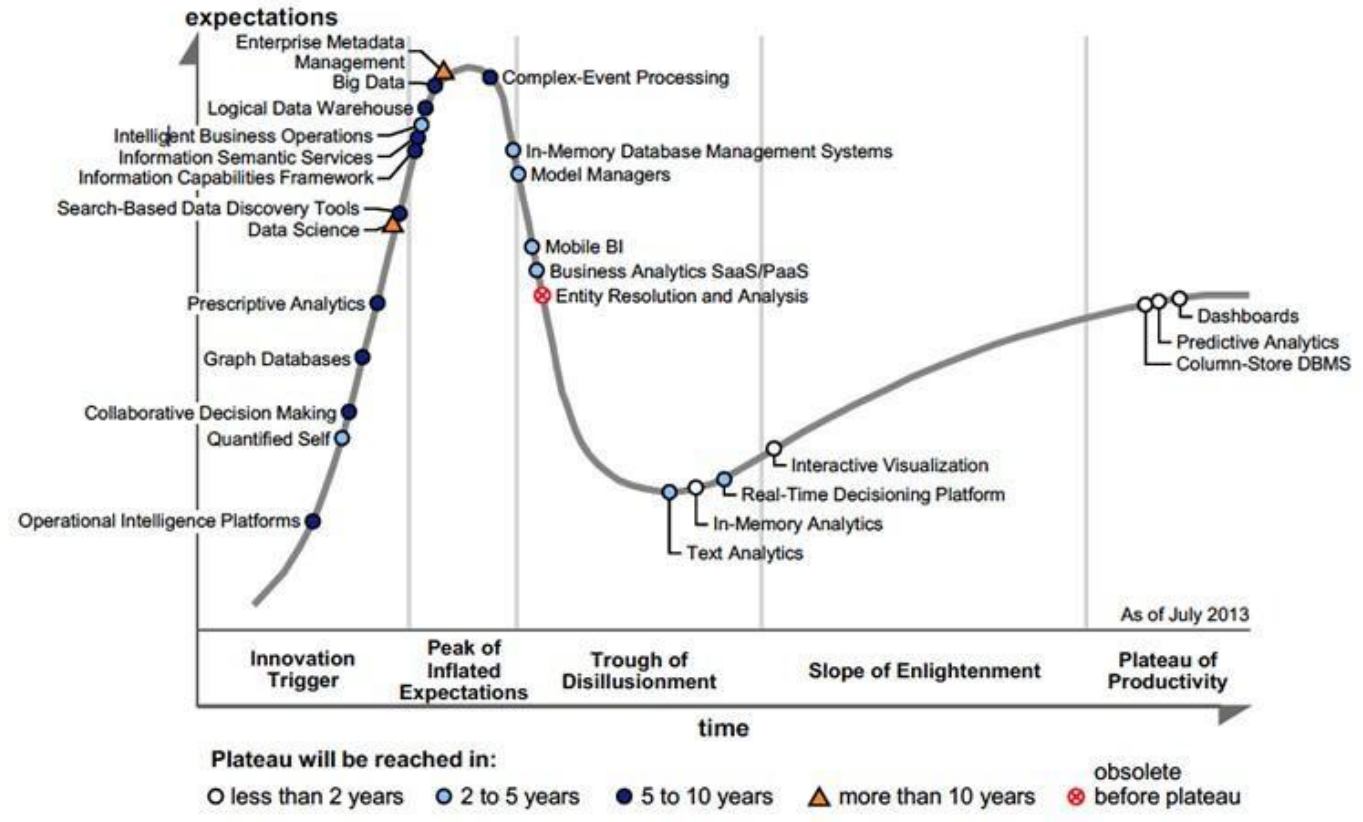


Gartner 的预测

世界顶级智库Gartner早在2013年之前[2]，就将图数据库作为主要的"商业智能与分析技术趋势"。在那个时候，Big Data正火热的如日中天，数据科学家更是炙手可热的职位。

2022 vesoft Inc.

Figure 1. Hype Cycle for Business Intelligence and Analytics, 2013



BI = business intelligence; DBMS = database management system; SaaS = software as a service; PaaS = platform as a service

直到最近，图数据库及相关的图技术依旧是"2021年十大数据与分析趋势"[3]:

2022 vesoft Inc.



趋势八：图技术使一切产生关联（**Graph Relates Everything**）

图技术已成为许多现代数据和分析能力的基础，能够在不同的数据资产中发现人、地点、事物、事件和位置之间的关系。数据和分析领导者依靠图技术快速回答需要在了解情况并理解多个实体之间的联系和优势的性质后才能回答的复杂业务问题。

Gartner预测，到2025年图技术在数据和分析创新中的占比将从2021年的10%上升到80%。该技术将促进整个企业机构的快速决策。

可以注意到，Gartner 的预测比较好的吻合了 DB-Engines 的统计结论。技术的进步并不是完全线性的，通常会有一段快速发展的泡沫期，然后进入一段平台期，之后由于新的技术的出现产生新一轮的泡沫期，再经历一段平台期。以此往复螺旋形的循环发展。

2022 vesoft Inc.

对于市场规模的预测

根据 verifiedmarketresearc[4], fnfresearch[5], marketsandmarkets[6], 以及 gartner[7] 等智库的统计和预测，图数据库市场（包括云服务）规模在2019年大约是8亿美元，将在未来6年保持25%左右的年复合增长(CAGR)至 30-40 亿美元，这大约对应于全球数据库市场 5-10% 的市场份额。



2.2.2 市场参与者

（第一代）图数据库的先行者 **Neo4j**

虽然在 1970 年代，人们已经提出了一些类似于"图”的数据模型和产品原型（例如 CODASYL[8])和相应的图语言 G/G+ 语言[9]。但真正能够让“图数据库”这个概念流行起来，不得不说到这个市场最主要的先行者 Neo4j，甚至(标签)属性图和图数据库这两个主要术语就是 Neo4j 最早提出并实践的。

> 本小节关于**Neo4j**和其创造的图查询语言**Cypher**的历史内容主要摘录自 **ISO WG3** 的工作论文**"An overview of the recent history of Graph Query Languages"**[10] 和[9]，本书作者根据最新两年的进展有删减和更新。

> 关于图查询语言**(Graph Query Language，GQL)** 和国际标准的制定

熟悉数据库的读者可能都知道结构化查询语言SQL。通过使用SQL，人们以接近自然语言的方式访问数据库。在 SQL 被广泛采用和标准化之前，关系型数据库的市场是非常碎片和割裂的——各家厂商的产品都有完全不同的接入访问方式，数据库产品自身的开发人员、数据库产品周边工具的开发人员、数据库最终的使用人员，都不得不学习各个厂商的完全不同的产品，在不同产品之间迁移极其困难。当1989年SQL-89标准被制定后，整个关系型数据库的市场快速收敛到SQL-89上。这大大降低了上述各种人员的学习曲线。

类似的，在图数据库领域，图语言(GQL)承担了类似于SQL的作用，是一种用户与图数据库主要的交互方式。但不同于SQL-89这种国际标准，GQL还没有任何国际标准。目前有两种主流的图语言：

Neo4j的Cypher (及其后续——ISO正在制定过程中的 GQL-standard 草案)和Apache TinkerPop的Gremlin。前者通常被称为声明式语言(Declarative query language)——也即用户只需要告诉系统“要什么”，而不管“怎么做”；后者通常被称为命令式语言(Imperative query language)，用户会显式地指定系统的操作。

GQL国际标准正在制定过程中。

2022 vesoft Inc.

年表简述

- 2000 年，Neo4j 的创始人产生将数据建模成网络（network）的想法。
- 2001 年，Neo4j 开发了最早的核心部分代码。
- 2007 年，Neo4j 开始以一个公司的方式运作。
- 2009 年，Neo4j 团队借鉴 XPath 作为图查询语言，Gremlin[11]最初也是基于这个想法。
- 2010 年，Neo4j 的员工 Marko Rodriguez 采用术语属性图（Property Graph）来描述 Neo4j 和 Tinkerpop / Gremlin 的数据模型。
- 2011 年，第一个公开发行版本 Neo4j 1.4; 并发布了Cypher的第一个版本。
- 2012 年，Neo4j 1.8 为 Cypher 增加写入图的能力。Neo4j 2.0 增加了标签和索引，Cypher 成为一种声明式的语言。
- 2015 年，Neo4j 将 Cypher 开源为 openCypher。
- 2017 年，ISO WG3 工作组开始讨论如何将属性图查询能力引入 SQL。
- 2018 年 12 月，从 Neo4j 3.5 开始其核心部分转为闭源。
- 2019 年, ISO 正式立项两个项目(ISO/IEC JTC 1 N 14279和ISO/IEC JTC 1/SC 32 N 3228），启动关于图数据库语言国际标准的制定工作。
- 2021 年，Neo4j 完成 F 轮 3.25 亿美元的融资，是整个数据库（包括关系型）历史上最大一轮融资。

NEO4J 的早期历史

Neo4j 和属性图这种数据模型，最早构想于 2000 年。Neo4j 的创始人们当时在开发一个媒体管理系统，所使用的数据库的 schema 经常会发生重大变化。为了支持这种灵活性，Neo4j 的联合创始人 Peter Neubauer，受到 Informix Cocoon 的启发，希望将系统能够建模为一种概念相互连接的网络。印度理工学院孟买分校的一群研究生们实现了最早的原型。Neo4j 的联合创始人 Emil Eifrém 和这些学生们花了一周的时间，将 Peter 最初的想法扩展成为一个更抽象的模型：节点通过关系连接，key-value 作为节点和关系的属性。这群人开发了一个 Java API 来和这种数据模型交互，并在关系型数据库之上实现了一个抽象层。

虽然这种网络模型极大的提高了生产力，但是性能一直很差。所以 Neo4j 联合创始人 Johan Svensson 花了不少精力，为这种网络模型实现了一个原生的数据管理系统。这个就成为了 Neo4j。在最初的几年，Neo4j 作为一个内部产品很成功。在 2007 年，Neo4j 的知识产权转移给了一家独立的数据库公司。

在 Neo4j 的第一个公开发行版中（Neo4j 1.4，2011 年），数据模型由节点和有类型的边构成，节点和边都有 key-value 组成的属性。Neo4j 的早期版本没有任何的索引，应用程序只能从根节点开始自己构造查询结构（search structure）。因为这样对于应用程序非常笨重，Neo4j 2.0（2013.12）引入了一个新概念——点上的标签（label）。基于点标签，Neo4j 可以为一些预定义的节点属性建立索引。

"节点"、"关系"、"属性"、"关系只能有一个标签"、"节点可以有零个或者多个标签"，以上这些概念构成了 Neo4j 属性图的数据模型定义。随着后来增加的索引功能，让 Cypher 成为了与 Neo4j 交互的主要方式。因为这样应用程序开发者只需要关注于数据本身，而不是上段提到的那个开发者自己构建的查询结构（search structure）。

GREMLIN 的创造

Gremlin是基于Apache TinkerPop开发的图语言，其风格接近于一连串的函数（过程）调用。最初 Neo4j 的查询方式是通过 Java API。应用程序可以将查询引擎作为库(library)嵌入到应用程序中，然后使用 API 来查询图。

就在这段时间，NOSQL 这个概念开始出现。NOSQL 型的数据库引擎一般用 REST 和 HTTP 来交互和查询。Neo4j 的早期员工 Tobias Lindaaker、Ivarsson、Peter Neubauer 、Marko Rodriguez用 XPath 作为图查询，Groovy 提供循环结构，分支和计算（等图灵完毕的功能）。 这个就是 Gremlin 最初的原型。 2009 年 11 月发布了第一个版本。

后来，Marko 发现同时用两种不同的解析器（XPath 和 Groovy）有很多问题，就将 Gremlin 改为基于 Groovy 的一种领域特定语言（DSL）。

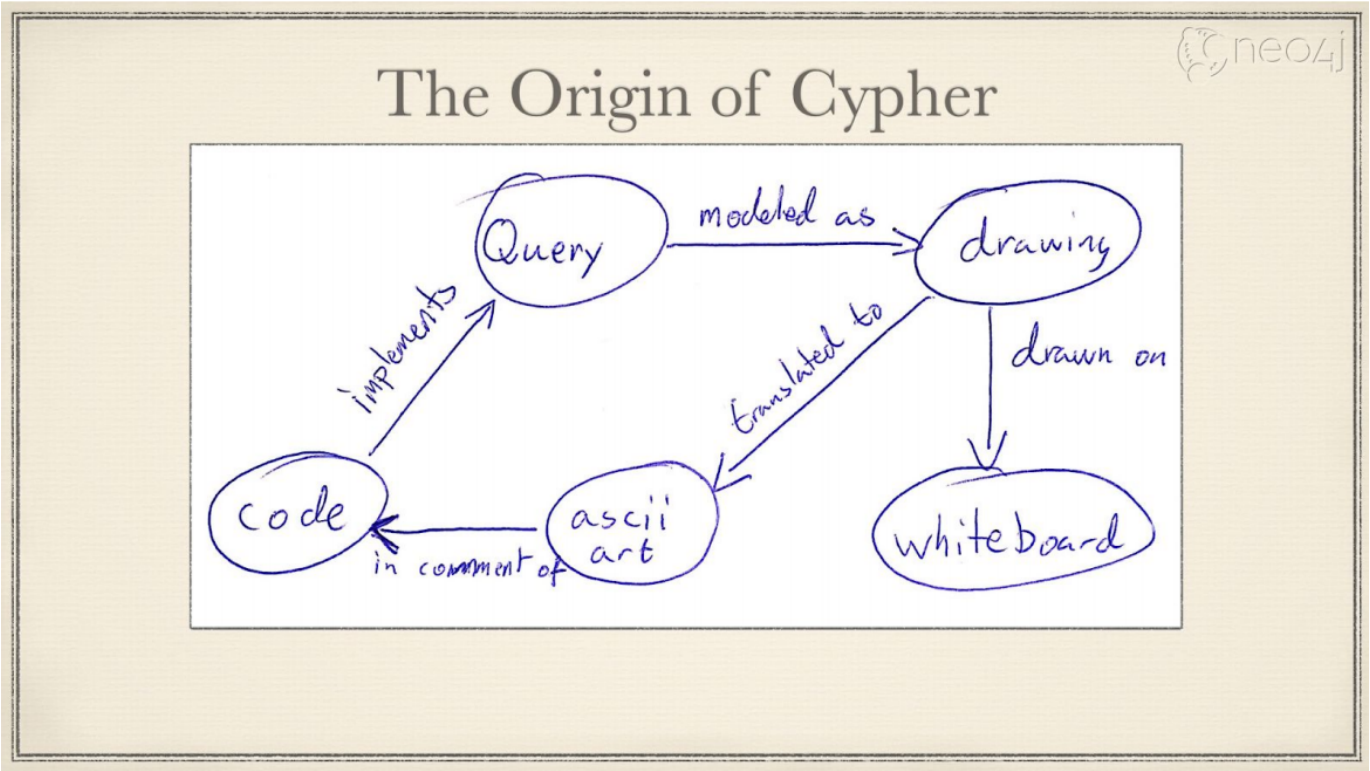
CYPHER 的创造

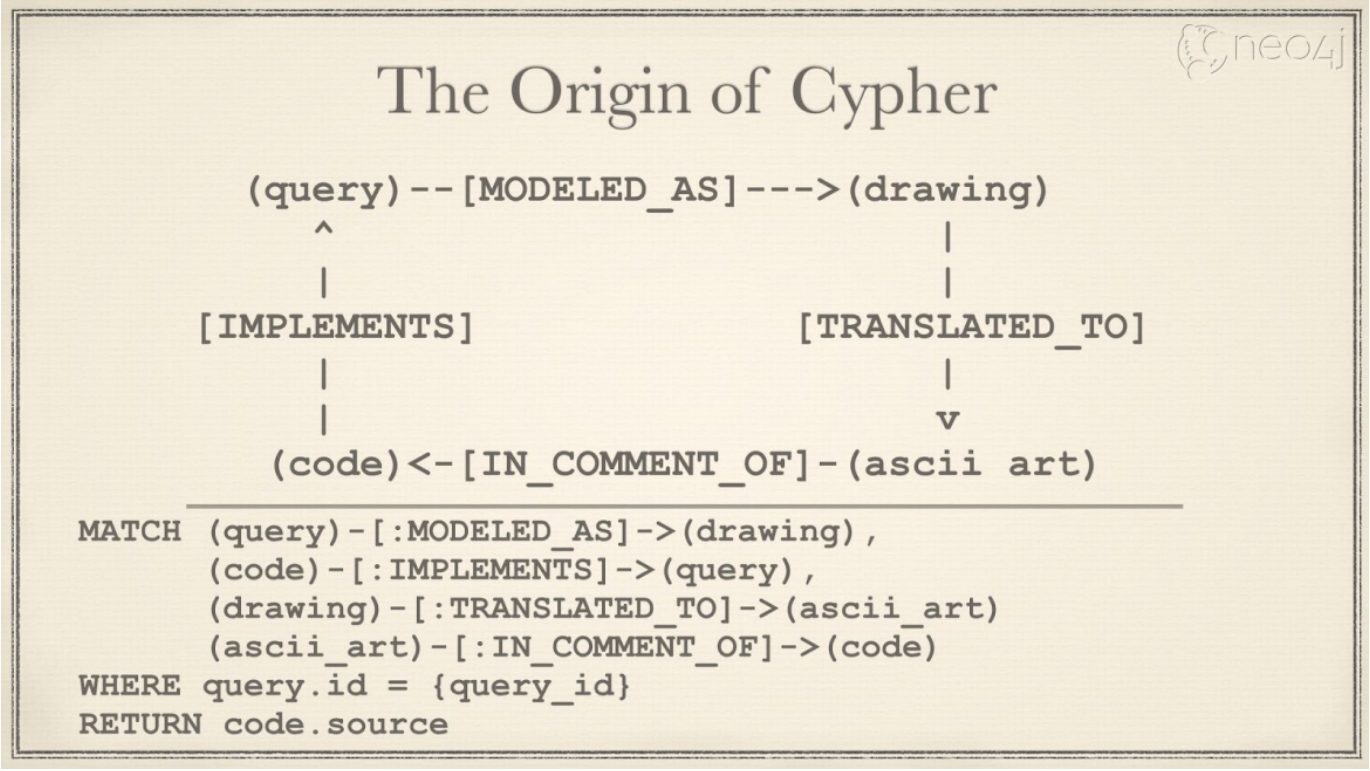
Gremlin 和 Neo4j 的 Java API 一样，最初用于表达如何查询数据库的一种过程（Procedural）。它允许更短的语法来表达查询，也允许通过网络远程访问数据库。Gremlin 这种过程式的特性，需要用户知道如何采用最好的办法查询结果，这样对于应用程序开发人员来说仍旧有负担。与此同时，在过去30年中，声明式语言 SQL 取得了极大的成功：SQL 可以将“获取数据的声明方式”和“引擎如何获取数据”相分开，所以 Neo4j 的工程师们希望开发一种声明式的图查询语言。

2010 年，Andrés Taylor 作为工程师加入 Neo4j。受 SQL 启发，他启动了一个项目来开发图查询语言，而这种新语言于 2011 年 Neo4j 1.4 发布，这种新语言就是如今大多数图查询语言的先祖——Cypher 。

Cypher 的语法基础，是用 "ASCII艺术(ASCII art)" 来描述图模式。这种方式最初来源于 Neo4j 工程师团队在源代码中评注如何描述图模式。可以看下图的例子:

2022 vesoft Inc.





ASCII art 简单说，就是如何用可打印文本来描述点和边。Cypher 文本用 () 表示点，`-[]->` 表示边。`(query)-[modeled as]->(drawing)` 来表示 `起点 query`， `终点 drawing`， `边 modeled as`，这样一个最简单的图关系(也可以称为图模式)。

Cypher 第一个版本实现了对图的读取，但是需要用户说明从哪些节点开始查询。只有从这些节点开始，才可以支持图的模式匹配。

在后面的版本，2012 年 10 月发布的 Neo4j 1.8 中，Cypher 增加了修改图的能力。但查询还是需要指明从哪些节点开始。

2013 年 12 月，Neo4j 2.0 引入了 label 的概念，label 本质上是个索引。这样，查询引擎就可以利用索引，来选择模式所匹配到的节点，而不需要用户指定开始查询的节点。

2022 vesoft Inc.

随着 Neo4j 的普及，Cypher 有着广泛的开发者群体，在各行各业的得到广泛的使用。至今仍是最受欢迎的图查询语言。

2015 年 9 月，Neo4j 发起成立了 openCypher Implementors Group（oCIG），将 Cypher 开放为 openCypher，通过开源的方式来治理和推进语言自身的演化。

后续

Cypher 启发了一系列后续的图查询语言，包括

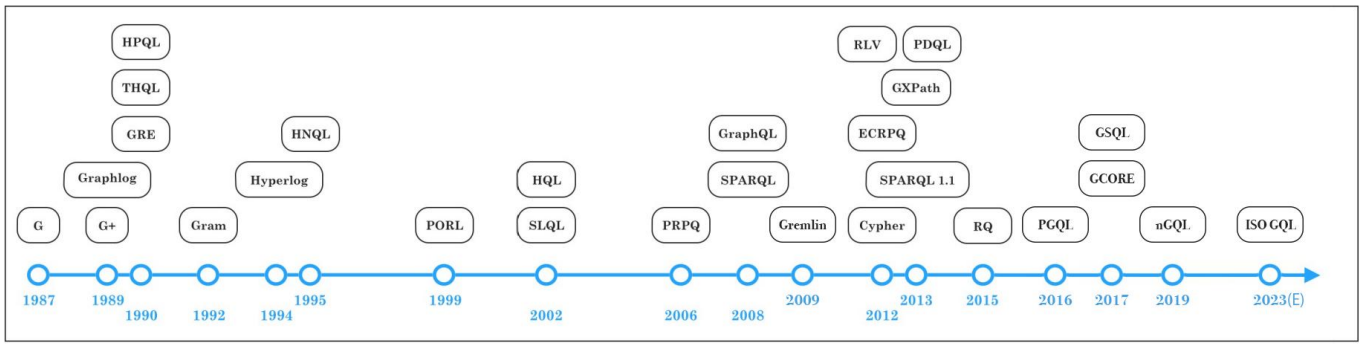
2015 年，Oracle 发布图引擎PGX使用的图语言PGQL。

2016 年，Linked Data Benchmarking Council, LDBC 是一个行业知名的图性能基准评测机构。LDBC 发布 G-CORE

2018 年，基于 Redis 的图库(library) RedisGraph 采用 Cypher 作为其图语言

2019 年，国际标准组织 ISO 启动两个项目，基于 openCypher, PGQL, GSQL[12], and G-CORE 等现有业界成果，启动图语言国际标准的制定过程

2019 年，Nebula Graph 以 openCypher 为基础发布其扩展的图语言 Nebula Graph Query Language, nGQL。



分布式图数据库

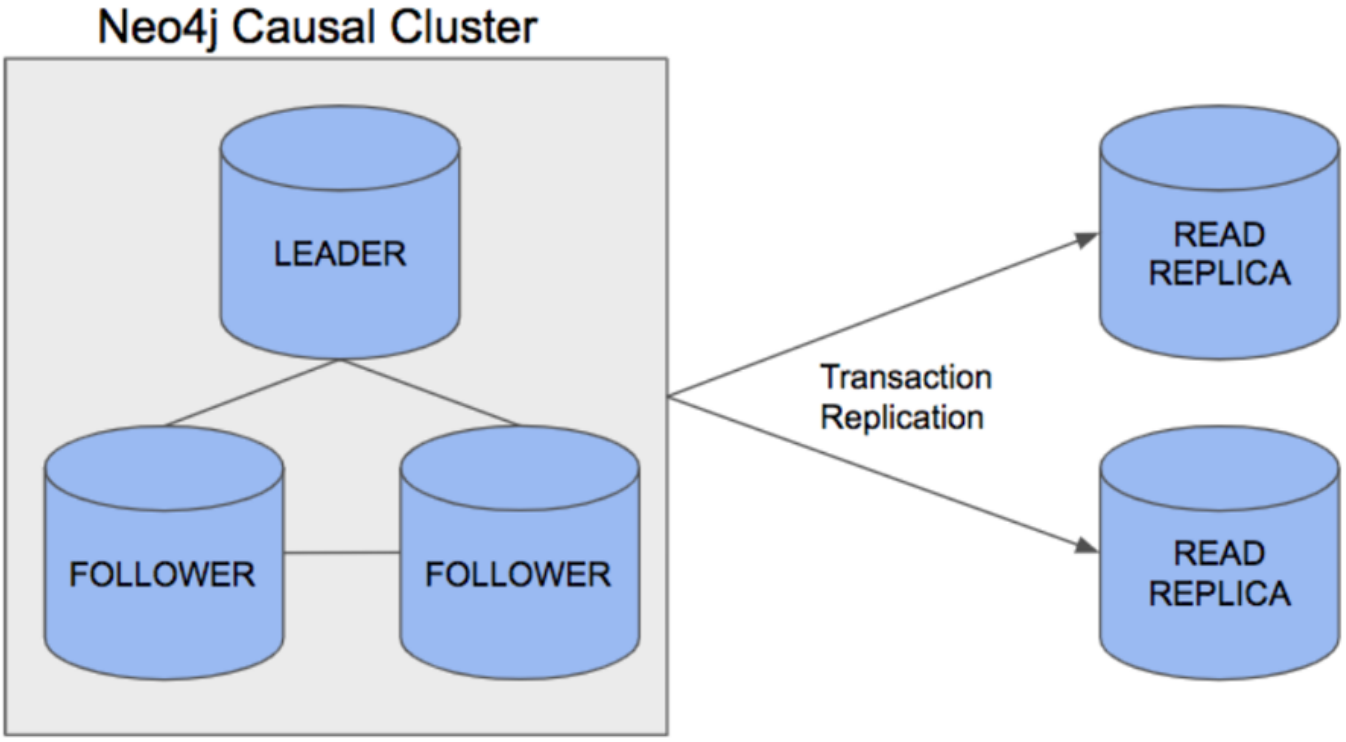
大约 2005-2010 年，随着 Google 云计算"三驾马车"的发布，各种分布式的架构开始越来越流行，其中就包括以开源方式运作的 Hadoop 和 Cassandra 等。这里包括几个方面的影响:

1. 由于数据量和计算量越来越大，相比于单机(例如 Neo4j)或者小型机这种方案，分布式系统的技术和成本优势更加明显；而同时，分布式系统使得应用程序在访问这成千上万台机器时，就如同访问本地的系统一样，不需要在代码层面进行过多改造；
2. 开源方式使得更多的人（包括代码开发者、数据科学家、产品经理等）以更加低成本和有效的方式参与新兴的技术，并反馈给社区。

严格说，Neo4j 也提供了不少的分布式的能力，但都和业界意义上的（对等、分片的）分布式系统有较大的不同:

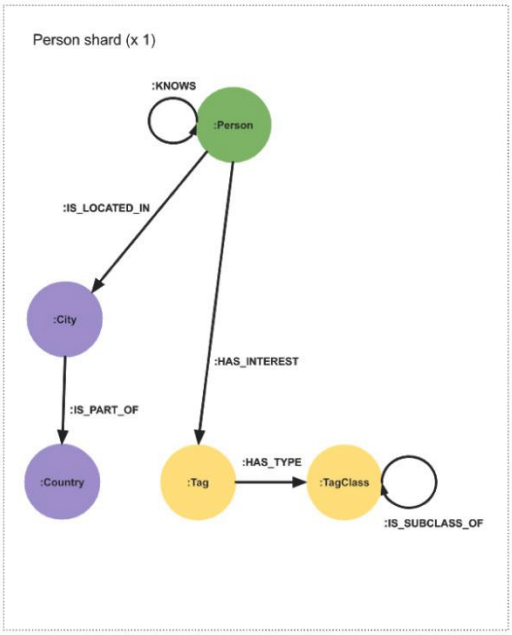
- Neo4j 3.X 要求全量数据必须存放在单机中。虽然其也提供多机之间(Master-slave/slave)做全量复制和高可用，但数据不可切分为不同子图存放。

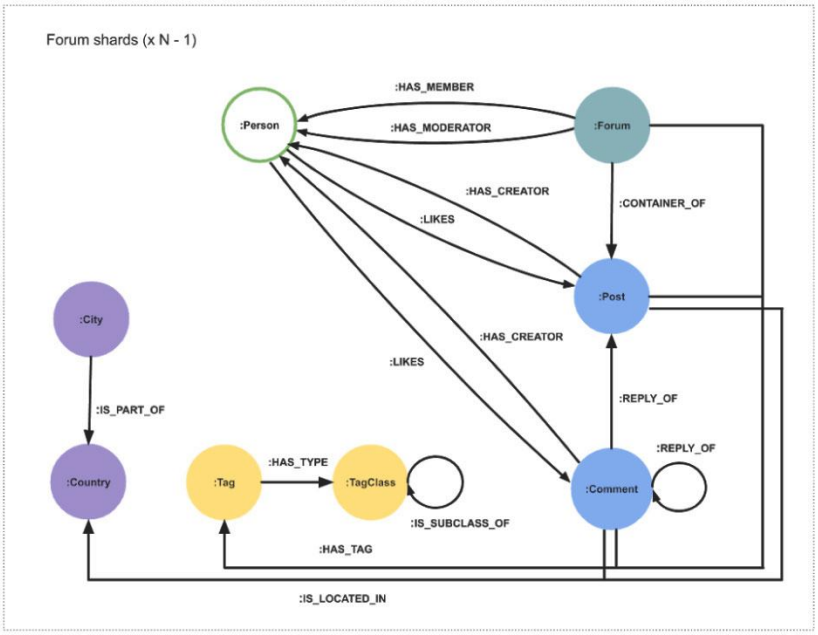
2022 vesoft Inc.



Cluster architecture

- Neo4j 4.X 允许在不同机器上各存放一部分数据（子图），然后在应用层需通过一定方式拼装后(其称为编织 Fabric)[13]，将读写分发到各个机器上。这种做法需要应用层代码有大量的参与和工作。例如，设计如何把不同子图应该放置在哪些机器上，如何将从各机器获取的部分结果重新编织为最终的结果。



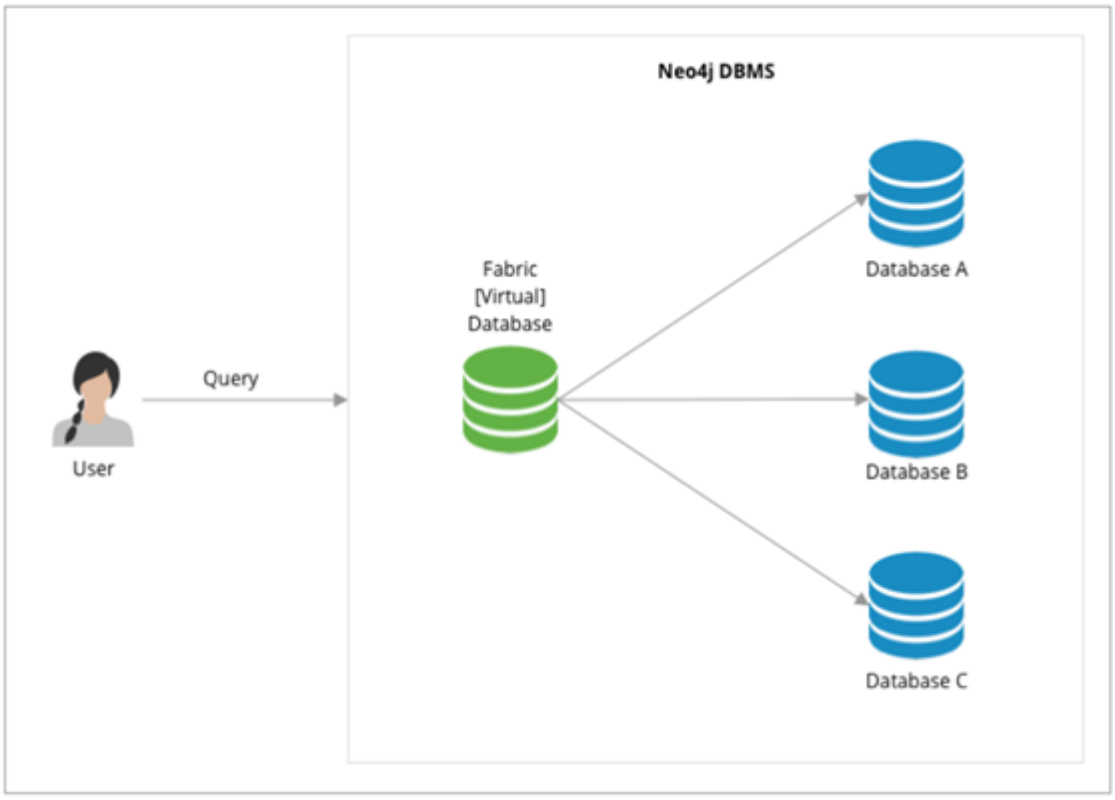


其语法风格大体是

```
USE graphA  # S1.1 从 Shard A 读
MATCH (movie:Movie)
Return movie.title AS title
    UNION   # S2. 在代理服务器 Join 结果
USE graphB  # S1.2 从 Shard B 读
```


2022 vesoft Inc.

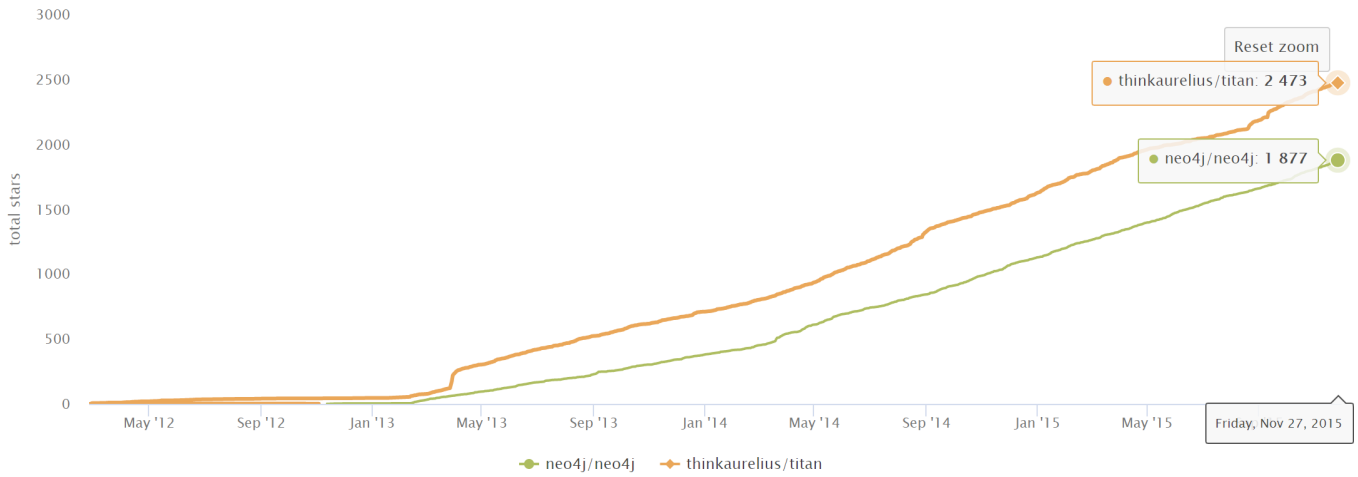
```
MATCH (move:Movie)
RETURN movie.title AS title
```



第二代（分布式）图数据库：**TITAN** 和其后继者 **JANUSGRAPH**

2011 年，Aurelius 公司成立，致力于开发一个开源的分布式图数据库 Titan[14]。到 2015 年 Titan 的第一个正式版发布，Titan 后端可以支持多种主流的分布式存储架构（例如 Cassandra, HBase, Elasticsearch, BerkeleyDB），并可以复用 Hadoop 生态的诸多便利，前端以 Gremlin 为统一的查询语言。对于程序员使用、开发和社区参与都很方便。大规模的图，可以分片后存放在 HBase 或者 Cassarndar上(这些当时都已经是相对成熟的分布式存储方案)，Gremlin 语言虽然略微冗长但相对功能完备。整个方案在当时(2011-2015)体现了不错的竞争力。

下图显示了 2012年 - 2015 年，Titan 和 Neo4j 在 GitHub.com 上 star 的增长情况。



2015 年 Aurelius(Titan) 被 DataStax 收购，这之后 Titan 逐渐转变为一个闭源的商业产品 (DataStax Enterprise Graph)。

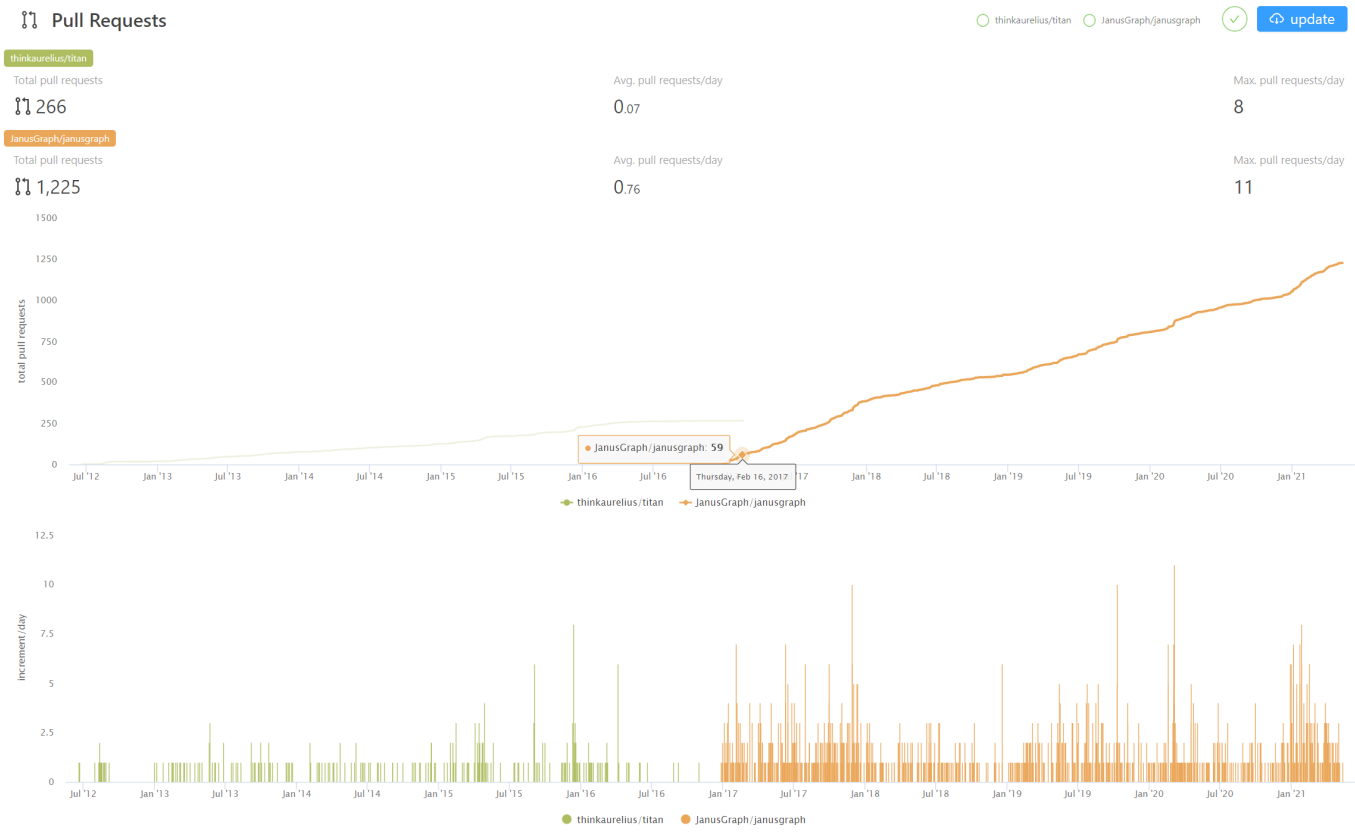
在 Aurelius(Titan) 被收购后，市场对于开源分布式的图数据库一直仍有比较强烈的需求，而当时市场上成熟和活跃的产品并不多。大数据时代，数据仍在远快于摩尔定律的速度，源源不断的产生。Linux 基金会以及一些技术巨头(Expero, Google, GRAKN.AI, Hortonworks, IBM and

2022 vesoft Inc.

Amazon) 在2017年，复制并分叉(fork)了原有的Titan项目，并启动为一个新项目 JanusGraph[15]。之后大多数的社区工作，包括开发、测试、发布和推广都逐步转移到了新的 JanusGraph。

下图显示了两个项目2012-2021年日常代码提交(pull request)的变化情况，可以观察到几点:

1. 即使 Aurelius(Titan) 2015 年被收购后，其开源代码仍有一定的活跃度(左侧)，但增速已经明显放缓。这体现了社区的力量。
2. 新项目 JanusGraph 项目在 2017 年 1 月启动后，其社区迅速活跃起来，短短一年时间就超越了 Titan 过去 5 年累计的 pull request 数量。而与此同时，Titan 开源项目就此停滞。



同期知名产品 **ORIENTDB, TIGERGRAPH, ARANGODB,** 和 **DGRAPH**

此后更多的厂商加入整个市场，除了由Linux基金会托管的 JanusGraph，还有一些由商业公司主导开发的分布式图数据，各方采用的数据模型和访问方式也有明显的不同。 本文不做一一介绍，仅简单列出主要区别。

厂商名	创立时间	核心产品名	开源协议	数据模型	查询语言
OrientDB LTD (2017 年 被 SAP 收购)	2011	OrientDB	开源	文档 + KV + 图	OrientDB SQL (基于SQL扩展的图能力)
GraphSQL (后改名 TigerGraph)	2012	TigerGraph	商业版本	图(分析)	GraphSQL (类SQL风格)
ArangoDB GmbH	2014	ArangoDB	Apache License 2.0	文档 + KV + 图	AQL (同时操作文档, KV 和图)
DGraph Labs	2016	DGraph	Apache Public License 2.0 + Dgraph Community License	原 RDF，后改为 GraphQL	GraphQL+-

2022 vesoft Inc.

传统巨头微软、亚马逊和甲骨文纷纷入场

除了聚焦于图产品的厂商外，传统巨头也纷纷进入这个领域。

Microsoft Azure Cosmos DB[16] 是一个在微软云上的多模数据库云服务，可以提供SQL、文档、图、key-value等多种能力； Amazon AWS Neptune[17] 是一种由 AWS 提供图数据库云服务, 可以提供属性图和 RDF 两种数据模型； Oracle graph[18] 是关系型数据库巨头 Oracle 在图技术与图数据库方向的产品。

新一代开源分布式图数据库 **NEBULA GRAPH**

在下一章，我们将正式介绍新一代开源分布式图数据库 Nebula Graph。

---

1. https://db-engines.com/en/ranking_categories ↩
2. https://www.yellowfinbi.com/blog/2014/06/yfcommunitynews-big-data-analytics-the-need-for-pragmatism-tangible-benefits-and-real-world-case-165305 ↩
3. https://www.gartner.com/smarterwithgartner/gartner-top-10-data-and-analytics-trends-for-2021/ ↩
4. https://www.verifiedmarketresearch.com/product/graph-database-market/ ↩
5. https://www.globenewswire.com/news-release/2021/01/28/2165742/0/en/Global-Graph-Database-Market-Size-Share-to-Exceed-USD-4-500-Million-By-2026-Facts-Factors.html ↩
6. https://www.marketsandmarkets.com/Market-Reports/graph-database-market-126230231.html ↩
7. https://www.gartner.com/en/newsroom/press-releases/2019-07-01-gartner-says-the-future-of-the-database-market-is-the ↩
8. https://www.amazon.com/Designing-Data-Intensive-Applications-Reliable-Maintainable/dp/1449373321 ↩
9. I. F. Cruz, A. O. Mendelzon, and P. T. Wood. A Graphical Query Language Supporting Recursion. In Proceedings of the Association for Computing Machinery Special Interest Group on Management of Data, pages 323–330. ACM Press, May 1987. ↩↩
10. "An overview of the recent history of Graph Query Languages". Authors: Tobias Lindaaker, U.S. National Expert.Date: 2018-05-14 ↩
11. Gremlin是基于Apache TinkerPop开发的图语言(https://tinkerpop.apache.org/)。 ↩
12. https://docs.tigergraph.com/dev/gsql-ref ↩
13. https://neo4j.com/fosdem20/ ↩
14. https://github.com/thinkaurelius/titan ↩
15. https://github.com/JanusGraph/janusgraph ↩
16. https://azure.microsoft.com/en-us/free/cosmos-db/ ↩
17. https://aws.amazon.com/cn/neptune/ ↩
18. https://www.oracle.com/database/graph/ ↩

最后更新: June 20, 2023

2022 vesoft Inc.

2.3 相关技术

本节主要介绍两个和分布式图数据库关系密切的领域，数据库方面和图技术方面。

2.3.1 数据库方面

关系型数据库

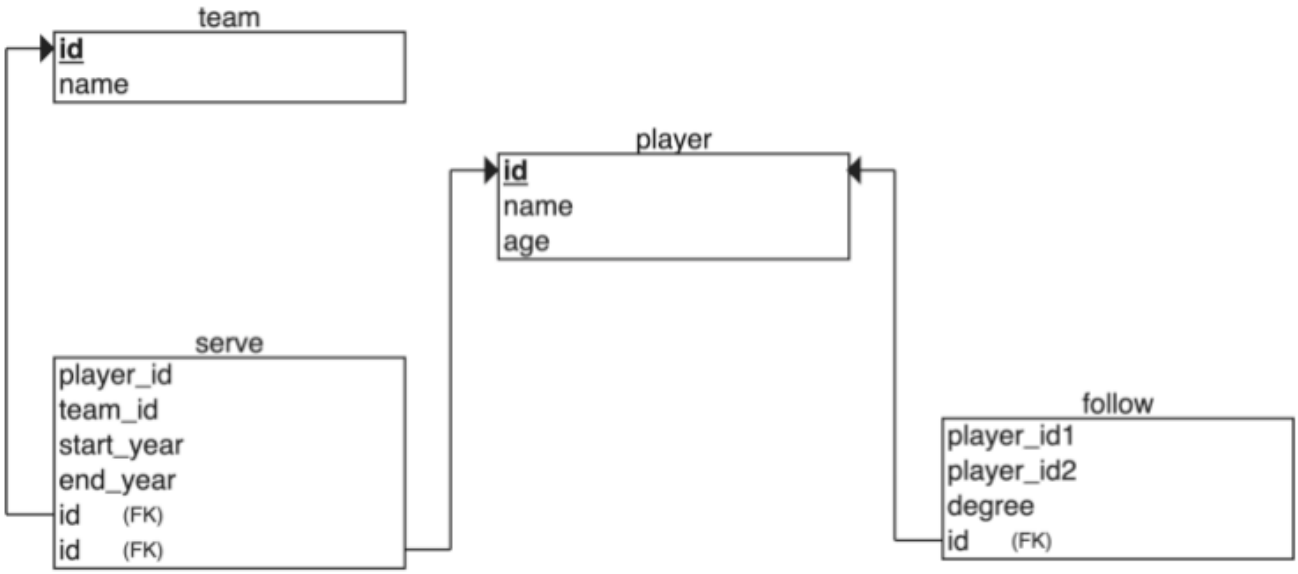
关系型数据库，是指采用了关系模型来组织数据的数据库。关系模型为二维表格模型，一个关系型数据库就是由二维表及其之间的联系所组成的一个数据组织。说到关系型数据库，大多数人都会想到 MySQL。MySQL 是目前最流行的数据库管理系统之一，支持使用最常见的结构化查询语言（SQL）进行数据库操作，并以表格、行、列的形式存储数据。这种存储数据的方法源自埃德加·科德（Edgar Frank Codd）于 1970 年提出的关系型数据模型。

在关系型数据库中，可以为待存储的每种类型的数据创建一个表。例如，球员表用来存储所有的球员信息，球队表用来存储球队信息等。SQL 表中的每行数据都必须包含一个主键（primary key）。主键是该行数据的唯一标识符。一般地，主键作为字段 ID 都是随行数自增的。关系型数据库自问世以来一直为计算机行业提供着非常好的服务，并将未来很长的时间内继续服务下去。

如果你用过 Excel、WPS 或其他类似的应用，你就会大概了解到关系数据库是如何工作的。首先设置好列，然后在对应的列下添加行数据。你可以对某一列数据进行求平均值或其他聚合操作，这与在关系型数据库 MySQL 中求平均值的操作类似。而 EXcel 中的数据透视表则相当于在关系型数据库 MySQL 中使用聚合函数和 CASE 语句对数据进行查询。一个 EXcel 文件可以有多张表，一张表就相当于 MySQL 的一张表。一个 EXcel 文件则类似于一个 MySQL 数据库。

关系型数据库中的关系

与图数据库不同，关系型数据库（或 SQL 型的数据库）中的边也是作为实体存储在专门的边表中的。先创建两个表，球员（player）和球队（team），然后再创建表 player_team 作为边表。边表通常由相关的表 join 而成。例如，此处的边表 player_team 就由球员表和球队表 join 而成。



这种存储边的方式在关联小型数据集时问题并不大，但是当关系型数据库中的关系太多时，问题就出现了。事实上，关系型数据库是非常“反关系的”。具体来说，当你只想查询一个球员的队友时，你必须对表中的所有数据进行 join 操作，然后再过滤掉你不需要的所有数据，当你的数据集达到一定规模时，这将给关系型数据库带来巨大压力。如果你想关联多张不同的表，可能在 join 爆炸（join bombs）前系统就已经无法响应了。

关系型数据库起源

上文提到，关系型数据模型最早是由 IBM 的工程师埃德加·科德（Edgar Frank Codd）于 1970 年提出的。科德写了几篇数据库管理系统方面的论文，论述了关系型数据模型的潜力。关系型数据模型不依赖于数据链接列表（网状数据或层级数据），而是更多依赖于数据集。他使用元组演算（tuple calculus）的数学方法论证了这些数据集能够完成与导航数据库管理系统相同的任务。唯一的要求是，关系型数据模型需要一种合适的查询语

2022 vesoft Inc.

言，以保证数据库的一致性要求。这就为后来声明型的结构化查询语言（SQL）提供了灵感来源。IBM 的 R 系统是关系型数据模型的最早使用者之一。然而，由前 IBM 员工拉里·埃里森创办的名叫软件开发实验室的小公司在市场上击败了 IBM。该公司的产品就是后来为我们熟知的 Oracle。

由于“关系数据库”在当时是一个比较时髦的词汇，因此许多数据库供应商都喜欢在其产品名称中使用这个词汇，尽管他们的产品实际上并不是关系型的。为了防止这种情况并减少关系型数据模型的错误使用，科德提出了著名科德 12 定律（Codd's 12 rules）。所有关系型数据系统都必须遵循科德 12 定律。

NoSQL 数据库

图数据库并不是可以克服关系型数据库缺点的唯一替代方案。现在市面上还有很多非关系型数据库的产品，这些产品都可以叫做 NoSQL。NoSQL 一词最早于上世纪 90 年代末提出，可以解释为“非 SQL” 或“不仅是 SQL”，具体解释要根据语境判断。为便于理解，这里 NoSQL 可以解释成 “非关系型数据库”。不同于关系型数据库，NoSQL 数据库提供的数据存储、检索机制并不是基于表关系建模的。 NoSQL 数据库可以分为四类：

- 键值存储（key-value stores）
- 列式存储（column-family stores）
- 文件存储（document stores）
- 图数据库（graph databases）

下面将分别介绍这四类数据库。

键值存储

键值存储，顾名思义，就是使用键值对存储数据的数据库。不同于关系型数据库，键值存储是没有表和列的。如果一定要做类比，键值数据库本身就像一张有很多列（也就是键）的大表。在键值存储数据库中，数据（即键值对中的值）都是通过键来存储和查询的，通常用哈希列表来实现。这比传统的 SQL 数据库要简单得多，而且对于某些 web 应用来说，这就足够了。

键值模型对于 IT 系统来说优势在于简单、易部署。多数情况下，这种存储方式对非关联的数据很适用。如是只是存储数据而无需查询的话，使用这种存储方法就没有问题。但是如果 DBA 只对部分值进行查询或更新的时候，键值模型就显得效率低下了。常见的键值存储数据库有：Redis、Voldemort、Oracle BDB。

列式存储

NoSQL 数据库的列式存储与 NoSQL 数据库的键值存储有许多相似之处，因为列式存储仍然在使用键进行存储和检索。区别在于列式存储数据库中，列是最小的存储单元，每一列均由键、值以及用于版本控制和冲突解决的时间戳组成。这在分布式扩展时特别有用，因为在数据库更新时，可以使用时间戳定位过期数据。由于列式存储良好的扩展性，因此适用于非常大的数据集。常见的列式存储数据库有：HBase、Cassandra、HadoopDB 等。

文档存储

准确来说，NoSQL 数据库文档存储实际上也是基于键值的数据库，只不过对功能做了增强。数据仍然以键值的形式存储，但是文档存储中的值是结构化的文档，而不仅仅是一个字符串或单个值。也就是说，由于信息结构的增加，文档存储能够执行更优化的查询，并且使数据检索更加容易。因此，文档存储特别适合存储、索引并管理面向文档的数据或者类似的半结构化数据。

从技术上讲，作为一个半结构化的信息单元，文档存储中的文档可以是任何形式可用的文档，包括 XML、JOSN、YAML 等，这取决于数据库供应商的设计。 比如，JSON 就是一种常见的选择。虽然 JSON 不是结构化数据的最佳选择，但是 JSON 型的数据在前端和后端应用中都可以使用。常见的文档存储数据库有：MongoDB、CouchDB、Terrastore 等。

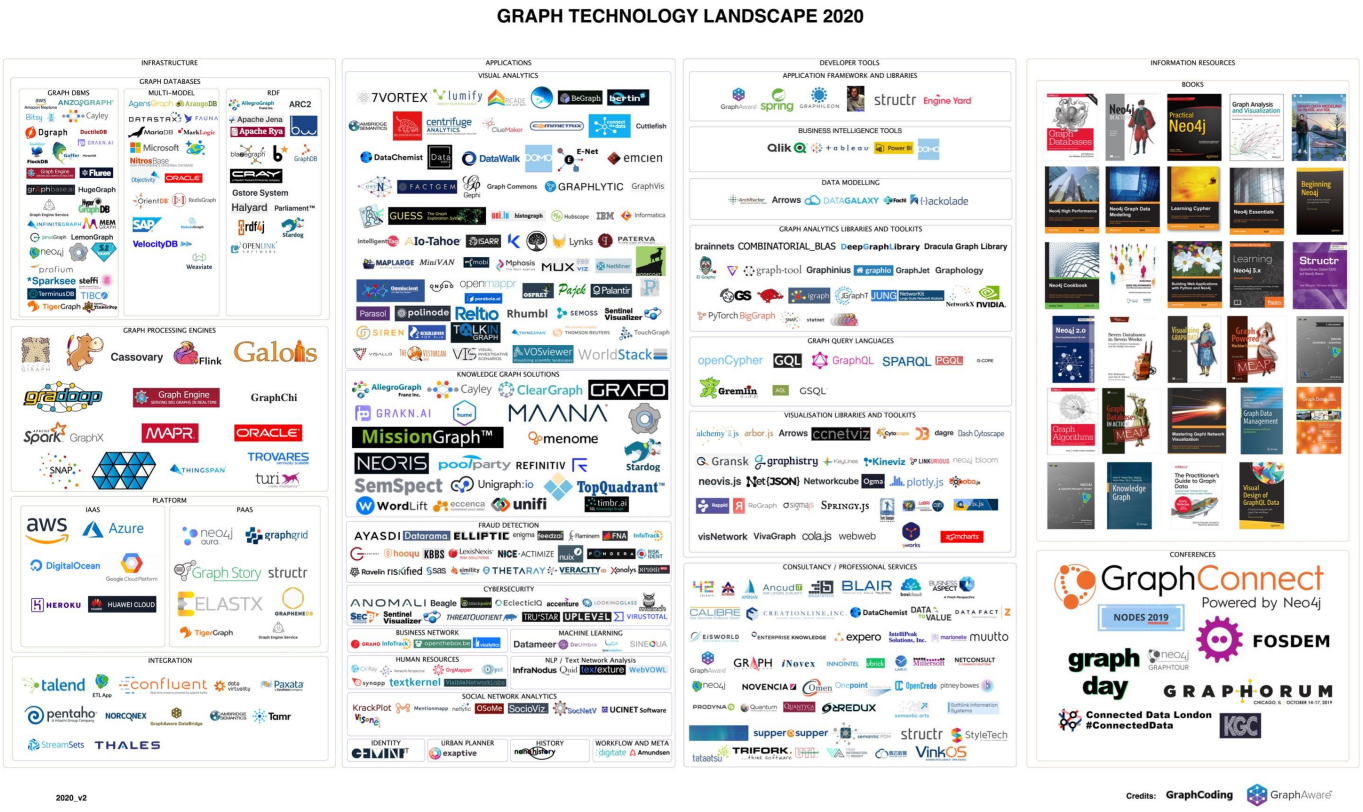
图存储

最后一类 NoSQL 数据库是图数据库。本书重点讨论的 Nebula Graph 也是一种图数据库。虽然同为 NoSQL 型数据库，但是图数据库与上述 NoSQL 数据库有本质上的差异。图数据库以点、边、属性的形式存储数据。其优点在于灵活性高，支持复杂的图形算法，可用于构建复杂的关系图谱。我们将在随后的章节中详细讨论图数据库。不过在本章中，你只要知道图数据库是一种 NoSQL 类型的数据库就可以了。常见的图数据库有：Nebula Graph、Neo4j、OrientDB 等。

2.3.2 图技术方面

来看一张 2020 年的图技术全景[1]

2022 vesoft Inc.



和图有关联的技术有很多，可以大致分为这么几类：

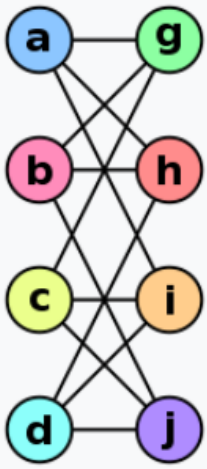
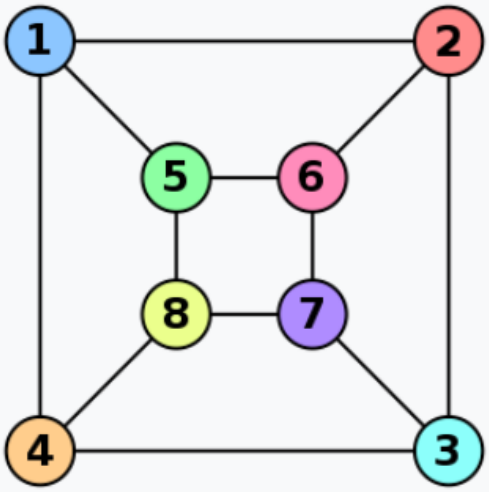
- 基础设施：包括图数据库、图计算(处理)引擎、图深度学习、云服务等。
- 应用：包括可视化、知识图谱、反诈骗、网络安全、社交网络等。
- 开发工具：包括图查询语言、建模工具、开发框架和库。
- 电子书籍[2]和会议等。

图语言

在上一节中，我们介绍了图语言的历史。在这一节中，我们对图语言的功能做一个分类：

- 近邻查询：查询给定点或者边的邻边、邻点，或者是 K 跳的近邻。
- 图模式匹配(Pattern matching)：找到一个/所有的子图，满足给定的图模式；这个问题非常接近于"子图同构映射(subgraph isomorphism)"——虽然两个看上去不同的图，但其实是一模一样的[3]。

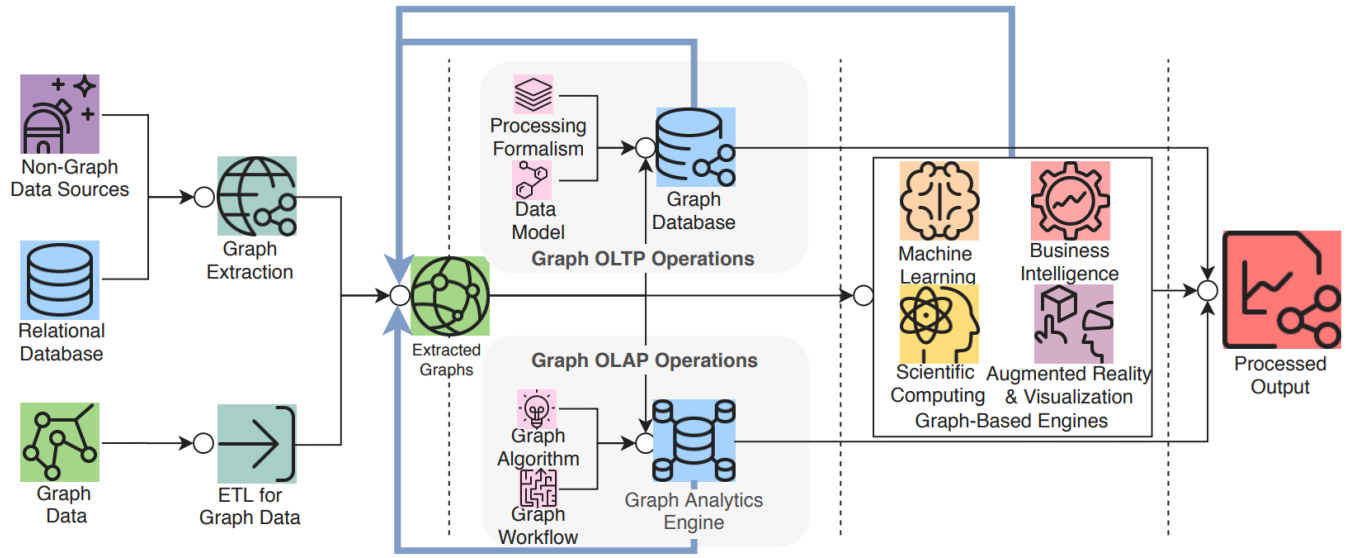
2022 vesoft Inc.

Graph G	Graph H	An isomorphism between G and H
		$f(a) = 1$ $f(b) = 6$ $f(c) = 8$ $f(d) = 3$ $f(g) = 5$ $f(h) = 2$ $f(i) = 4$ $f(j) = 7$

- 可达性（连通性）问题：最常见的可达性问题就是最短路径问题。泛化一些，这类问题通常用正则路径(Regular Path Query)的方式来描述——一系列联通的点组构成了一条路径，而该路径需要满足某种正则表达。
- 分析型问题：通常与一些汇聚型算子相关（平均值、count、最大值、点的出入度），或者度量所有两两点之间的距离、某节点与其他节点之间的互动程度（介数中心性）等。

图数据库**(Graph Database)**与图处理**(Graph processing)**系统

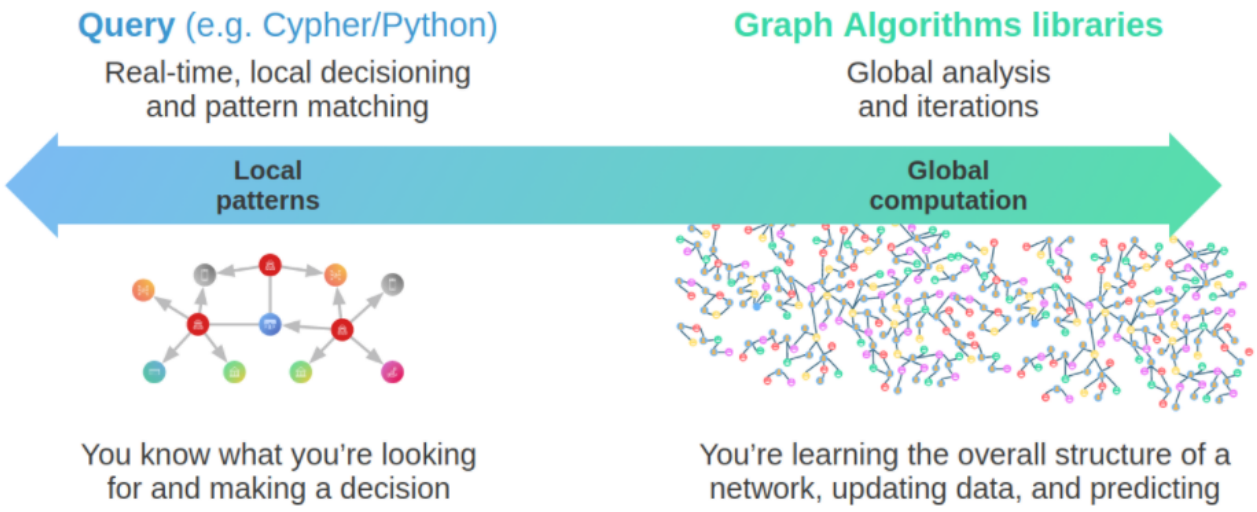
一个图系统通常会涉及到复杂的数据流水线[4]，从数据源(左边)到处理输出(右边)，会经过多个数据加工处理环节和系统；大的模块可以分为 ETL模块，图数据库系统(Graph OLTP)，图处理系统(Graph OLAP)，基于图引擎的应用系统（BI、知识图谱等）。



2022 vesoft Inc.

虽然这两类系统都是与图数据和图技术相关的系统，也处理类似的目标，但是他们有着不同的起源和特长（及弱点）：

- （在线）图数据库目标是图的持久化存储管理、高效的子图操作。硬盘（及网络）是目标运行设备，物理/逻辑数据映射，数据完整性和（故障）一致性是主要目标。每一个请求通常只会涉及到全图的一小部分，通常可以在一台服务器上完成；单个请求时延通常在毫秒到秒级别，请求并发量通常在几千到几十万。早期的 Neo4j 是图数据库领域的起源之一。
- （离线）图处理系统目标是全图的大批量、并行、迭代、处理与分析，内存（及网络）是目标运行设备。每一个请求会涉及到所有的图节点，需要所有的服务器参与完成；单个请求的时延通常在分钟到小时（天），请求并发量通常为个位数。Google 的 Pregel[5] 是图处理系统的典型起源代表，它的点中心编程抽象与BSP的运行模式构成的编程范式，相比之前 Hadoop Map-Reduce 是更为图友好的 API 抽象。



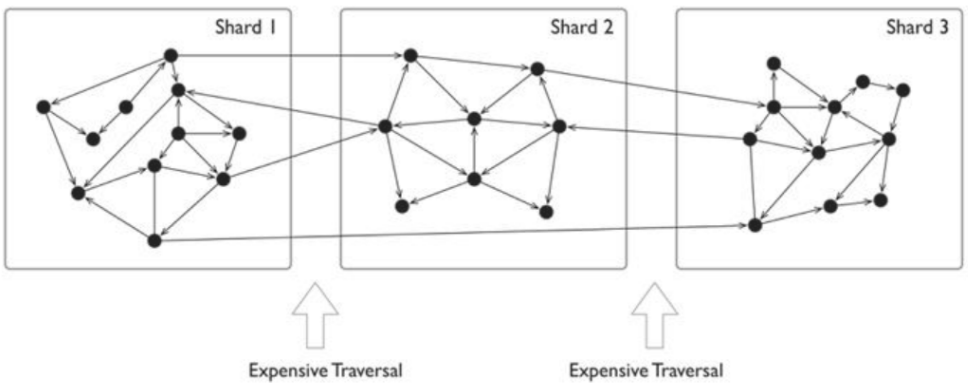
6

图的分片方式

对于一个大规模的图数据来说，是很难存放在单个服务器的内存中的，即使仅仅存放图结构本身也不够。而且通过增加单服务器的能力，其成本价格通常成指数级别上升。 此外，随着数据量的增加，例如到达千亿级别的时候，已经超过了市面上所有商用服务器的容量能力。

与此对应的，另外一个经常使用的方案，是对数据进行分片，并将每个分片放置在不同的服务器上（并进行冗余备份），以此来增加可靠性和性能。对于一些 NoSQL 型的系统，例如 key-value 或者文档型的系统来说，这个分片方式是比较直观和自然的；通常可以根据 key 或者 docID，来将每个记录或者数据单元(key-value, doc)放在不同的服务器上。

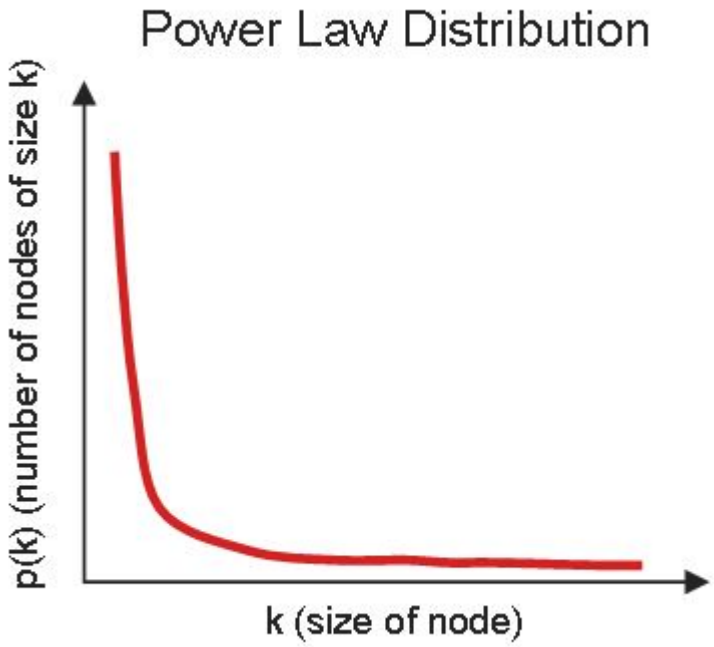
但是图这种数据结构的分片通常不那么直观，这是因为通常图是“全联通”的，每个点通常只要6跳就可以联通到其他任何节点； 而理论上早已证明图的划分问题是 NP 的。 与此同时，当把整个图数据分散到多个服务器时，跨服务器的网络访问时延10倍于同一个服务器内部的硬件(内存)访问时间；因此对于一些深度优先遍历的场景，会发生大量的跨网络访问，导致整体时延极高。

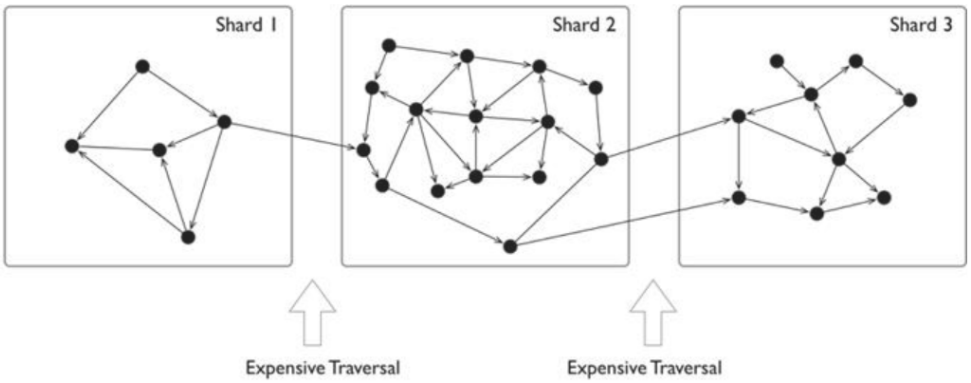


7

2022 vesoft Inc.

另一方面，通常图有着明显的幂律分布；少量节点的邻边稠密程度远大于平均的节点，虽然处理这些节点通常可以在同一台服务器内，减少了跨网络访问，但这也意味着这些服务器压力会远大于平均。

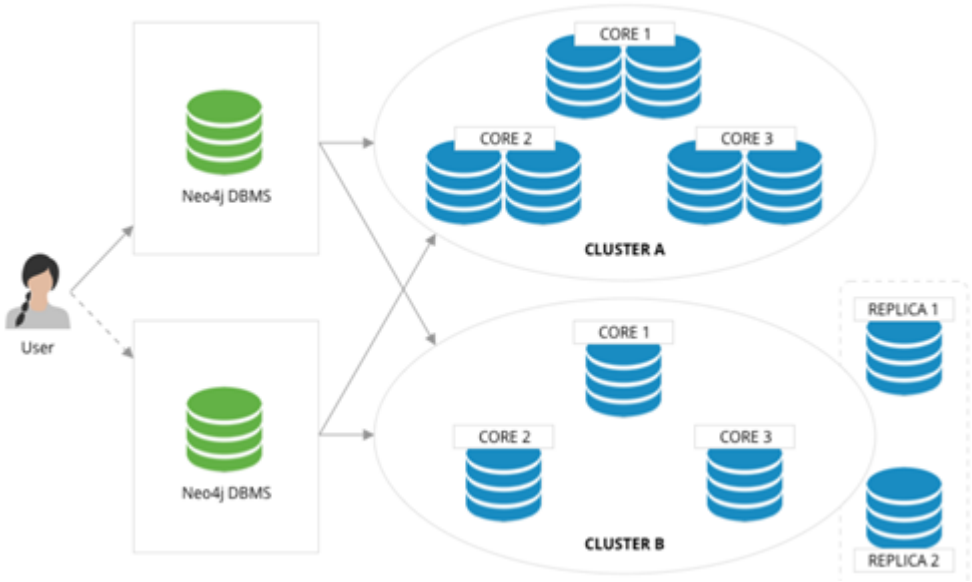




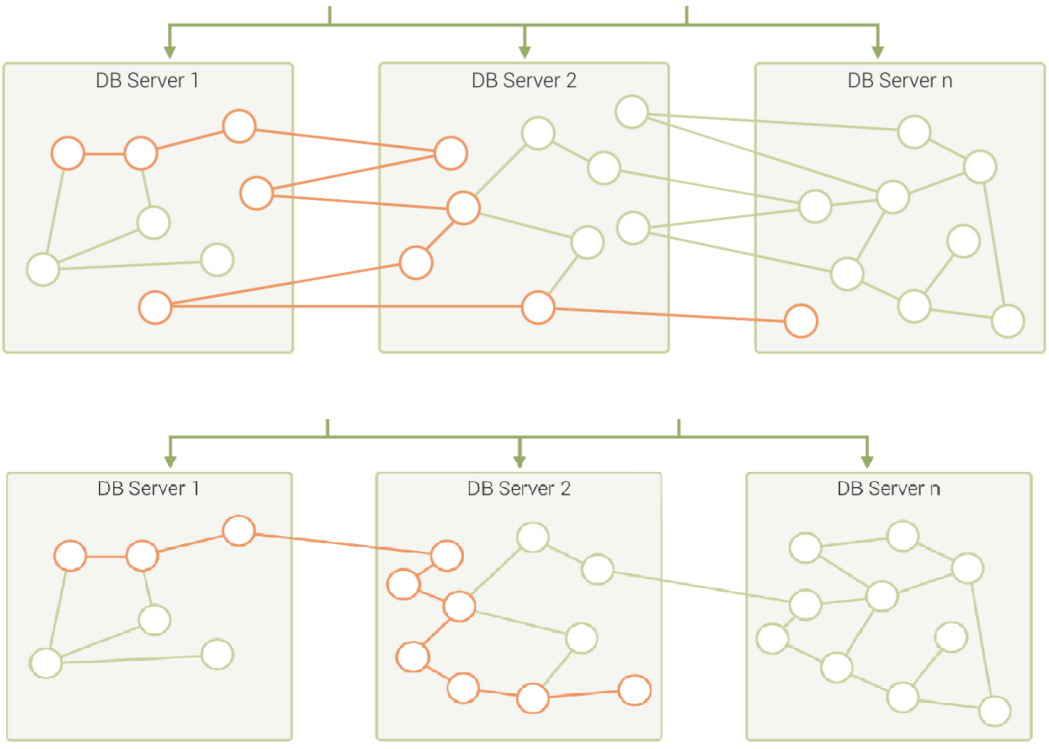
因此，常见的图分片(Sharding)方式有几类：

- 偏应用层面的分片：应用层感知并控制每个点和边应该落在哪个分片上，一般来说可以根据点和边的类型（比如业务意义来人为）判断。将一组相同类型的点放在一个分片，另一组相同类型的点放在另一个分片。当然，为了高可靠，分片本身还可以做多副本。在应用使用时，从各个分片取回所要的点和边，然后在偏应用侧（或者某个代理服务器端），将取回的数据拼装成最终的结果。其典型代表是 Neo4j 4.x 的 Fabric。

2022 vesoft Inc.



- 使用分布式的缓存层：在硬盘之上增加内存缓存层，并对重要的部分分片和数据（例如图结构）进行缓存，并预热这部分缓存。
- 增加（只读）副本或视图 (View)：为部分图分片增加只读的副本或者建立一个视图，将较重的读请求负载通过这些分片服务器承担。
- 进行细颗粒度的图划分：例如将点和边组成多个小分片（Partition），而不是一台服务器一个大分片（Sharding），再将关联性较强的 Partition 尽量放置在同一个服务器上[8]。



具体工程实践时，也会混合使用上述几种方式。通常，离线的图处理系统会通过一个ETL过程，将图进行一定程度的预处理以提高局部性；而在线图数据库系统通常会选择周期性的数据再平衡过程来提高数据局部性。

2022 vesoft Inc.

一些技术上的挑战

在文献[9]中，对于无处不在的大图和挑战做了详尽的调研，下面是其列出的十大图技术挑战：

- 可扩展性(软件可以处理更大的图规模)：包括大图的加载、更新、图计算和图遍历，触发器，超级节点；
- 可视化：可定制布局，大图的渲染，多层次展示，动态（更新）的展示
- 查询语言和编程 API：包括语言表达能力、标准兼容性、与现有系统的兼容性；子查询的设计和跨多图之间的关联查询
- 更快的图（及机器学习）算法
- 易用性（配置和使用）
- 性能指标与测试
- 更通用的图技术软件（例如，处理离线、在线、流式的计算）
- 图清洗（ETL）
- Debug 调试与测试

一些开源的单机图工具

对于图数据库通常会有一个误解，只要涉及到图结构的数据存取就需要存放在图数据库中，这是一种很大的浪费。

这就像也许你只需要一个 SQLite，却用了一个 Oracle。

当数据量并不大时，通常单机内存可以放下，例如数据量几千万的点边关系，使用一些单机的开源工具也可以取得很好的效果。

> **Note**
>
> 下面是一些推荐的单机图库，也可以集成在你的应用程序里面。

- JGraphT[10]：一个知名的开源 Java 图论库(library)，其实现了相当多的高效图算法。
- JUNG[11]是BSD许可下用Java编写的开源图建模和可视化框架。该框架内置了许多布局算法，以及诸如图聚类和节点中心性度量之类的分析算法。
- igraph[12]：一个轻量且功能强大的 Library，支持R、python、C
- NetworkX[13]：数据科学家做图论分析第一选择， python。
- Cytoscape[14]：功能强大的可视化开源图分析工具。
- Gephi[15]：功能强大的可视化开源图分析工具。
- arrows.app[16]： 非常简单的脑图工具，用于可视化生成 Cypher 语句.

一些行业数据集和 **Benchmark**

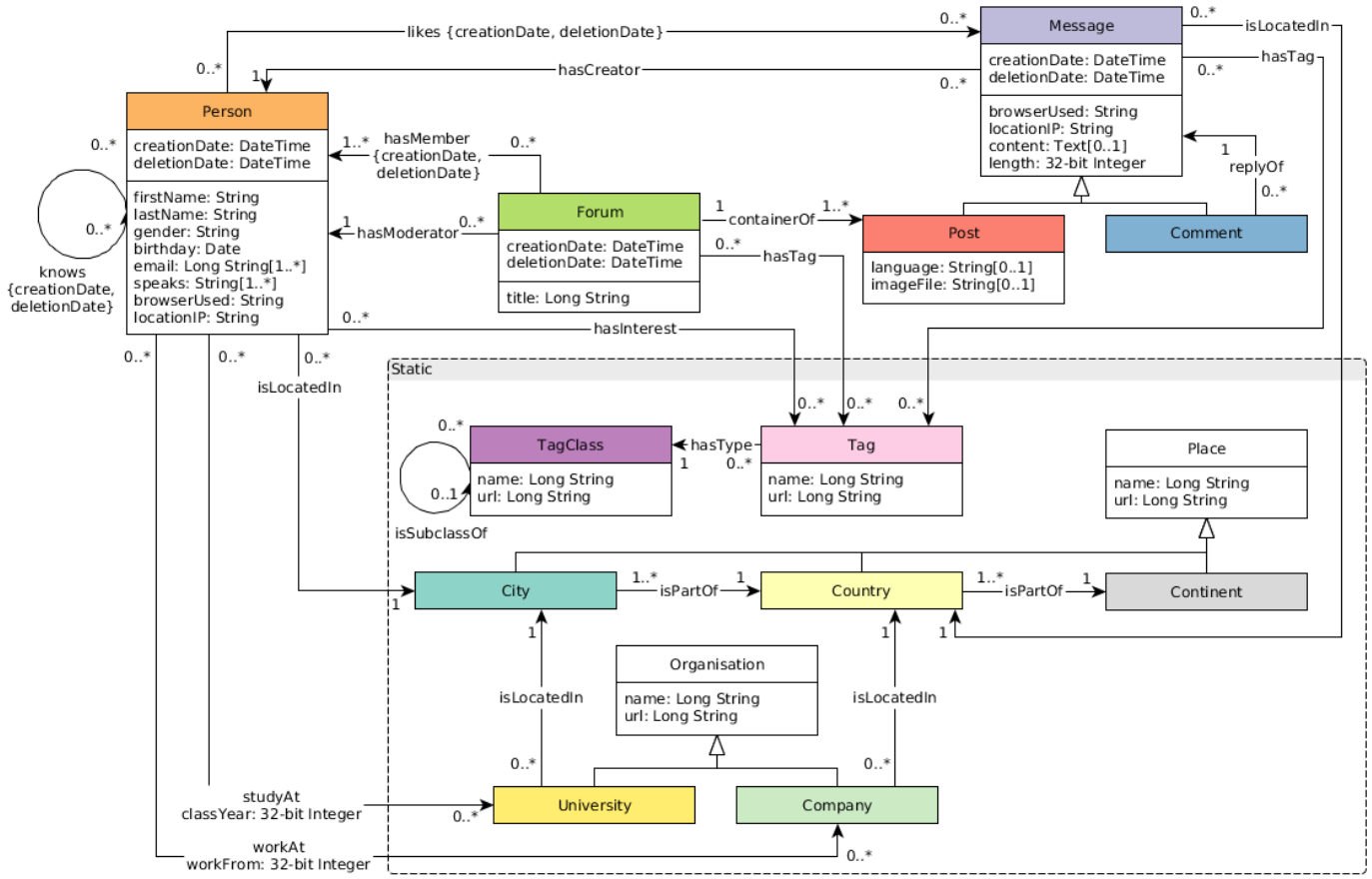
LDBC

关联数据基准委员会（LDBC[17]，Linked Data Benchmark Council）是由Oracle、Intel等软硬件巨头和主流图数据库厂商Neo4j和TigerGraph等组成的非赢利机构，是图的基准指南制定者与测试结果发布机构，在行业内有着很高的影响力。

社交网络基准测试（SNB，Social Network Benchmark）是由关联数据基准委员会（LDBC）开发的面向图数据库的基准测试（Benchmark）之一，分为交互式查询（Interactive）和商业智能（BI）两个场景。其作用类似于 TPC-C, TPC-H 等测试在 SQL 型数据库中的功能，可以帮助用户比较多种图数据库产品的功能、性能、容量。

SNB 数据集模拟一个社交网络的人、发帖之间的关系，考虑了社交网络的分布属性、人的活跃度等等社交信息。

2022 vesoft Inc.

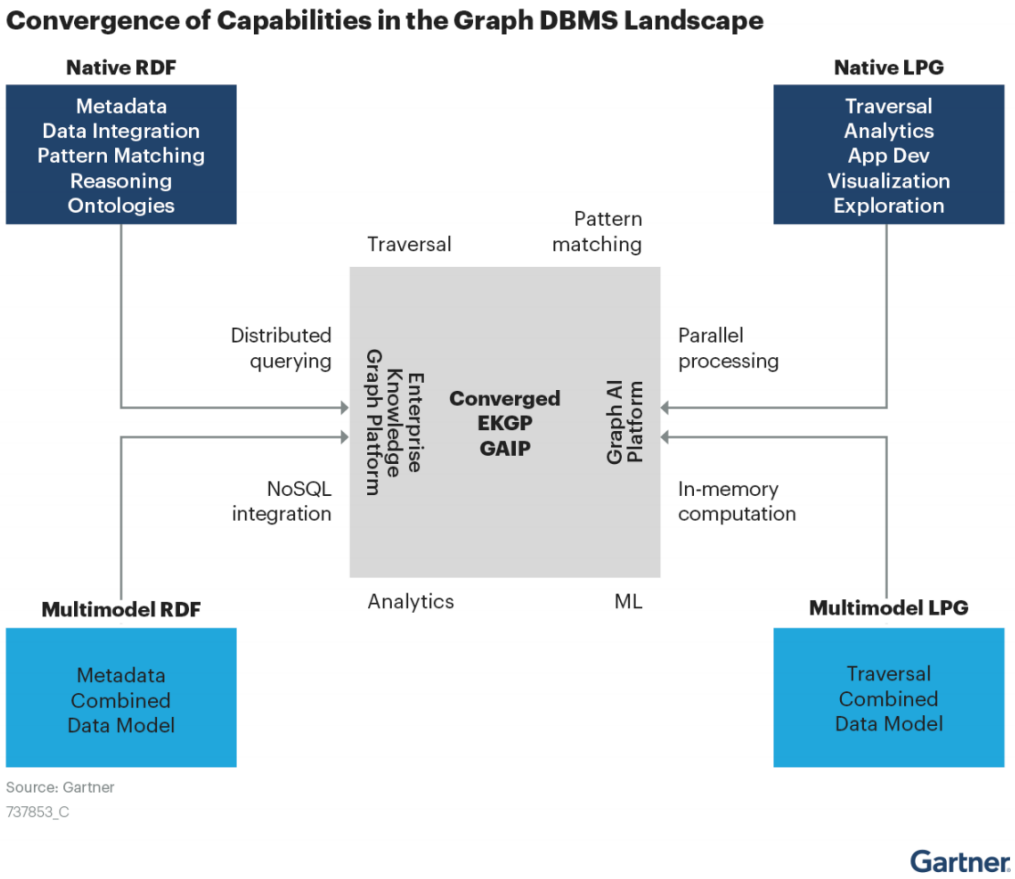


其标准数据量从 0.1 GB (scale factor 0.1) 到 1000 GB (sf1000)，也可以生成 10 TB，100 TB等更大的数据集；其点、边数量如下表。

2022 vesoft Inc.

2.3.3 一些趋势

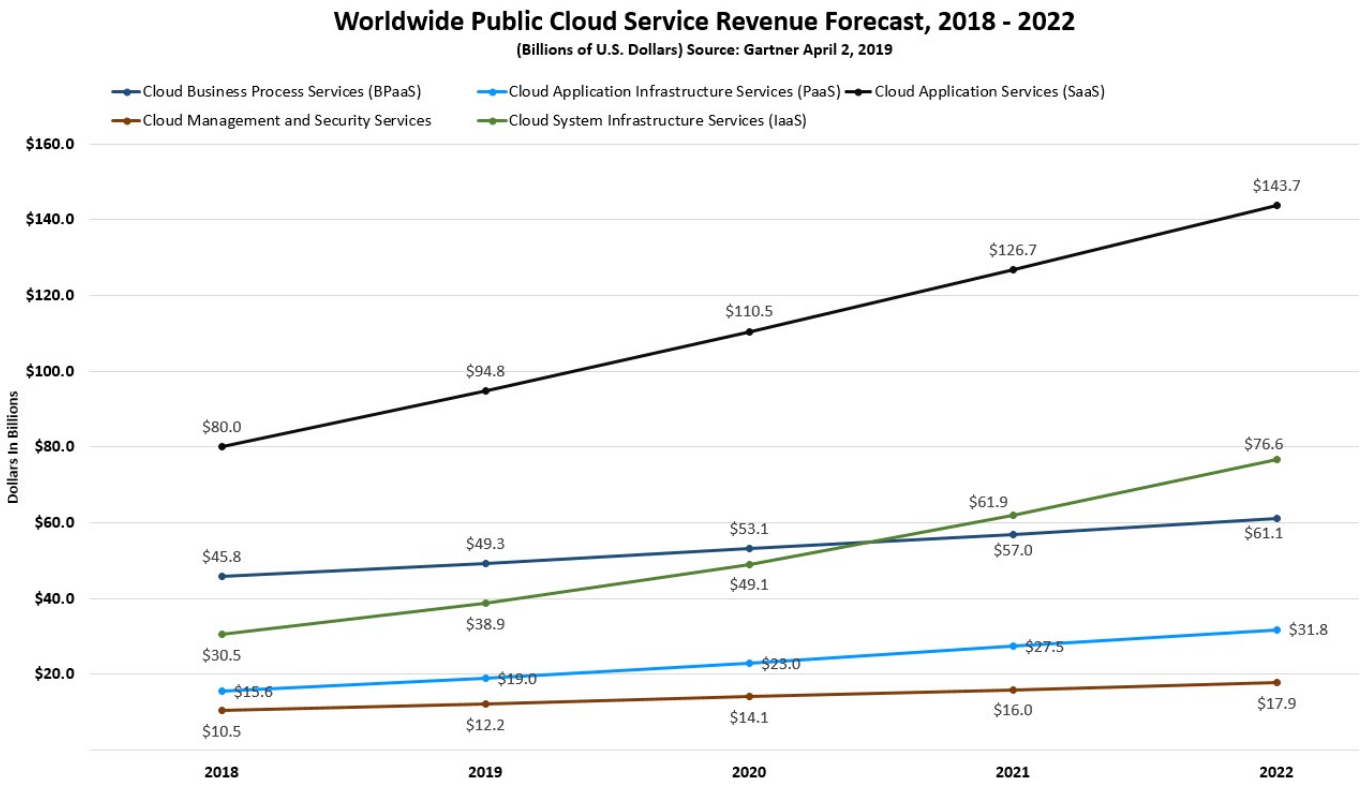
虽然图的各种技术起源和目标并不相同，但在相互借鉴和融合



上云的趋势在加速，对于弹性能力提出更高要求

根据 Gartner 的预计，云服务一直保持较快的增速和渗透率[18]。大量的商业软件，正在从 10 年前完全私有本地逐步转向基于云服务的商业模式。云服务的一大优点是其提供了近乎无限的弹性能力；这也要求各种基于云基础设施的软件必须有更好的快速弹性扩缩容能力。

2022 vesoft Inc.

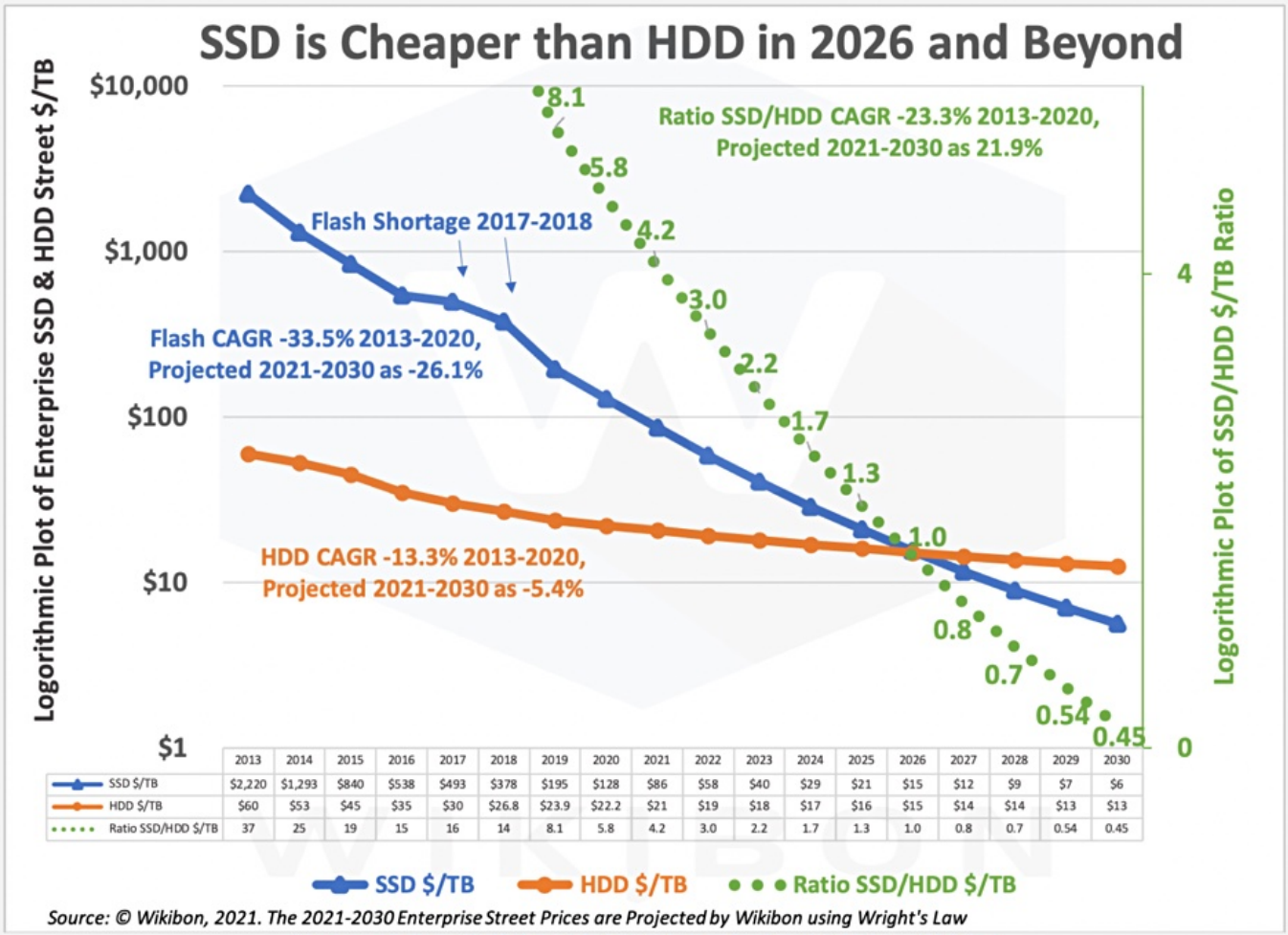


硬件趋势，**SSD** 将成为主流的持久化设备

硬件决定了软件的架构——从发现摩尔定律的 50 年代到进入多核的 00 年代，硬件发展趋势和速度一直深刻的决定了软件的架构。数据库类系统大多围绕“硬盘+内存”设计，高性能计算型系统大多围绕“内存 + CPU”设计，分布式系统面对千兆、万兆和 RDMA 网卡的设计也完全不同。

图基于拓扑的遍历有着极其明显的随机访问特点，因此大多数早期图数据库系统都采用了“大内存 + HDD” 的架构————通过设计常驻在内存中的一些数据结构（例如B+树、Hash表等），在内存中实现随机访问目的，以优化图的拓扑遍历，再将这些随机访问转换成 HDD 所适合的顺序读写。整套软件的架构（包括存储和计算层）都必须基于和围绕这样的 IO 流程来展开。随着 SSD 价格的快速下降[19]，SSD 正在替代 HDD 成为持久化设备的主流。SSD 随机访问友好、IO 队列深、按块存取的特点与 HDD 高度顺序、随机时延极高、磁道易损坏的访问特点有着明显的不同。全部的软件架构也需要重新设计，这成为沉重的历史技术负担。

2022 vesoft Inc.



	2013	2014	2015	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030
SSD $/TB	$2,220	$1,293	$840	$538	$493	$378	$195	$128	$86	$58	$40	$29	$21	$15	$12	$9	$7	$6
HDD $/TB	$60	$53	$45	$35	$30	$26.8	$23.9	$22.2	$21	$19	$18	$17	$16	$15	$14	$14	$13	$13
Ratio SSD/HDD $/TB	37	25	19	15	16	14	8.1	5.8	4.2	3.0	2.2	1.7	1.3	1.0	0.8	0.7	0.54	0.45

Figure 4 - SSD/HDD Pricing Ratio 2013 - 2030

Source: © Wikibon, 2021.

2022 vesoft Inc.

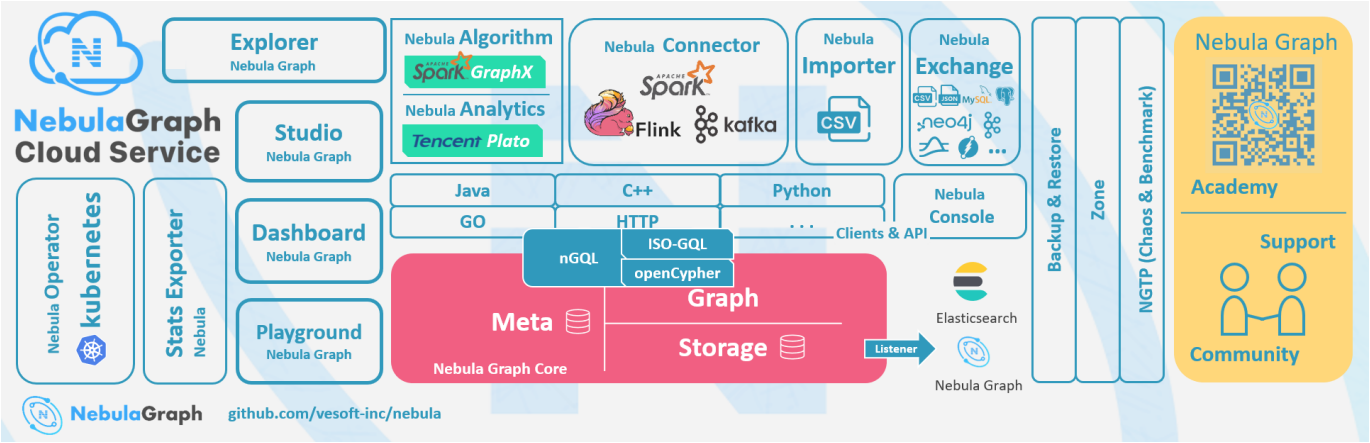
1. https://graphaware.com/graphaware/2020/02/17/graph-technology-landscape-2020.html ↩
2. 学习目的(非商业用途)可以联系[作者]((mailto:min.wu@vesoft.com)获取电子版。 ↩
3. https://en.wikipedia.org/wiki/Graph_isomorphism ↩
4. The Future is Big Graphs! A Community View on Graph Processing Systems. https://arxiv.org/abs/2012.06171 ↩
5. G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In Proceedings of the International Conference on Management of data (SIGMOD), pages 135–146, New York, NY, USA, 2010. ACM ↩
6. https://neo4j.com/graphacademy/training-iga-40/02-iga-40-overview-of-graph-algorithms/ ↩
7. https://livebook.manning.com/book/graph-powered-machine-learning/welcome/v-8/ ↩
8. https://www.arangodb.com/learn/graphs/using-smartgraphs-arangodb/ ↩
9. https://arxiv.org/abs/1709.03188 ↩
10. https://jgrapht.org/ ↩
11. https://github.com/jrtom/jung ↩
12. https://igraph.org/ ↩
13. https://networkx.org/ ↩
14. https://cytoscape.org/ ↩
15. https://gephi.org/ ↩
16. https://arrows.app/ ↩
17. https://github.com/ldbc/ldbc_snb_docs ↩
18. https://cloudcomputing-news.net/news/2019/apr/15/public-cloud-soaring-to-331b-by-2022-according-to-gartner/ ↩
19. https://blocksandfiles.com/2021/01/25/wikibon-ssds-vs-hard-drives-wrights-law/ ↩

最后更新: June 20, 2023

2022 vesoft Inc.

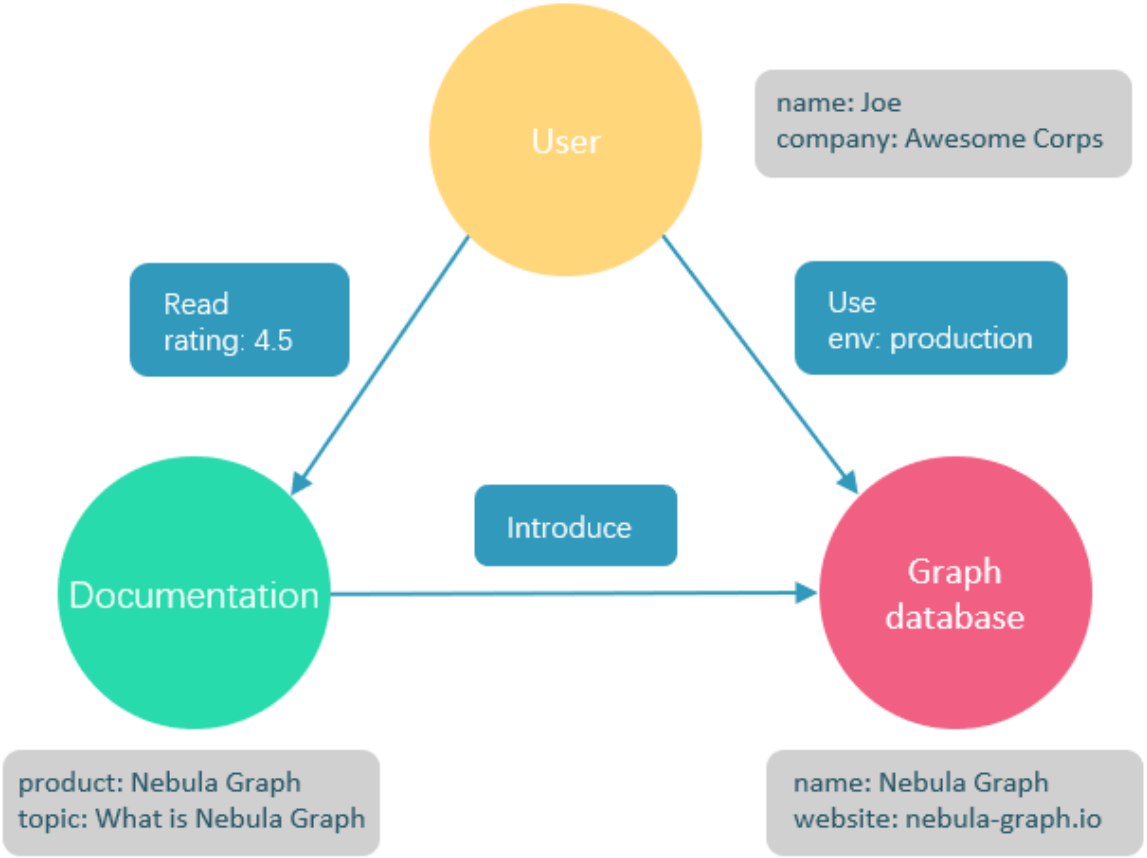
2.4 什么是 Nebula Graph

Nebula Graph 是一款开源的、分布式的、易扩展的原生图数据库，能够承载包含数千亿个点和数万亿条边的超大规模数据集，并且提供毫秒级查询。



2.4.1 什么是图数据库

图数据库是专门存储庞大的图形网络并从中检索信息的数据库。它可以将图中的数据高效存储为点（Vertex）和边（Edge），还可以将属性（Property）附加到点和边上。



2022 vesoft Inc.

图数据库适合存储大多数从现实抽象出的数据类型。世界上几乎所有领域的事物都有内在联系，像关系型数据库这样的建模系统会提取实体之间的关系，并将关系单独存储到表和列中，而实体的类型和属性存储在其他列甚至其他表中，这使得数据管理费时费力。

Nebula Graph 作为一个典型的图数据库，可以将丰富的关系通过边及其类型和属性自然地呈现。

2.4.2 Nebula Graph 的优势

开源

Nebula Graph 是在 Apache 2.0 条款下开发的。越来越多的人，如数据库开发人员、数据科学家、安全专家、算法工程师，都参与到 Nebula Graph 的设计和开发中来，欢迎访问 Nebula Graph GitHub 主页参与开源项目。

高性能

基于图数据库的特性使用 C++ 编写的 Nebula Graph，可以提供毫秒级查询。众多数据库中，Nebula Graph 在图数据服务领域展现了卓越的性能，数据规模越大，Nebula Graph 优势就越大。详情请参见 Nebula Graph benchmarking 页面。

易扩展

Nebula Graph 采用 shared-nothing 架构，支持在不停止数据库服务的情况下扩缩容。

易开发

Nebula Graph 提供 Java、Python、C++ 和 Go 等流行编程语言的客户端，更多客户端仍在开发中。详情请参见 Nebula Graph clients。

高可靠访问控制

Nebula Graph 支持严格的角色访问控制和 LDAP（Lightweight Directory Access Protocol）等外部认证服务，能够有效提高数据安全性。详情请参见验证和授权。

生态多样化

Nebula Graph 开放了越来越多的原生工具，例如 Nebula Studio、Nebula Console、Nebula Exchange 等，更多工具可以查看生态工具概览。

此外，Nebula Graph 还具备与 Spark、Flink、HBase 等产品整合的能力，在这个充满挑战与机遇的时代，大大增强了自身的竞争力。

兼容 **openCypher** 查询语言

Nebula Graph 查询语言，简称为 nGQL，是一种声明性的、部分兼容 openCypher 的文本查询语言，易于理解和使用。详细语法请参见 nGQL 指南。

面向未来硬件，读写平衡

闪存型设备有着极高的性能，并且价格快速下降， Nebula Graph 是一个面向 SSD 设计的产品，相比于基于 HDD + 大内存的产品，更适合面向未来的硬件趋势，也更容易做到读写平衡。

灵活数据建模

用户可以轻松地在 Nebula Graph 中建立数据模型，不必将数据强制转换为关系表。而且可以自由增加、更新和删除属性。详情请参见数据模型。

广受欢迎

腾讯、美团、京东、快手、360 等科技巨头都在使用 Nebula Graph。详情请参见 Nebula Graph 官网。

2022 vesoft Inc.

2.4.3 适用场景

Nebula Graph 可用于各种基于图的业务场景。为节约转换各类数据到关系型数据库的时间，以及避免复杂查询，建议使用 Nebula Graph。

欺诈检测

金融机构必须仔细研究大量的交易信息，才能检测出潜在的金融欺诈行为，并了解某个欺诈行为和设备的内在关联。这种场景可以通过图来建模，然后借助 Nebula Graph，可以很容易地检测出诈骗团伙或其他复杂诈骗行为。

实时推荐

Nebula Graph 能够及时处理访问者产生的实时信息，并且精准推送文章、视频、产品和服务。

知识图谱

自然语言可以转化为知识图谱，存储在 Nebula Graph 中。用自然语言组织的问题可以通过智能问答系统中的语义解析器进行解析并重新组织，然后从知识图谱中检索出问题的可能答案，提供给提问人。

社交网络

人际关系信息是典型的图数据，Nebula Graph 可以轻松处理数十亿人和数万亿人际关系的社交网络信息，并在海量并发的情况下，提供快速的好友推荐和工作岗位查询。

2.4.4 视频

用户也可以通过视频了解什么是图数据。

- 图数据库 Nebula Graph 介绍视频（01 分 39 秒）

2.4.5 主题演讲

查看演讲快速了解图数据库概况。

2.4.6 相关链接

- 官方网站
- 文档首页
- 博客首页
- 论坛
- GitHub

最后更新: June 20, 2023

2022 vesoft Inc.

2.5 数据模型

本文介绍 Nebula Graph 的数据模型。数据模型是一种组织数据并说明它们如何相互关联的模型。

2.5.1 数据模型

Nebula Graph 数据模型使用 6 种基本的数据模型：

- 图空间（Space）

  图空间用于隔离不同团队或者项目的数据。不同图空间的数据是相互隔离的，可以指定不同的存储副本数、权限、分片等。

- 点（Vertex）

  点用来保存实体对象，特点如下：

- 点是用点标识符（`VID`）标识的。`VID` 在同一图空间中唯一。VID 是一个 int64，或者 fixed_string(N)。
- 点可以有 0 到多个 Tag。

  Compatibility

  Nebula Graph 2.x 及以下版本中的点必须包含至少一个 Tag。

- 边（Edge）

  边是用来连接点的，表示两个点之间的关系或行为，特点如下：

- 两点之间可以有多条边。
- 边是有方向的，不存在无向边。
- 四元组 `<起点 VID、Edge type、边排序值 (rank)、终点 VID>` 用于唯一标识一条边。边没有 EID。
- 一条边有且仅有一个 Edge type。
- 一条边有且仅有一个 Rank，类型为 int64，默认值为 0。

  关于

  Rank 可以用来区分 Edge type、起始点、目的点都相同的边。该值完全由用户自己指定。
  读取时必须自行取得全部的 Rank 值后排序过滤和拼接。
  不支持诸如 `next()`, `pre()`, `head()`, `tail()`, `max()`, `min()`, `lessThan()`, `moreThan()` 等函数功能，也不能通过创建索引加速访问或者条件过滤。

- 标签（Tag）

  Tag 由一组事先预定义的属性构成。

- 边类型（Edge type）

  Edge type 由一组事先预定义的属性构成。

- 属性（Property）

  属性是指以键值对（Key-value pair）形式表示的信息。

2022 vesoft Inc.

Note

Tag 和 Edge type 的作用，类似于关系型数据库中“点表”和“边表”的表结构。

2.5.2 有向属性图

Nebula Graph 使用有向属性图模型，指点和边构成的图，这些边是有方向的，点和边都可以有属性。

下表为篮球运动员数据集的结构示例，包括两种类型的点（**player**、**team**）和两种类型的边（**serve**、**follow**）。

类型	名称	属性名（数据类型）	说明
Tag	**player**	name (string) age（int）	表示球员。
Tag	**team**	name (string)	表示球队。
Edge type	**serve**	start_year (int) end_year (int)	表示球员的行为。 该行为将球员和球队联系起来，方向是从球员到球队。
Edge type	**follow**	degree (int)	表示球员的行为。 该行为将两个球员联系起来，方向是从一个球员到另一个球员。

Note

Nebula Graph 中没有无向边，只支持有向边。

Compatibility

由于 Nebula Graph 3.1.0 的数据模型中，允许存在"悬挂边"，因此在增删时，用户需自行保证“一条边所对应的起点和终点”的存在性。详见 INSERT VERTEX、DELETE VERTEX、INSERT EDGE、DELETE EDGE。

不支持 openCypher 中的 MERGE 语句。

最后更新: June 20, 2023

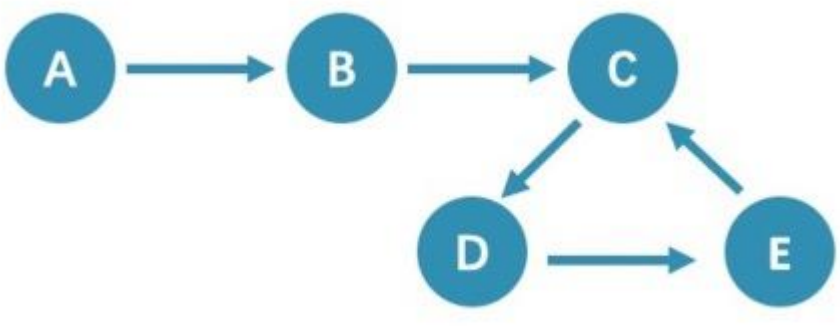
2022 vesoft Inc.

2.6 路径

图论中一个非常重要的概念是路径，路径是指一个有限或无限的边序列，这些边连接着一系列点。

路径的类型分为三种：`walk`、`trail`、`path`。关于路径的详细说明，请参见维基百科。

本文以下图为例进行简单介绍。



2.6.1 walk

`walk` 类型的路径由有限或无限的边序列构成。遍历时点和边可以重复。

查看示例图，由于 C、D、E 构成了一个环，因此该图包含无限个路径，例如 `A->B->C->D->E`、`A->B->C->D->E->C`、`A->B->C->D->E->C->D`。

> **Note**
>
> `GO` 语句采用的是 `walk` 类型路径。

2.6.2 trail

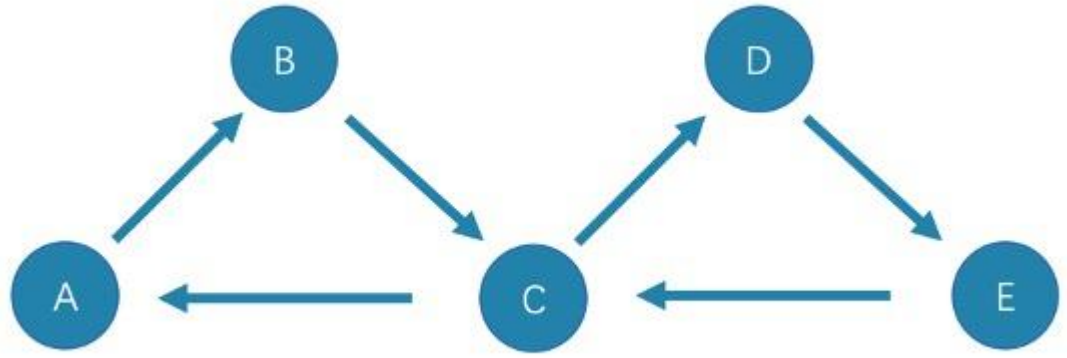
`trail` 类型的路径由有限的边序列构成。遍历时只有点可以重复，边不可以重复。柯尼斯堡七桥问题的路径类型就是 `trail`。

查看示例图，由于边不可以重复，所以该图包含有限个路径，最长路径由 5 条边组成：`A->B->C->D->E->C`。

> **Note**
>
> `MATCH`、`FIND PATH` 和 `GET SUBGRAPH` 语句采用的是 `trail` 类型路径。

在 trail 类型中，还有 `cycle` 和 `circuit` 两种特殊的路径类型，以下图为例对这两种特殊的路径类型进行介绍。

2022 vesoft Inc.



- cycle

  `cycle` 是封闭的 `trail` 类型的路径，遍历时边不可以重复，起点和终点重复，并且没有其他点重复。在此示例图中，最长路径由三条边组成: `A->B->C->A` 或 `C->D->E->C` .

- circuit

  `circuit` 也是封闭的 `trail` 类型的路径，遍历时边不可以重复，除起点和终点重复外，可能存在其他点重复。在此示例图中，最长路径为: `A->B->C->D->E->C->A` 。

2.6.3 path

`path` 类型的路径由有限的边序列构成。遍历时点和边都不可以重复。

查看示例图，由于点和边都不可以重复，所以该图包含有限个路径，最长路径由 4 条边组成: `A->B->C->D->E` 。

2.6.4 视频

用户也可以观看视频了解路径的相关概念。

Path（03 分 09 秒）

最后更新: June 20, 2023

2022 vesoft Inc.

2.7 点 VID

在 Nebula Graph 中，一个点由点的 ID 唯一标识，即 VID 或 Vertex ID。

2.7.1 VID 的特点

- VID 数据类型只可以为定长字符串 `FIXED_STRING(<N>)` 或 `INT64`；一个图空间只能选用其中一种 VID 类型。
- VID 在一个图空间中必须唯一，其作用类似于关系型数据库中的主键（索引+唯一约束）。但不同图空间中的 VID 是完全独立无关的。
- 点 VID 的生成方式必须由用户自行指定，系统不提供自增 ID 或者 UUID。
- VID 相同的点，会被认为是同一个点。例如：
- VID 相当于一个实体的唯一标号，例如一个人的身份证号。Tag 相当于实体所拥有的类型，例如"滴滴司机"和"老板"。不同的 Tag 又相应定义了两组不同的属性，例如"驾照号、驾龄、接单量、接单小号"和"工号、薪水、债务额度、商务电话"。
- 同时操作相同 VID 并且相同 Tag 的两条 `INSERT` 语句（均无 `IF NOT EXISTS` 参数），晚写入的 `INSERT` 会覆盖先写入的。
- 同时操作包含相同 VID 但是两个不同 `TAG A` 和 `TAG B` 的两条 `INSERT` 语句，对 `TAG A` 的操作不会影响 `TAG B`。
- VID 通常会被（LSM-tree 方式）索引并缓存在内存中，因此直接访问 VID 的性能最高。

2.7.2 VID 使用建议

- Nebula Graph 1.x 只支持 VID 类型为 `INT64`，从 2.x 开始支持 `INT64` 和 `FIXED_STRING(<N>)`。在 `CREATE SPACE` 中通过参数 `vid_type` 可以指定 VID 类型。
- 可以使用 `id()` 函数，指定或引用该点的 VID；
- 可以使用 `LOOKUP` 或者 `MATCH` 语句，来通过属性索引查找对应的 VID;
- 性能上，直接通过 VID 找到点的语句性能最高，例如 `DELETE xxx WHERE id(xxx) == "player100"`，或者 `GO FROM "player100"` 等语句。通过属性先查找 VID，再进行图操作的性能会变差，例如 `LOOKUP | GO FROM $-.ids` 等语句，相比前者多了一次内存或硬盘的随机读（`LOOKUP`）以及一次序列化（`|`）。

2.7.3 VID 生成建议

VID 的生成工作完全交给应用端，有一些通用的建议：

- （最优）通过有唯一性的主键或者属性来直接作为 VID；属性访问依赖于 VID;
- 通过有唯一性的属性组合来生成 VID，属性访问依赖于属性索引。
- 通过 snowflake 等算法生成 VID，属性访问依赖于属性索引；
- 如果个别记录的主键特别长，但绝大多数记录的主键都很短的情况，不要将 `FIXED_STRING(<N>)` 的 `N` 设置成超大，这会浪费大量内存和硬盘，也会降低性能。此时可通过 BASE64，MD5，hash 编码加拼接的方式来生成。
- 如果用 hash 方式生成 int64 VID：在有 10 亿个点的情况下，发生 hash 冲突的概率大约是 1/10。边的数量与碰撞的概率无关。

2.7.4 定义和修改 VID 与其数据类型

VID 的数据类型必须在创建图空间时定义，且一旦定义无法修改。

VID 必须在插入点时设置，且一旦设置无法修改。

2.7.5 "查询起始点"(`start vid`) 与全局扫描

绝大多数情况下，Nebula Graph 的查询语句（`MATCH`、`GO`、`LOOKUP`）的执行计划，必须要通过一定方式找到查询起始点的 VID（`start vid`）。

2022 vesoft Inc.

定位 `start vid` 只有两种方式:

1. 例如 `GO FROM "player100" OVER` 是在语句中显式的指明 `start vid` 是 "player100";
2. 例如 `LOOKUP ON player WHERE player.name == "Tony Parker"` 或者 `MATCH (v:player {name:"Tony Parker"})`，是通过属性 `player.name` 的索引来定位到 `start vid`;



`match (n) return n;` 会返回错误 `Scan vertices or edges need to specify a limit number, or limit number can not push down.`，这是一个全局扫描，需要用 `LIMIT` 子句限制返回数量才能执行。

最后更新: June 20, 2023

2022 vesoft Inc.

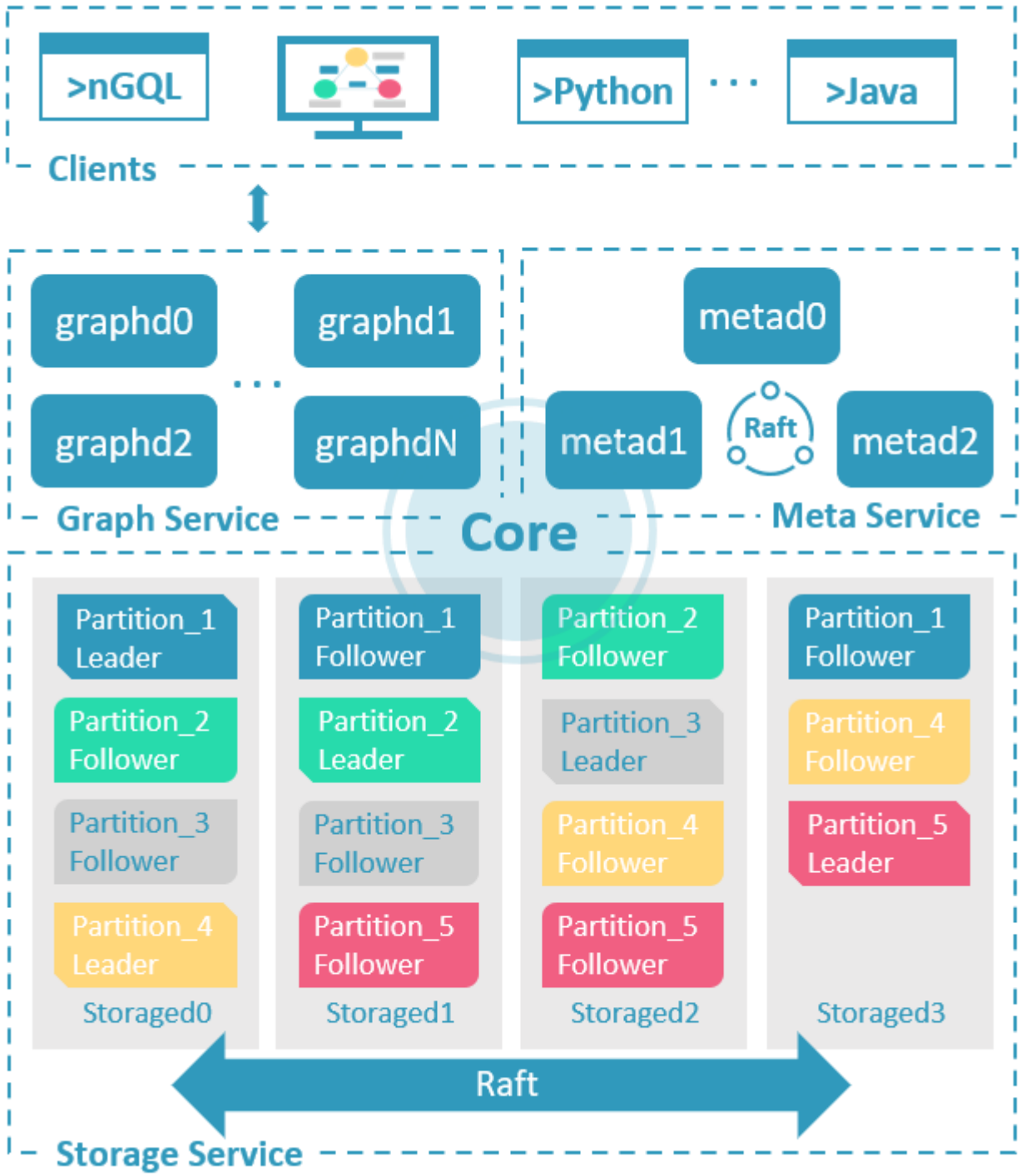
2.8 服务架构

2.8.1 Nebula Graph 架构总览

Nebula Graph 由三种服务构成：Graph 服务、Meta 服务和 Storage 服务，是一种存储与计算分离的架构。

每个服务都有可执行的二进制文件和对应进程，用户可以使用这些二进制文件在一个或多个计算机上部署 Nebula Graph 集群。

下图展示了 Nebula Graph 集群的经典架构。



2022 vesoft Inc.

Meta 服务

在 Nebula Graph 架构中，Meta 服务是由 nebula-metad 进程提供的，负责数据管理，例如 Schema 操作、集群管理和用户权限管理等。

Meta 服务的详细说明，请参见 Meta 服务。

Graph 服务和 Storage 服务

Nebula Graph 采用计算存储分离架构。Graph 服务负责处理计算请求，Storage 服务负责存储数据。它们由不同的进程提供，Graph 服务是由 nebula-graphd 进程提供，Storage 服务是由 nebula-storaged 进程提供。计算存储分离架构的优势如下：

- 易扩展

  分布式架构保证了 Graph 服务和 Storage 服务的灵活性，方便扩容和缩容。

- 高可用

  如果提供 Graph 服务的服务器有一部分出现故障，其余服务器可以继续为客户端提供服务，而且 Storage 服务存储的数据不会丢失。服务恢复速度较快，甚至能做到用户无感知。

- 节约成本

  计算存储分离架构能够提高资源利用率，而且可根据业务需求灵活控制成本。

- 更多可能性

  基于分离架构的特性，Graph 服务将可以在更多类型的存储引擎上单独运行，Storage 服务也可以为多种目的计算引擎提供服务。

Graph 服务和 Storage 服务的详细说明，请参见 Graph 服务和 Storage 服务。

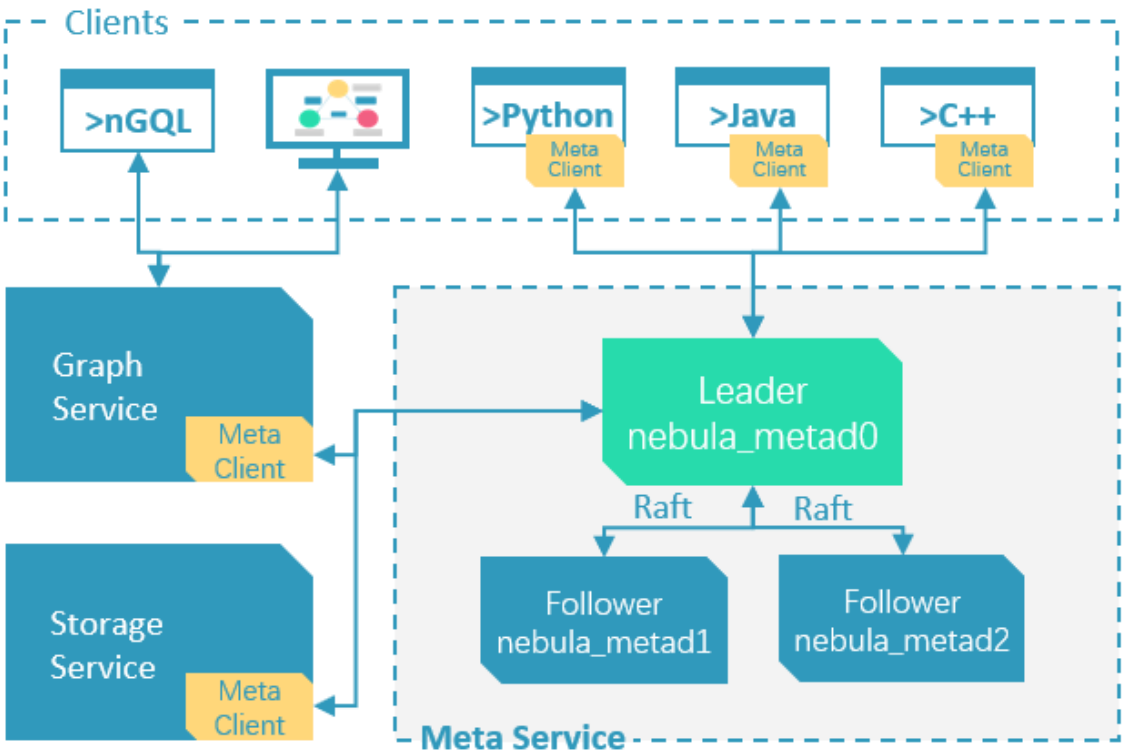
最后更新: June 20, 2023

2022 vesoft Inc.

2.8.2 Meta 服务

本文介绍 Meta 服务的架构和功能。

Meta 服务架构



Meta 服务是由 nebula-metad 进程提供的，用户可以根据场景配置 nebula-metad 进程数量：

- 测试环境中，用户可以在 Nebula Graph 集群中部署 1 个或 3 个 nebula-metad 进程。如果要部署 3 个，用户可以将它们部署在 1 台机器上，或者分别部署在不同的机器上。
- 生产环境中，建议在 Nebula Graph 集群中部署 3 个 nebula-metad 进程。请将这些进程部署在不同的机器上以保证高可用。

所有 nebula-metad 进程构成了基于 Raft 协议的集群，其中一个进程是 leader，其他进程都是 follower。

leader 是由多数派选举出来，只有 leader 能够对客户端或其他组件提供服务，其他 follower 作为候补，如果 leader 出现故障，会在所有 follower 中选举出新的 leader。

> **Note**
>
> leader 和 follower 的数据通过 Raft 协议保持一致，因此 leader 故障和选举新 leader 不会导致数据不一致。更多关于 Raft 的介绍见 Storage 服务。

Meta 服务功能

管理用户账号

Meta 服务中存储了用户的账号和权限信息，当客户端通过账号发送请求给 Meta 服务，Meta 服务会检查账号信息，以及该账号是否有对应的请求权限。

2022 vesoft Inc.

更多 Nebula Graph 的访问控制说明，请参见身份验证。

管理分片

Meta 服务负责存储和管理分片的位置信息，并且保证分片的负载均衡。

管理图空间

Nebula Graph 支持多个图空间，不同图空间内的数据是安全隔离的。Meta 服务存储所有图空间的元数据（非完整数据），并跟踪数据的变更，例如增加或删除图空间。

管理 **SCHEMA** 信息

Nebula Graph 是强类型图数据库，它的 Schema 包括 Tag、Edge type、Tag 属性和 Edge type 属性。

Meta 服务中存储了 Schema 信息，同时还负责 Schema 的添加、修改和删除，并记录它们的版本。

更多 Nebula Graph 的 Schema 信息，请参见数据模型。

管理 **TTL** 信息

Meta 服务存储 TTL（Time To Live）定义信息，可以用于设置数据生命周期。数据过期后，会由 Storage 服务进行处理，具体过程参见 TTL。

管理作业

Meta 服务中的作业管理模块负责作业的创建、排队、查询和删除。

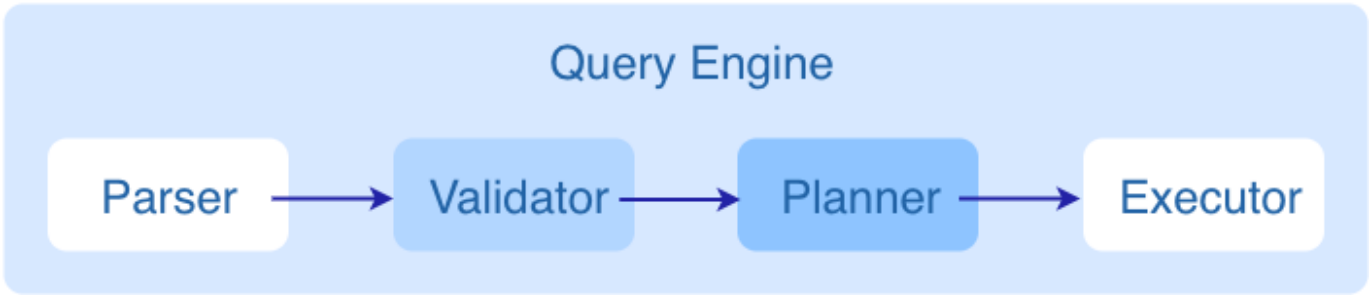
最后更新: June 20, 2023

2022 vesoft Inc.

2.8.3 Graph 服务

Graph 服务主要负责处理查询请求，包括解析查询语句、校验语句、生成执行计划以及按照执行计划执行四个大步骤，本文将基于这些步骤介绍 Graph 服务。

Graph 服务架构



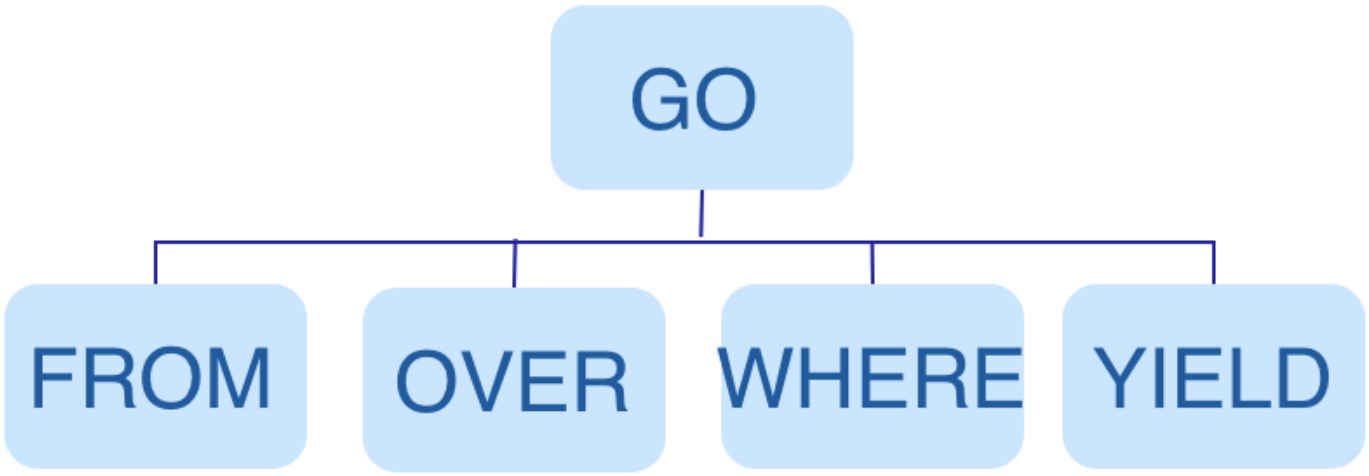
查询请求发送到 Graph 服务后，会由如下模块依次处理：

1. **Parser**：词法语法解析模块。
2. **Validator**：语义校验模块。
3. **Planner**：执行计划与优化器模块。
4. **Executor**：执行引擎模块。

Parser

Parser 模块收到请求后，通过 Flex（词法分析工具）和 Bison（语法分析工具）生成的词法语法解析器，将语句转换为抽象语法树（AST），在语法解析阶段会拦截不符合语法规则的语句。

例如 `GO FROM "Tim" OVER like WHERE properties(edge).likeness > 8.0 YIELD dst(edge)` 语句转换的 AST 如下。



2022 vesoft Inc.

Validator

Validator 模块对生成的 AST 进行语义校验，主要包括：

- 校验元数据信息

  校验语句中的元数据信息是否正确。

  例如解析 `OVER`、`WHERE` 和 `YIELD` 语句时，会查找 Schema 校验 Edge type、Tag 的信息是否存在，或者插入数据时校验插入的数据类型和 Schema 中的是否一致。

- 校验上下文引用信息

  校验引用的变量是否存在或者引用的属性是否属于变量。

  例如语句 `$var = GO FROM "Tim" OVER like YIELD dst(edge) AS ID; GO FROM $var.ID OVER serve YIELD dst(edge)`，Validator 模块首先会检查变量 `var` 是否定义，其次再检查属性 `ID` 是否属于变量 `var`。

- 校验类型推断

  推断表达式的结果类型，并根据子句校验类型是否正确。

  例如 `WHERE` 子句要求结果是 `bool`、`null` 或者 `empty`。

- 校验 `*` 代表的信息

  查询语句中包含 `*` 时，校验子句时需要将 `*` 涉及的 Schema 都进行校验。

  例如语句 `GO FROM "Tim" OVER * YIELD dst(edge), properties(edge).likeness, dst(edge)`，校验 `OVER` 子句时需要校验所有的 Edge type，如果 Edge type 包含 `like` 和 `serve`，该语句会展开为 `GO FROM "Tim" OVER like,serve YIELD dst(edge), properties(edge).likeness, dst(edge)`。

- 校验输入输出

  校验管道符（|）前后的一致性。

  例如语句 `GO FROM "Tim" OVER like YIELD dst(edge) AS ID | GO FROM $-.ID OVER serve YIELD dst(edge)`，Validator 模块会校验 `$-.ID` 在管道符左侧是否已经定义。

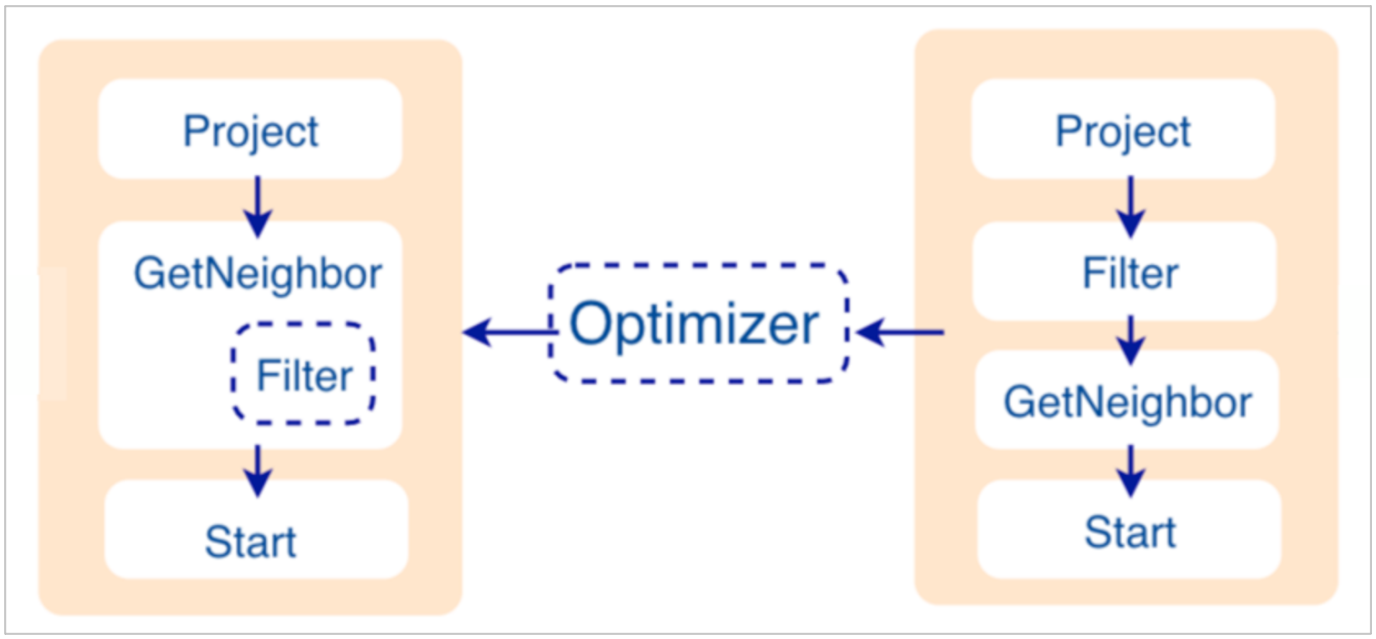
校验完成后，Validator 模块还会生成一个默认可执行，但是未进行优化的执行计划，存储在目录 `src/planner` 内。

Planner

如果配置文件 `nebula-graphd.conf` 中 `enable_optimizer` 设置为 `false`，Planner 模块不会优化 Validator 模块生成的执行计划，而是直接交给 Executor 模块执行。

如果配置文件 `nebula-graphd.conf` 中 `enable_optimizer` 设置为 `true`，Planner 模块会对 Validator 模块生成的执行计划进行优化。如下图所示。

2022 vesoft Inc.



- 优化前

  如上图右侧未优化的执行计划，每个节点依赖另一个节点，例如根节点 `Project` 依赖 `Filter` 、`Filter` 依赖 `GetNeighbor`，最终找到叶子节点 `Start`，才能开始执行（并非真正执行）。

  在这个过程中，每个节点会有对应的输入变量和输出变量，这些变量存储在一个哈希表中。由于执行计划不是真正执行，所以哈希表中每个 key 的 value 值都为空（除了 `Start` 节点，起始数据会存储在该节点的输入变量中）。哈希表定义在仓库 `nebula-graph` 内的 `src/context/ExecutionContext.cpp` 中。

  例如哈希表的名称为 `ResultMap`，在建立 `Filter` 这个节点时，定义该节点从 `ResultMap["GN1"]` 中读取数据，然后将结果存储在 `ResultMap["Filter2"]` 中，依次类推，将每个节点的输入输出都确定好。

- 优化过程

  Planner 模块目前的优化方式是 RBO（rule-based optimization），即预定义优化规则，然后对 Validator 模块生成的默认执行计划进行优化。新的优化规则 CBO（cost-based optimization）正在开发中。优化代码存储在仓库 `nebula-graph` 的目录 `src/optimizer/` 内。

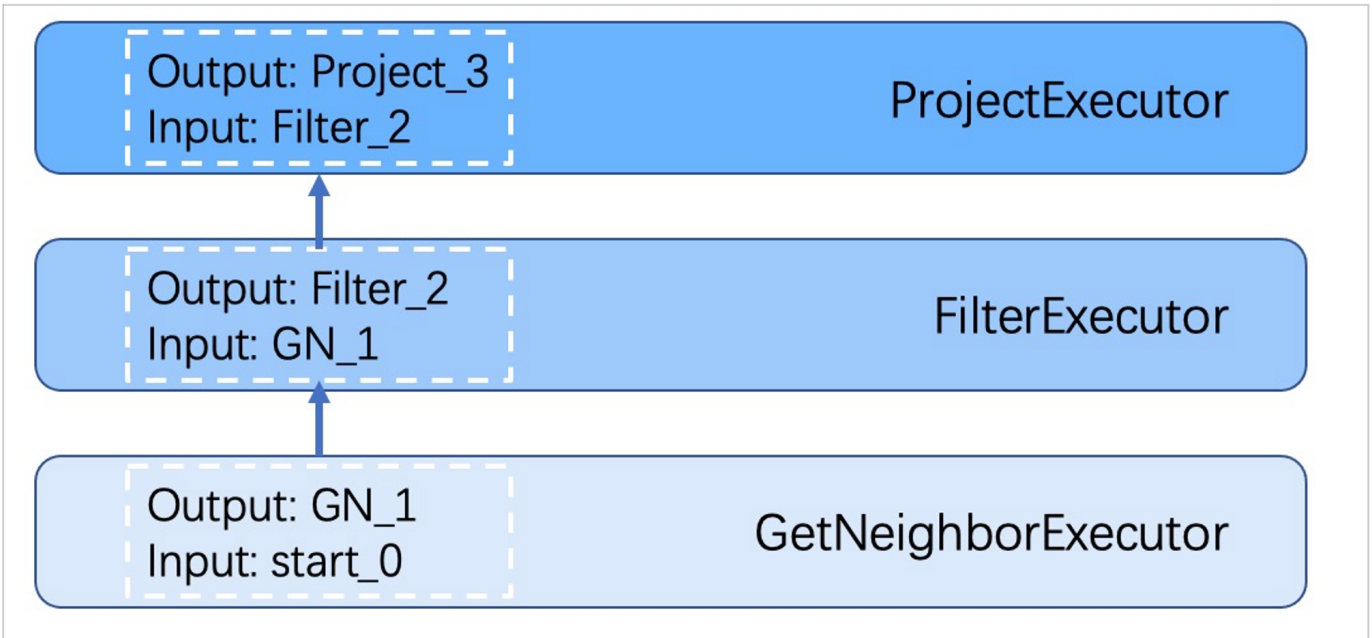
  RBO 是一个自底向上的探索过程，即对于每个规则而言，都会由执行计划的根节点（示例是 `Project`）开始，一步步向下探索到最底层的节点，在过程中查看是否可以匹配规则。

  如上图所示，探索到节点 `Filter` 时，发现依赖的节点是 `GetNeighbor`，匹配预先定义的规则，就会将 `Filter` 融入到 `GetNeighbor` 中，然后移除节点 `Filter`，继续匹配下一个规则。在执行阶段，当算子 `GetNeighbor` 调用 Storage 服务的接口获取一个点的邻边时，Storage 服务内部会直接将不符合条件的边过滤掉，这样可以极大地减少传输的数据量，该优化称为过滤下推。

Executor

Executor 模块包含调度器（Scheduler）和执行器（Executor），通过调度器调度执行计划，让执行器根据执行计划生成对应的执行算子，从叶子节点开始执行，直到根节点结束。如下图所示。

2022 vesoft Inc.



每一个执行计划节点都一一对应一个执行算子，节点的输入输出在优化执行计划时已经确定，每个算子只需要拿到输入变量中的值进行计算，最后将计算结果放入对应的输出变量中即可，所以只需要从节点 `Start` 一步步执行，最后一个算子的输出变量会作为最终结果返回给客户端。

代码结构

Nebula Graph 的代码层次结构如下：

```
|--src
   |--context    //校验期和执行期上下文
   |--daemons
   |--executor   //执行算子
   |--mock
   |--optimizer  //优化规则
   |--parser     //词法语法分析
   |--planner    //执行计划结构
   |--scheduler  //调度器
   |--service
   |--util       //基础组件
   |--validator  //语句校验
   |--visitor
```

视频

用户也可以通过视频全方位了解 Nebula Graph 的查询引擎。

- nMeetup·上海 | 全面解析 Query Engine（33 分 30 秒）

最后更新: June 20, 2023

2022 vesoft Inc.

2.8.4 Storage 服务

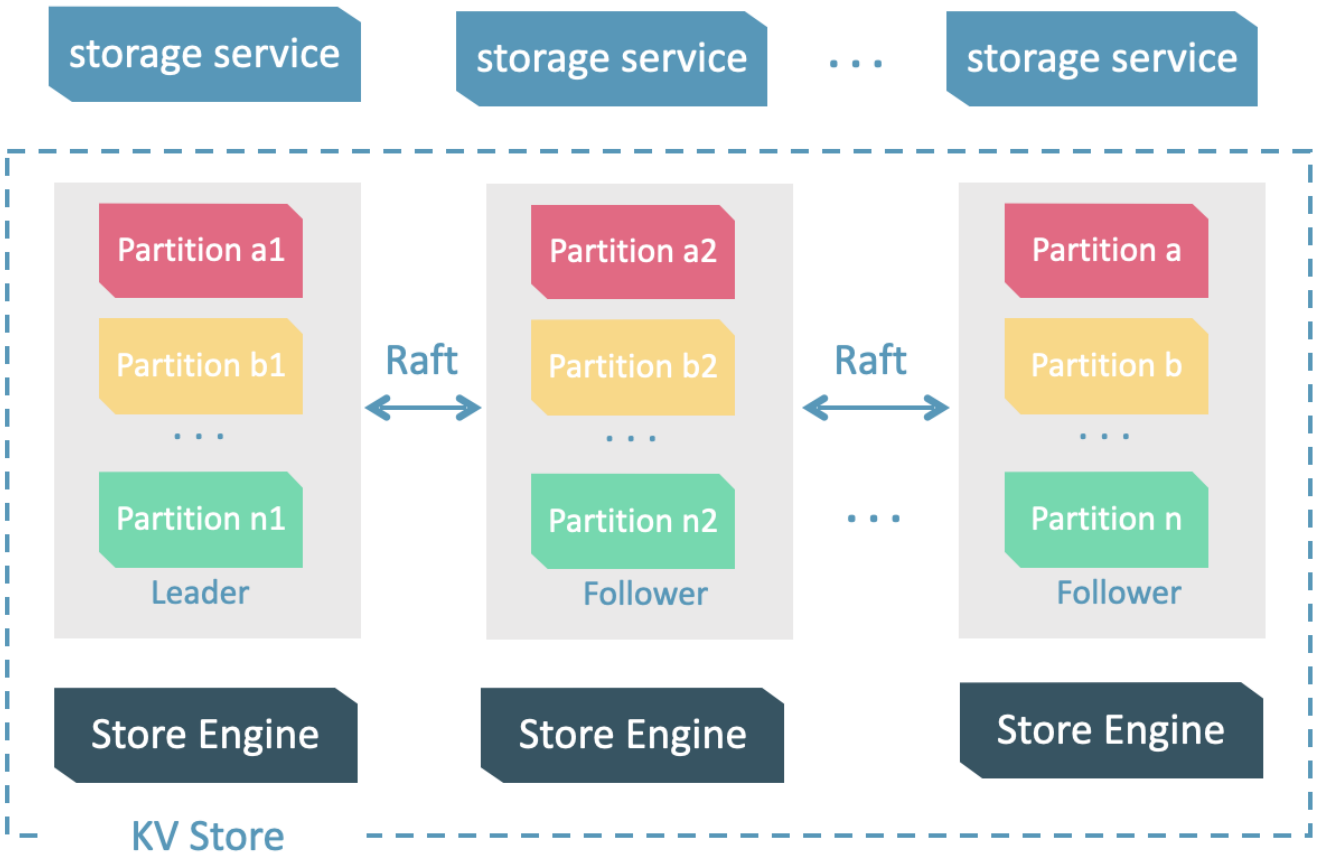
Nebula Graph 的存储包含两个部分，一个是 Meta 相关的存储，称为 Meta 服务，在前文已有介绍。

另一个是具体数据相关的存储，称为 Storage 服务。其运行在 nebula-storaged 进程中。本文仅介绍 Storage 服务的架构设计。

优势

- 高性能（自研 KVStore）
- 易水平扩展（Shared-nothing 架构，不依赖 NAS 等硬件设备）
- 强一致性（Raft）
- 高可用性（Raft）
- 支持向第三方系统进行同步（例如全文索引）

Storage 服务架构



Storage 服务是由 nebula-storaged 进程提供的，用户可以根据场景配置 nebula-storaged 进程数量，例如测试环境 1 个，生产环境 3 个。

2022 vesoft Inc.

所有 nebula-storaged 进程构成了基于 Raft 协议的集群，整个服务架构可以分为三层，从上到下依次为：

- Storage interface 层

  Storage 服务的最上层，定义了一系列和图相关的 API。API 请求会在这一层被翻译成一组针对分片的 KV 操作，例如：

- `getNeighbors`：查询一批点的出边或者入边，返回边以及对应的属性，并且支持条件过滤。
- `insert vertex/edge`：插入一条点或者边及其属性。
- `getProps`：获取一个点或者一条边的属性。

  正是这一层的存在，使得 Storage 服务变成了真正的图存储，否则 Storage 服务只是一个 KV 存储服务。

- Consensus 层

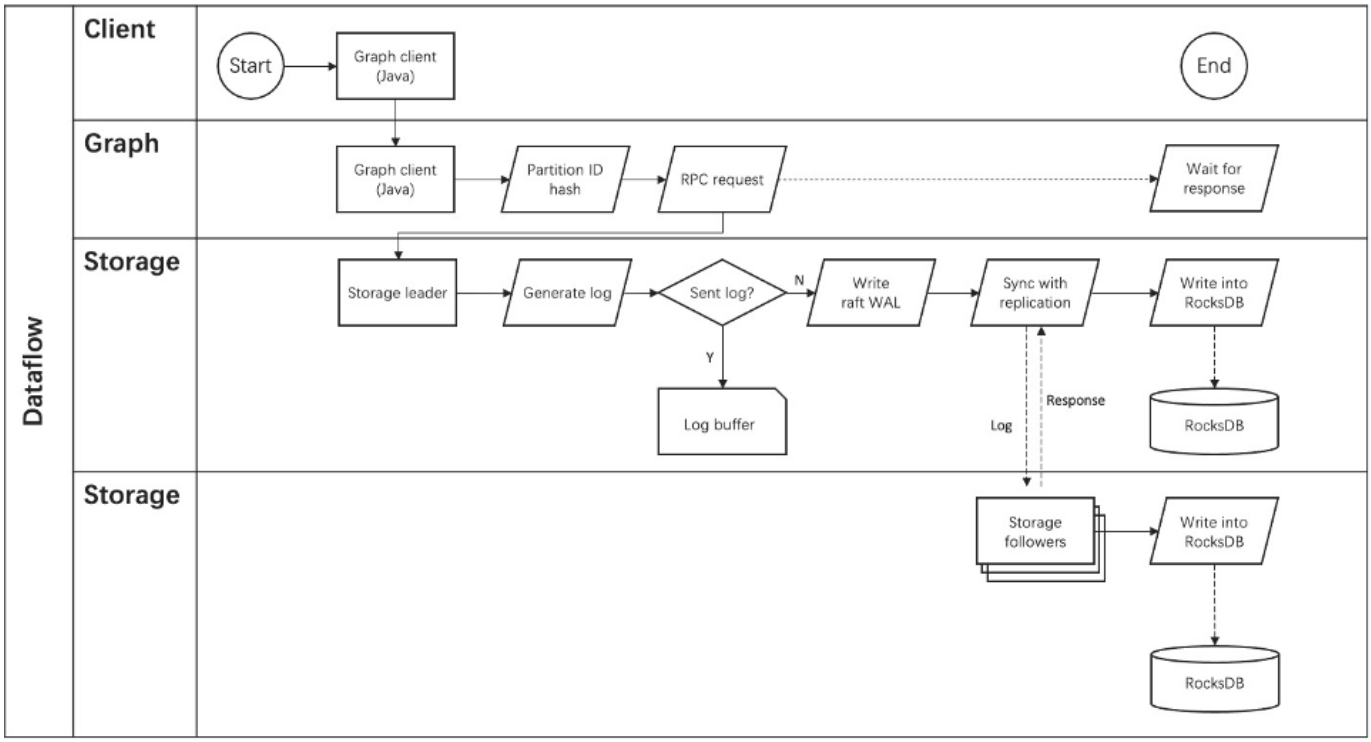
  Storage 服务的中间层，实现了 Multi Group Raft，保证强一致性和高可用性。

- Store Engine 层

  Storage 服务的最底层，是一个单机版本地存储引擎，提供对本地数据的 `get`、`put`、`scan` 等操作。相关接口存储在 `KVStore.h` 和 `KVEngine.h` 文件，用户可以根据业务需求定制开发相关的本地存储插件。

下文将基于架构介绍 Storage 服务的部分特性。

Storage 写入流程



KVStore

Nebula Graph 使用自行开发的 KVStore，而不是其他开源 KVStore，原因如下：

- 需要高性能 KVStore。
- 需要以库的形式提供，实现高效计算下推。对于强 Schema 的 Nebula Graph 来说，计算下推时如何提供 Schema 信息，是高效的关键。
- 需要数据强一致性。

2022 vesoft Inc.

基于上述原因，Nebula Graph 使用 RocksDB 作为本地存储引擎，实现了自己的 KVStore，有如下优势：

- 对于多硬盘机器，Nebula Graph 只需配置多个不同的数据目录即可充分利用多硬盘的并发能力。
- 由 Meta 服务统一管理所有 Storage 服务，可以根据所有分片的分布情况和状态，手动进行负载均衡。

> **Note**
>
> 不支持自动负载均衡是为了防止自动数据搬迁影响线上业务。

- 定制预写日志（WAL），每个分片都有自己的 WAL。
- 支持多个图空间，不同图空间相互隔离，每个图空间可以设置自己的分片数和副本数。

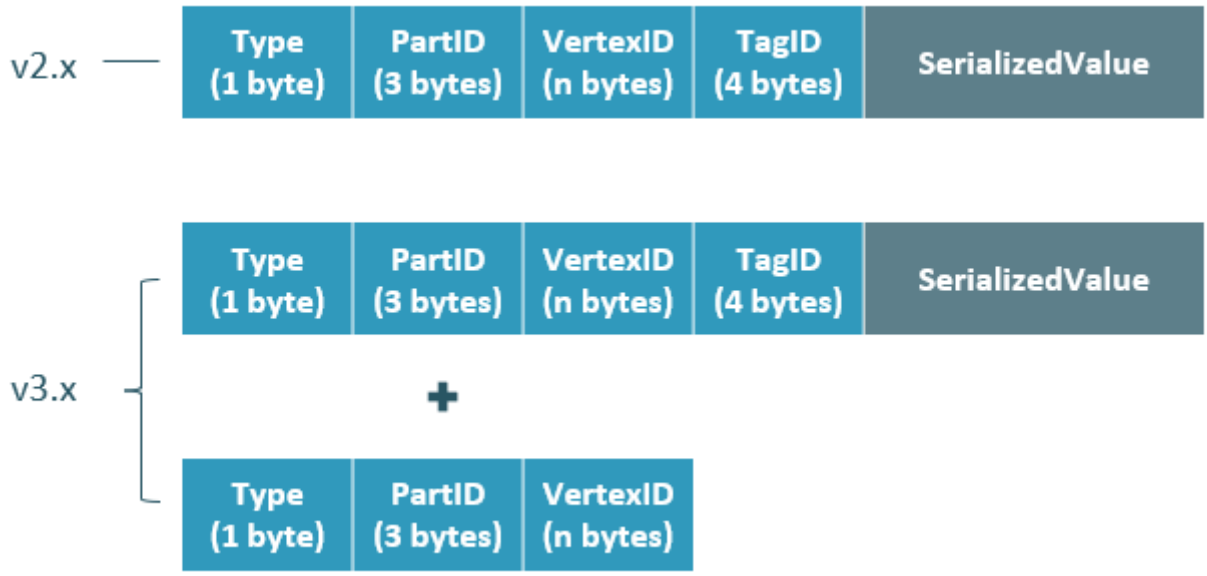
2022 vesoft Inc.

数据存储格式

图存储的主要数据是点和边，Nebula Graph 将点和边的信息存储为 key，同时将点和边的属性信息存储在 value 中，以便更高效地使用属性过滤。

- 点数据存储格式

  相比 Nebula Graph 2.x 版本，3.x 版本的每个点多了一个不含 TagID 字段并且无 value 的 key，用于支持无 Tag 的点。



字段	说明
`Type`	key 类型。长度为 1 字节。
`PartID`	数据分片编号。长度为 3 字节。此字段主要用于 Storage 负载均衡（balance）时方便根据前缀扫描整个分片的数据。
`VertexID`	点 ID。当点 ID 类型为 int 时，长度为 8 字节；当点 ID 类型为 string 时，长度为创建图空间时指定的 `fixed_string` 长度。
`TagID`	点关联的 Tag ID。长度为 4 字节。
`SerializedValue`	序列化的 value，用于保存点的属性信息。

2022 vesoft Inc.

- 边数据存储格式

Type (1 byte)	PartID (3 bytes)	VertexID (n bytes)	EdgeType (4 bytes)	Rank (8 bytes)	VertexID (n bytes)	PlaceHolder (1 byte)	SerializedValue

字段	说明
`Type`	key 类型。长度为 1 字节。
`PartID`	数据分片编号。长度为 3 字节。此字段主要用于 Storage 负载均衡（balance）时方便根据前缀扫描整个分片的数据。
`VertexID`	点 ID。前一个 `VertexID` 在出边里表示起始点 ID，在入边里表示目的点 ID；后一个 `VertexID` 出边里表示目的点 ID，在入边里表示起始点 ID。
`Edge type`	边的类型。大于 0 表示出边，小于 0 表示入边。长度为 4 字节。
`Rank`	用来处理两点之间有多个同类型边的情况。用户可以根据自己的需求进行设置，例如存放交易时间、交易流水号等。长度为 8 字节，
`PlaceHolder`	预留字段。长度为 1 字节。
`SerializedValue`	序列化的 value，用于保存边的属性信息。

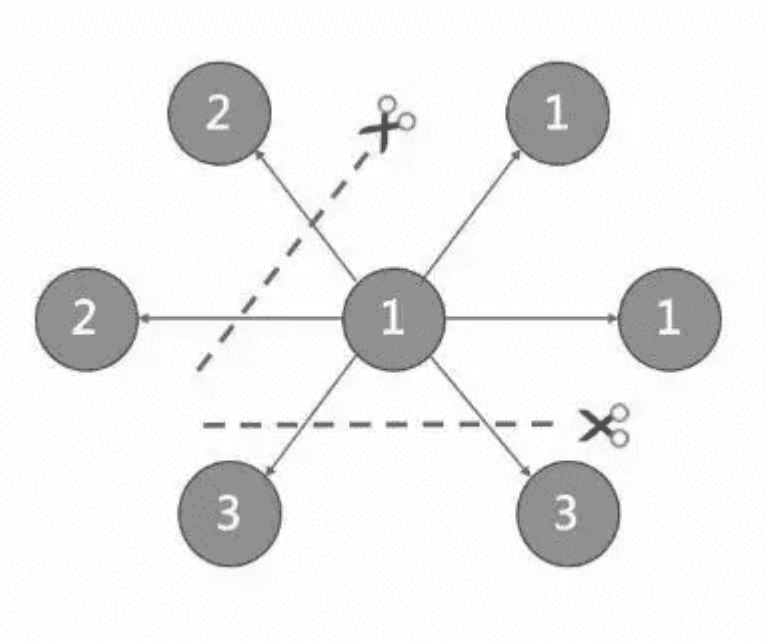
属性说明

Nebula Graph 使用强类型 Schema。

对于点或边的属性信息，Nebula Graph 会将属性信息编码后按顺序存储。由于属性的长度是固定的，查询时可以根据偏移量快速查询。在解码之前，需要先从 Meta 服务中查询具体的 Schema 信息（并缓存）。同时为了支持在线变更 Schema，在编码属性时，会加入对应的 Schema 版本信息。

数据分片

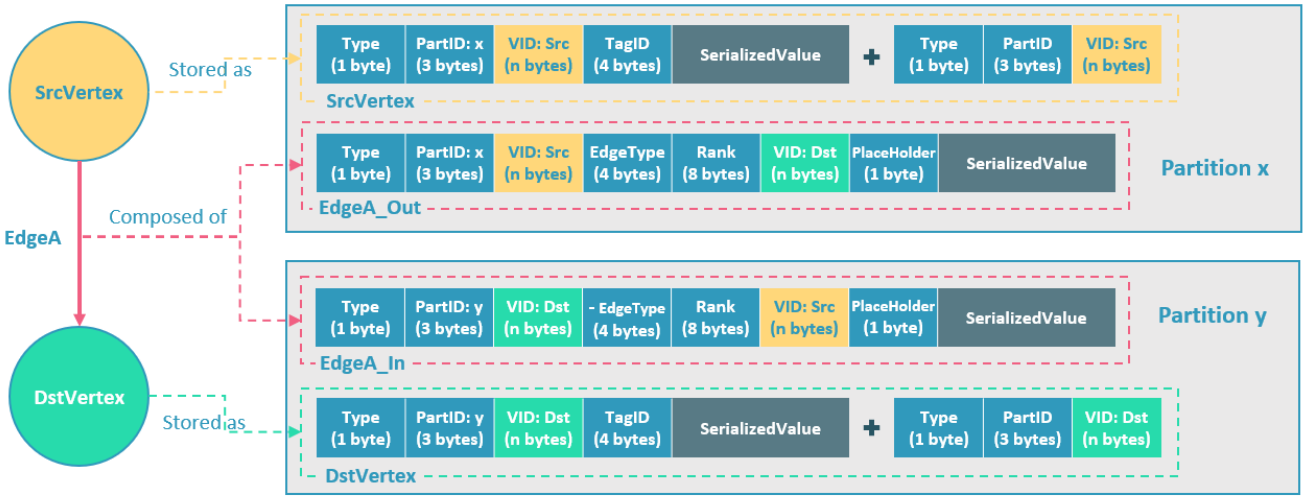
由于超大规模关系网络的节点数量高达百亿到千亿，而边的数量更会高达万亿，即使仅存储点和边两者也远大于一般服务器的容量。因此需要有方法将图元素切割，并存储在不同逻辑分片（Partition）上。Nebula Graph 采用边分割的方式。



2022 vesoft Inc.

切边与存储放大

Nebula Graph 中逻辑上的一条边对应着硬盘上的两个键值对（key-value pair），在边的数量和属性较多时，存储放大现象较明显。边的存储方式如下图所示。



上图以最简单的两个点和一条边为例，起点 SrcVertex 通过边 EdgeA 连接目的点 DstVertex，形成路径 `(SrcVertex)-[EdgeA]->(DstVertex)`。这两个点和一条边会以 6 个键值对的形式保存在存储层的两个不同分片，即 Partition x 和 Partition y 中，详细说明如下：

- 点 SrcVertex 的键值保存在 Partition x 中。
- 边 EdgeA 的第一份键值，这里用 EdgeA_Out 表示，与 SrcVertex 一同保存在 Partition x 中。key 的字段有 Type、PartID（x）、VID（Src，即点 SrcVertex 的 ID）、EdgeType（符号为正，代表边方向为出）、Rank（0）、VID（Dst，即点 DstVertex 的 ID）和 PlaceHolder。SerializedValue 即 Value，是序列化的边属性。
- 点 DstVertex 的键值保存在 Partition y 中。
- 边 EdgeA 的第二份键值，这里用 EdgeA_In 表示，与 DstVertex 一同保存在 Partition y 中。key 的字段有 Type、PartID（y）、VID（Dst，即点 DstVertex 的 ID）、EdgeType（符号为负，代表边方向为入）、Rank（0）、VID（Src，即点 SrcVertex 的 ID）和 PlaceHolder。SerializedValue 即 Value，是序列化的边属性，与 EdgeA_Out 中该部分的完全相同。

EdgeA_Out 和 EdgeA_In 以方向相反的两条边的形式存在于存储层，二者组合成了逻辑上的一条边 EdgeA。EdgeA_Out 用于从起点开始的遍历请求，例如 `(a)-[]->()`；EdgeA_In 用于指向目的点的遍历请求，或者说从目的点开始，沿着边的方向逆序进行的遍历请求，例如例如 `()-[]->(a)`。

如 EdgeA_Out 和 EdgeA_In 一样，Nebula Graph 冗余了存储每条边的信息，导致存储边所需的实际空间翻倍。因为边对应的 key 占用的硬盘空间较小，但 value 占用的空间与属性值的长度和数量成正比，所以，当边的属性值较大或数量较多时候，硬盘空间占用量会比较大。

如果对边进行操作，为了保证两个键值对的最终一致性，可以开启 TOSS 功能，开启后，会先在正向边所在的分片进行操作，然后在反向边所在分片进行操作，最后返回结果。

分片算法

分片策略采用静态 **Hash** 的方式，即对点 VID 进行取模操作，同一个点的所有 Tag、出边和入边信息都会存储到同一个分片，这种方式极大地提升了查询效率。

> **Note**
>
> 创建图空间时需指定分片数量，分片数量设置后无法修改，建议设置时提前满足业务将来的扩容需求。

多机集群部署时，分片分布在集群内的不同机器上。分片数量在 CREATE SPACE 语句中指定，此后不可更改。

如果需要将某些点放置在相同的分片（例如在一台机器上），可以参考公式或代码。

2022 vesoft Inc.

下文用简单代码说明 VID 和分片的关系。

```
// 如果 ID 长度为 8，为了兼容 1.0，将数据类型视为 int64。
uint64_t vid = 0;
if (id.size() == 8) {
    memcpy(static_cast<void*>(&vid), id.data(), 8);
} else {
    MurmurHash2 hash;
    vid = hash(id.data());
}
PartitionID pId = vid % numParts + 1;
```

简单来说，上述代码是将一个固定的字符串进行哈希计算，转换成数据类型为 int64 的数字（int64 数字的哈希计算结果是数字本身），将数字取模，然后加 1，即：

```
pId = vid % numParts + 1;
```

示例的部分参数说明如下。

参数	说明
`%`	取模运算。
`numParts`	`VID` 所在图空间的分片数，即 CREATE SPACE 语句中的 `partition_num` 值。
`pId`	`VID` 所在分片的 ID。

例如有 100 个分片，`VID` 为 1、101 和 1001 的三个点将会存储在相同的分片。分片 ID 和机器地址之间的映射是随机的，所以不能假定任何两个分片位于同一台机器上。

Raft

关于 RAFT 的简单介绍

分布式系统中，同一份数据通常会有多个副本，这样即使少数副本发生故障，系统仍可正常运行。这就需要一定的技术手段来保证多个副本之间的一致性。

基本原理：Raft 就是一种用于保证多副本一致性的协议。Raft 采用多个副本之间竞选的方式，赢得”超过半数”副本投票的（候选）副本成为 Leader，由 Leader 代表所有副本对外提供服务；其他 Follower 作为备份。当该 Leader 出现异常后（通信故障、运维命令等），其余 Follower 进行新一轮选举，投票出一个新的 Leader。Leader 和 Follower 之间通过心跳的方式相互探测是否存活，并以 Raft-wal 的方式写入硬盘，超过多个心跳仍无响应的副本会被认为发生故障。

> **Note**
>
> 因为 Raft-wal 需要定期写硬盘，如果硬盘写能力瓶颈会导致 Raft 心跳失败，导致重新发起选举。硬盘 IO 严重堵塞情况下，会导致长期无法选举出 Leader。

读写流程：对于客户端的每个写入请求，Leader 会将该写入以 Raft-wal 的方式，将该条同步给其他 Follower，并只有在“超过半数”副本都成功收到 Raft-wal 后，才会返回客户端该写入成功。对于客户端的每个读取请求，都直接访问 Leader，而 Follower 并不参与读请求服务。

故障流程：场景 1：考虑一个配置为单副本（图空间）的集群；如果系统只有一个副本时，其自身就是 Leader；如果其发生故障，系统将完全不可用。场景 2：考虑一个配置为 3 副本（图空间）的集群；如果系统有 3 个副本，其中一个副本是 Leader，其他 2 个副本是 Follower；即使原 Leader 发生故障，剩下两个副本仍可投票出一个新的 Leader（以及一个 Follower），此时系统仍可使用；但是当这 2 个副本中任一者再次发生故障后，由于投票人数不足，系统将完全不可用。

> **Note**
>
> Raft 多副本的方式与 HDFS 多副本的方式是不同的，Raft 基于“多数派”投票，因此副本数量不能是偶数。

2022 vesoft Inc.

MULTI GROUP RAFT

由于 Storage 服务需要支持集群分布式架构，所以基于 Raft 协议实现了 Multi Group Raft，即每个分片的所有副本共同组成一个 Raft group，其中一个副本是 leader，其他副本是 follower，从而实现强一致性和高可用性。Raft 的部分实现如下。

由于 Raft 日志不允许空洞，Nebula Graph 使用 Multi Group Raft 缓解此问题，分片数量较多时，可以有效提高 Nebula Graph 的性能。但是分片数量太多会增加开销，例如 Raft group 内部存储的状态信息、WAL 文件，或者负载过低时的批量操作。

实现 Multi Group Raft 有 2 个关键点：

- 共享 Transport 层

  每一个 Raft group 内部都需要向对应的 peer 发送消息，如果不能共享 Transport 层，会导致连接的开销巨大。

- 共享线程池

  如果不共享一组线程池，会造成系统的线程数过多，导致大量的上下文切换开销。

批量（BATCH）操作

Nebula Graph 中，每个分片都是串行写日志，为了提高吞吐，写日志时需要做批量操作，但是由于 Nebula Graph 利用 WAL 实现一些特殊功能，需要对批量操作进行分组，这是 Nebula Graph 的特色。

例如无锁 CAS 操作需要之前的 WAL 全部提交后才能执行，如果一个批量写入的 WAL 里包含了 CAS 类型的 WAL，就需要拆分成粒度更小的几个组，还要保证这几组 WAL 串行提交。

LEADER 切换（TRANSFER LEADERSHIP）

leader 切换对于负载均衡至关重要，当把某个分片从一台机器迁移到另一台机器时，首先会检查分片是不是 leader，如果是的话，需要先切换 leader，数据迁移完毕之后，通常还要重新均衡 leader 分布。

对于 leader 来说，提交 leader 切换命令时，就会放弃自己的 leader 身份，当 follower 收到 leader 切换命令时，就会发起选举。

成员变更

为了避免脑裂，当一个 Raft group 的成员发生变化时，需要有一个中间状态，该状态下新旧 group 的多数派需要有重叠的部分，这样就防止了新的 group 或旧的 group 单方面做出决定。为了更加简化，Diego Ongaro 在自己的博士论文中提出每次只增减一个 peer 的方式，以保证新旧 group 的多数派总是有重叠。Nebula Graph 也采用了这个方式，只不过增加成员和移除成员的实现有所区别。具体实现方式请参见 Raft Part class 里 addPeer/removePeer 的实现。

缓存

由于 RocksDB 本身的缓存管理存在限制，无法按需缓存点或边，所以 Nebula Graph 自行实现了 Storage 缓存管理，可以更自由地设置缓存大小、内容等。详情参见 storage cache 配置。

与 HDFS 的区别

Storage 服务基于 Raft 协议实现的分布式架构，与 HDFS 的分布式架构有一些区别。例如：

- Storage 服务本身通过 Raft 协议保证一致性，副本数量通常为奇数，方便进行选举 leader，而 HDFS 存储具体数据的 DataNode 需要通过 NameNode 保证一致性，对副本数量没有要求。
- Storage 服务只有 leader 副本提供读写服务，而 HDFS 的所有副本都可以提供读写服务。
- Storage 服务无法修改副本数量，只能在创建图空间时指定副本数量，而 HDFS 可以调整副本数量。
- Storage 服务是直接访问文件系统，而 HDFS 的上层（例如 HBase）需要先访问 HDFS，再访问到文件系统，远程过程调用（RPC）次数更多。

总而言之，Storage 服务更加轻量级，精简了一些功能，架构没有 HDFS 复杂，可以有效提高小块存储的读写性能。

最后更新: June 20, 2023

2022 vesoft Inc.

3. 快速入门

3.1 快速入门

快速入门将介绍如何简单地使用 Nebula Graph，包括部署、连接 Nebula Graph，以及基础的增删改查操作。

3.1.1 文档

按照以下步骤可以快速部署并且使用 Nebula Graph。

1. 安装 Nebula Graph

   使用 RPM 或 DEB 文件可以快速安装 Nebula Graph。关于其它部署方式及相应的准备工作，参见安装部署章节。

2. 启动 Nebula Graph

   部署好 Nebula Graph 之后需要启动 Nebula Graph 服务。

3. 连接 Nebula Graph

   启动 Nebula Graph 服务后即可使用客户端连接。Nebula Graph 支持多种客户端，快速入门中介绍使用原生命令行客户端 Nebula Console 连接 Nebula Graph 的方法。

4. 注册 Storage 服务

   首次连接 Nebula Graph 后需要先注册 Storage 服务才能正常查询数据。

5. 使用常用 nGQL（CRUD 命令）

   注册 Storage 服务之后即可使用 nGQL（Nebula Graph Query Language）进行增删改查。

3.1.2 视频

用户也可以观看视频了解 Nebula Graph 的相关概念和操作。

热点视频

- Foesa 小学姐课堂——Nebula Graph 那些磨人的概念（04 分 20 秒）

- Foesa 小学姐课堂——path 的三种类型（03 分 09 秒）

NG 辅导班

- 第一篇：图世界的那些概念、术语（08 分 12 秒）

- 第二篇：如何更好地学习 Nebula Graph（07 分 44 秒）

请访问 Bilibili 空间，查看 30 多个，500 多分钟的系列视频。

最后更新: June 20, 2023

2022 vesoft Inc.

3.2 步骤 1：安装 Nebula Graph

RPM 和 DEB 是 Linux 系统下常见的两种安装包格式，本文介绍如何使用 RPM 或 DEB 文件在一台机器上快速安装 Nebula Graph。

Note

部署 Nebula Graph 集群的方式参见使用 RPM/DEB 包部署集群。

Enterpriseonly

企业版请发送邮件至 inquiry@vesoft.com。

3.2.1 前提条件

安装 wget

3.2.2 下载安装包

阿里云 **OSS** 下载

- 下载 release 版本

  URL 格式如下：

```
//Centos 7
https://oss-cdn.nebula-graph.com.cn/package/<release_version>/nebula-graph-<release_version>.el7.x86_64.rpm

//Centos 8
https://oss-cdn.nebula-graph.com.cn/package/<release_version>/nebula-graph-<release_version>.el8.x86_64.rpm

//Ubuntu 1604
https://oss-cdn.nebula-graph.com.cn/package/<release_version>/nebula-graph-<release_version>.ubuntu1604.amd64.deb

//Ubuntu 1804
https://oss-cdn.nebula-graph.com.cn/package/<release_version>/nebula-graph-<release_version>.ubuntu1804.amd64.deb

//Ubuntu 2004
https://oss-cdn.nebula-graph.com.cn/package/<release_version>/nebula-graph-<release_version>.ubuntu2004.amd64.deb
```

  例如要下载适用于 `Centos 7.5` 的 `3.1.0` 安装包：

```
wget https://oss-cdn.nebula-graph.com.cn/package/3.1.0/nebula-graph-3.1.0.el7.x86_64.rpm
wget https://oss-cdn.nebula-graph.com.cn/package/3.1.0/nebula-graph-3.1.0.el7.x86_64.rpm.sha256sum.txt
```

  下载适用于 `ubuntu 1804` 的 `3.1.0` 安装包：

```
wget https://oss-cdn.nebula-graph.com.cn/package/3.1.0/nebula-graph-3.1.0.ubuntu1804.amd64.deb
wget https://oss-cdn.nebula-graph.com.cn/package/3.1.0/nebula-graph-3.1.0.ubuntu1804.amd64.deb.sha256sum.txt
```


2022 vesoft Inc.

- 下载日常开发版本 (nightly)

  Danger

  - nightly 版本通常用于测试新功能、新特性，请不要在生产环境中使用 nightly 版本。
  - nightly 版本不保证每日都能完整发布，也不保证是否会更改文件名。

  URL 格式如下：

```
//Centos 7
https://oss-cdn.nebula-graph.com.cn/package/nightly/<yyyy.mm.dd>/nebula-graph-<yyyy.mm.dd>-nightly.el7.x86_64.rpm

//Centos 8
https://oss-cdn.nebula-graph.com.cn/package/nightly/<yyyy.mm.dd>/nebula-graph-<yyyy.mm.dd>-nightly.el8.x86_64.rpm

//Ubuntu 1604
https://oss-cdn.nebula-graph.com.cn/package/nightly/<yyyy.mm.dd>/nebula-graph-<yyyy.mm.dd>-nightly.ubuntu1604.amd64.deb

//Ubuntu 1804
https://oss-cdn.nebula-graph.com.cn/package/nightly/<yyyy.mm.dd>/nebula-graph-<yyyy.mm.dd>-nightly.ubuntu1804.amd64.deb

//Ubuntu 2004
https://oss-cdn.nebula-graph.com.cn/package/nightly/<yyyy.mm.dd>/nebula-graph-<yyyy.mm.dd>-nightly.ubuntu2004.amd64.deb
```

  例如要下载 `2021.11.24` 适用于 `Centos 7.5` 的 `2.x` 安装包：

```
wget https://oss-cdn.nebula-graph.com.cn/package/nightly/2021.11.24/nebula-graph-2021.11.24-nightly.el7.x86_64.rpm
wget https://oss-cdn.nebula-graph.com.cn/package/nightly/2021.11.24/nebula-graph-2021.11.24-nightly.el7.x86_64.rpm.sha256sum.txt
```

  要下载 `2021.11.24` 适用于 `Ubuntu 1804` 的 `2.x` 安装包：

```
wget https://oss-cdn.nebula-graph.com.cn/package/nightly/2021.11.24/nebula-graph-2021.11.24-nightly.ubuntu1804.amd64.deb
wget https://oss-cdn.nebula-graph.com.cn/package/nightly/2021.11.24/nebula-graph-2021.11.24-nightly.ubuntu1804.amd64.deb.sha256sum.txt
```

3.2.3 安装 Nebula Graph

- 安装 RPM 包

```
$ sudo rpm -ivh --prefix=<installation_path> <package_name>
```

  `--prefix` 为可选项，用于指定安装路径。如不设置，系统会将 Nebula Graph 安装到默认路径 `/usr/local/nebula/` 。

  例如，要在默认路径下安装3.1.0版本的 RPM 包，运行如下命令：

```
sudo rpm -ivh nebula-graph-3.1.0.el7.x86_64.rpm
```

- 安装 DEB 包

```
$ sudo dpkg -i <package_name>
```

  Note

  使用 DEB 包安装 Nebula Graph 时不支持自定义安装路径。默认安装路径为 `/usr/local/nebula/` 。

  例如安装3.1.0版本的 DEB 包：

```
sudo dpkg -i nebula-graph-3.1.0.ubuntu1804.amd64.deb
```


2022 vesoft Inc.

3.2.4 后续操作

- （企业版）设置 License
- 启动 Nebula Graph
- 连接 Nebula Graph

最后更新: June 20, 2023

2022 vesoft Inc.

3.3 步骤 2：启动 Nebula Graph 服务

Nebula Graph 支持通过脚本或 systemd 管理服务。本文详细介绍这两种方式。

Enterpriseonly

仅企业版支持使用 systemd 管理服务。

Danger

这两种方式互不兼容，选择使用其中一种。

3.3.1 使用脚本管理服务

使用脚本 `nebula.service` 管理服务，包括启动、停止、重启、中止和查看。

Note

`nebula.service` 的默认路径是 `/usr/local/nebula/scripts`，如果修改过安装路径，请使用实际路径。

语法

```
$ sudo /usr/local/nebula/scripts/nebula.service
[-v] [-c <config_file_path>]
<start | stop | restart | kill | status>
<metad | graphd | storaged | all>
```

参数	说明
`-v`	显示详细调试信息。
`-c`	指定配置文件路径，默认路径为 `/usr/local/nebula/etc/`。
`start`	启动服务。
`stop`	停止服务。
`restart`	重启服务。
`kill`	中止服务。
`status`	查看服务状态。
`metad`	管理 Meta 服务。
`graphd`	管理 Graph 服务。
`storaged`	管理 Storage 服务。
`all`	管理所有服务。

3.3.2 使用 systemd 管理服务

为方便使用，Nebula Graph 企业版支持用 systemd 管理服务，通过 `systemctl` 启动、停止、重启和查看服务。

2022 vesoft Inc.

Note

- 安装 Nebula Graph 企业版后，systemd 所需的 `.service` 文件在安装目录的 `etc/unit` 目录内，使用 RPM/DEB 包安装的 Nebula Graph，会自动将这些 `.service` 文件放入 `/usr/lib/systemd/system` 目录内，并且 `ExecStart` 也会根据指定的 Nebula Graph 安装路径进行生成，因此可以直接使用 `systemctl` 命令。
- 对于使用企业版 Dashboard 安装的企业版 Nebula Graph，不支持使用 `systemctl` 管理服务。
- 对于其他方式安装的企业版 Nebula Graph，需要用户手动将 `.service` 文件移动到 `/usr/lib/systemd/system` 目录内，并修改 `.service` 文件内的 `ExecStart` 的文件路径，才可以正常使用 `systemctl` 命令。

语法

```
$ systemctl <start | stop | restart | status > <nebula | nebula-metad | nebula-graphd | nebula-storaged>
```

参数	说明
`start`	启动服务。
`stop`	停止服务。
`restart`	重启服务。
`status`	查看服务状态。
`nebula`	管理所有服务。
`nebula-metad`	管理 Meta 服务。
`nebula-graphd`	管理 Graph 服务。
`nebula-storaged`	管理 Storage 服务。

3.3.3 启动 Nebula Graph 服务

非容器部署

对于非容器部署的 Nebula Graph，执行如下命令启动服务：

```
$ sudo /usr/local/nebula/scripts/nebula.service start all
[INFO] Starting nebula-metad...
[INFO] Done
[INFO] Starting nebula-graphd...
[INFO] Done
[INFO] Starting nebula-storaged...
[INFO] Done
```

或者：

```
$ systemctl start nebula
```

如果需要设置开机自动启动，命令如下：

```
$ systemctl enable nebula
```

容器部署

对于使用 Docker Compose 部署的 Nebula Graph，在 `nebula-docker-compose/` 目录内执行如下命令启动服务：

```
[nebula-docker-compose]$ docker-compose up -d
Building with native build. Learn about native build in Compose here: https://docs.docker.com/go/compose-native-build/
Creating network "nebula-docker-compose_nebula-net" with the default driver
Creating nebula-docker-compose_metad0_1 ... done
Creating nebula-docker-compose_metad2_1 ... done
Creating nebula-docker-compose_metad1_1 ... done
Creating nebula-docker-compose_storaged2_1 ... done
Creating nebula-docker-compose_graphd1_1   ... done
```


2022 vesoft Inc.

```
Creating nebula-docker-compose_storaged1_1 ... done
Creating nebula-docker-compose_storaged0_1 ... done
Creating nebula-docker-compose_graphd2_1   ... done
Creating nebula-docker-compose_graphd_1    ... done
```

3.3.4 停止 Nebula Graph 服务

Danger

请勿使用 `kill -9` 命令强制终止进程，否则可能较小概率出现数据丢失。

非容器部署

执行如下命令停止 Nebula Graph 服务：

```
$ sudo /usr/local/nebula/scripts/nebula.service stop all
[INFO] Stopping nebula-metad...
[INFO] Done
[INFO] Stopping nebula-graphd...
[INFO] Done
[INFO] Stopping nebula-storaged...
[INFO] Done
```

或者：

```
$ systemctl stop nebula
```

容器部署

在 `nebula-docker-compose/` 目录内执行如下命令停止 Nebula Graph 服务：

```
[nebula-docker-compose]$ docker-compose down
Stopping nebula-docker-compose_graphd_1    ... done
Stopping nebula-docker-compose_graphd2_1   ... done
Stopping nebula-docker-compose_storaged0_1 ... done
Stopping nebula-docker-compose_storaged1_1 ... done
Stopping nebula-docker-compose_graphd1_1   ... done
Stopping nebula-docker-compose_storaged2_1 ... done
Stopping nebula-docker-compose_metad1_1    ... done
Stopping nebula-docker-compose_metad2_1    ... done
Stopping nebula-docker-compose_metad0_1    ... done
Removing nebula-docker-compose_graphd_1    ... done
Removing nebula-docker-compose_graphd2_1   ... done
Removing nebula-docker-compose_storaged0_1 ... done
Removing nebula-docker-compose_storaged1_1 ... done
Removing nebula-docker-compose_graphd1_1   ... done
Removing nebula-docker-compose_storaged2_1 ... done
Removing nebula-docker-compose_metad1_1    ... done
Removing nebula-docker-compose_metad2_1    ... done
Removing nebula-docker-compose_metad0_1    ... done
Removing network nebula-docker-compose_nebula-net
```

Note

命令 `docker-compose down -v` 将会删除所有本地 Nebula Graph 的数据。如果使用的是 developing 或 nightly 版本，并且有一些兼容性问题，请尝试这个命令。

3.3.5 查看 Nebula Graph 服务

非容器部署

执行如下命令查看 Nebula Graph 服务状态：

2022 vesoft Inc.

```
$ sudo /usr/local/nebula/scripts/nebula.service status all
```

- 如果返回如下结果，表示 Nebula Graph 服务正常运行。

```
[INFO] nebula-metad(33fd35e): Running as 29020, Listening on 9559
[INFO] nebula-graphd(33fd35e): Running as 29095, Listening on 9669
[WARN] nebula-storaged after v3.0.0 will not start service until it is added to cluster.
[WARN] See Manage Storage hosts:ADD HOSTS in https://docs.nebula-graph.io/
[INFO] nebula-storaged(33fd35e): Running as 29147, Listening on 9779
```

> **Note**
>
> 正常启动 Nebula Graph 后，`nebula-storaged` 进程的端口显示红色。这是因为 `nebula-storaged` 在启动流程中会等待 `nebula-metad` 添加当前 Storage 服务，当前 Storage 服务收到 Ready 信号后才会正式启动服务。从 3.0.0 版本开始，在配置文件中添加的 Storage 节点无法直接读写，配置文件的作用仅仅是将 Storage 节点注册至 Meta 服务中。必须使用 `ADD HOSTS` 命令后，才能正常读写 Storage 节点。更多信息，参见管理 Storage 主机。

- 如果返回类似如下结果，表示 Nebula Graph 服务异常，可以根据异常服务信息进一步排查，或者在 Nebula Graph 社区寻求帮助。

```
[INFO] nebula-metad: Running as 25600, Listening on 9559
[INFO] nebula-graphd: Exited
[INFO] nebula-storaged: Running as 25646, Listening on 9779
```

也可以使用 `systemctl` 命令查看 Nebula Graph 服务状态:

```
$ systemctl status nebula
● nebula.service
   Loaded: loaded (/usr/lib/systemd/system/nebula.service; disabled; vendor preset: disabled)
   Active: active (exited) since 一 2022-03-28 04:13:24 UTC; 1h 47min ago
  Process: 21772 ExecStart=/usr/local/ent-nightly/scripts/nebula.service start all (code=exited, status=0/SUCCESS)
 Main PID: 21772 (code=exited, status=0/SUCCESS)
    Tasks: 325
   Memory: 424.5M
   CGroup: /system.slice/nebula.service
           ├─21789 /usr/local/ent-nightly/bin/nebula-metad --flagfile /usr/local/ent-nightly/etc/nebula-metad.conf
           ├─21827 /usr/local/ent-nightly/bin/nebula-graphd --flagfile /usr/local/ent-nightly/etc/nebula-graphd.conf
           └─21900 /usr/local/ent-nightly/bin/nebula-storaged --flagfile /usr/local/ent-nightly/etc/nebula-storaged.conf

3月 28 04:13:24 xxxxxx systemd[1]: Started nebula.service.
...
```

Nebula Graph 服务由 Meta 服务、Graph 服务和 Storage 服务共同提供，这三种服务的配置文件都保存在安装目录的 `etc` 目录内，默认路径为 `/usr/local/nebula/etc/`，用户可以检查相应的配置文件排查问题。

容器部署

在 `nebula-docker-compose` 目录内执行如下命令查看 Nebula Graph 服务状态:

```
[nebula-docker-compose]$ docker-compose ps
            Name                           Command                State                                        Ports
-------------------------------------------------------------------------------------------------------------------------------------------------------
nebula-docker-compose_graphd1_1     /usr/local/nebula/bin/nebu ...   Up (healthy)   0.0.0.0:49223->19669/tcp, 0.0.0.0:49222->19670/tcp, 0.0.0.0:49224->9669/tcp
nebula-docker-compose_graphd2_1     /usr/local/nebula/bin/nebu ...   Up (healthy)   0.0.0.0:49229->19669/tcp, 0.0.0.0:49228->19670/tcp, 0.0.0.0:49230->9669/tcp
nebula-docker-compose_graphd_1      /usr/local/nebula/bin/nebu ...   Up (healthy)   0.0.0.0:49221->19669/tcp, 0.0.0.0:49220->19670/tcp, 0.0.0.0:9669->9669/tcp
nebula-docker-compose_metad0_1      ./bin/nebula-metad --flagf ...   Up (healthy)   0.0.0.0:49212->19559/tcp, 0.0.0.0:49211->19560/tcp, 0.0.0.0:49213->9559/tcp,
                                                                                    9560/tcp
nebula-docker-compose_metad1_1      ./bin/nebula-metad --flagf ...   Up (healthy)   0.0.0.0:49209->19559/tcp, 0.0.0.0:49208->19560/tcp, 0.0.0.0:49210->9559/tcp,
                                                                                    9560/tcp
nebula-docker-compose_metad2_1      ./bin/nebula-metad --flagf ...   Up (healthy)   0.0.0.0:49206->19559/tcp, 0.0.0.0:49205->19560/tcp, 0.0.0.0:49207->9559/tcp,
                                                                                    9560/tcp
nebula-docker-compose_storaged0_1   ./bin/nebula-storaged --fl ...   Up (healthy)   0.0.0.0:49218->19779/tcp, 0.0.0.0:49217->19780/tcp, 9777/tcp, 9778/tcp,
                                                                                    0.0.0.0:49219->9779/tcp, 9780/tcp
nebula-docker-compose_storaged1_1   ./bin/nebula-storaged --fl ...   Up (healthy)   0.0.0.0:49215->19779/tcp, 0.0.0.0:49214->19780/tcp, 9777/tcp, 9778/tcp,
                                                                                     0.0.0.0:49216->9779/tcp, 9780/tcp
nebula-docker-compose_storaged2_1   ./bin/nebula-storaged --fl ...   Up (healthy)   0.0.0.0:49226->19779/tcp, 0.0.0.0:49225->19780/tcp, 9777/tcp, 9778/tcp,
                                                                                    0.0.0.0:49227->9779/tcp, 9780/tcp
```

如果服务有异常，用户可以先确认异常的容器名称（例如 `nebula-docker-compose_graphd2_1`），

然后执行 `docker ps` 查看对应的 `CONTAINER ID`（示例为 `2a6c56c405f5`）。

```
[nebula-docker-compose]$ docker ps
CONTAINER ID   IMAGE                             COMMAND                  CREATED         STATUS
PORTS                                                                                 NAMES
```


2022 vesoft Inc.

```
2a6c56c405f5   vesoft/nebula-graphd:nightly     "/usr/local/nebula/b..."   36 minutes ago   Up 36 minutes (healthy)   0.0.0.0:49230->9669/tcp, 0.0.0.0:49229->19669/tcp, 0.0.0.0:49228->19670/
tcp                             nebula-docker-compose_graphd2_1
7042e0a8e83d   vesoft/nebula-storaged:nightly   "./bin/nebula-storag..."   36 minutes ago   Up 36 minutes (healthy)   9777-9778/tcp, 9780/tcp, 0.0.0.0:49227->9779/tcp, 0.0.0.0:49226->19779/
tcp, 0.0.0.0:49225->19780/tcp   nebula-docker-compose_storaged2_1
18e3ea63ad65   vesoft/nebula-storaged:nightly   "./bin/nebula-storag..."   36 minutes ago   Up 36 minutes (healthy)   9777-9778/tcp, 9780/tcp, 0.0.0.0:49219->9779/tcp, 0.0.0.0:49218->19779/
tcp, 0.0.0.0:49217->19780/tcp   nebula-docker-compose_storaged0_1
4dcabfe8677a   vesoft/nebula-graphd:nightly     "/usr/local/nebula/b..."   36 minutes ago   Up 36 minutes (healthy)   0.0.0.0:49224->9669/tcp, 0.0.0.0:49223->19669/tcp, 0.0.0.0:49222->19670/
tcp                             nebula-docker-compose_graphd1_1
a74054c6ae25   vesoft/nebula-graphd:nightly     "/usr/local/nebula/b..."   36 minutes ago   Up 36 minutes (healthy)   0.0.0.0:9669->9669/tcp, 0.0.0.0:49221->19669/tcp, 0.0.0.0:49220->19670/
tcp                             nebula-docker-compose_graphd_1
880025a3858c   vesoft/nebula-storaged:nightly   "./bin/nebula-storag..."   36 minutes ago   Up 36 minutes (healthy)   9777-9778/tcp, 9780/tcp, 0.0.0.0:49216->9779/tcp, 0.0.0.0:49215->19779/
tcp, 0.0.0.0:49214->19780/tcp   nebula-docker-compose_storaged1_1
45736a32a23a   vesoft/nebula-metad:nightly      "./bin/nebula-metad ..."   36 minutes ago   Up 36 minutes (healthy)   9560/tcp, 0.0.0.0:49213->9559/tcp, 0.0.0.0:49212->19559/tcp, 0.
0.0.0:49211->19560/tcp          nebula-docker-compose_metad0_1
3b2c90eb073e   vesoft/nebula-metad:nightly      "./bin/nebula-metad ..."   36 minutes ago   Up 36 minutes (healthy)   9560/tcp, 0.0.0.0:49207->9559/tcp, 0.0.0.0:49206->19559/tcp, 0.
0.0.0:49205->19560/tcp          nebula-docker-compose_metad2_1
7bb31b7a5b3f   vesoft/nebula-metad:nightly      "./bin/nebula-metad ..."   36 minutes ago   Up 36 minutes (healthy)   9560/tcp, 0.0.0.0:49210->9559/tcp, 0.0.0.0:49209->19559/tcp, 0.
0.0.0:49208->19560/tcp          nebula-docker-compose_metad1_1
```

最后登录容器排查问题

```
[nebula-docker-compose]$ docker exec -it 2a6c56c405f5 bash
[root@2a6c56c405f5 nebula]#
```

3.3.6 下一步

连接 Nebula Graph

最后更新: June 20, 2023

2022 vesoft Inc.

3.4 步骤 3：连接 Nebula Graph

本文介绍如何使用原生命令行客户端 Nebula Console 连接 Nebula Graph。

Caution

首次连接到 Nebula Graph 后，必须先注册 Storage 服务，才能正常查询数据。

Nebula Graph 支持多种类型的客户端，包括命令行客户端、可视化界面客户端和流行编程语言客户端。详情参见客户端列表。

3.4.1 前提条件

- Nebula Graph 服务已启动。
- 运行 Nebula Console 的机器和运行 Nebula Graph 的服务器网络互通。
- Nebula Console 的版本兼容 Nebula Graph 的版本。

Note

版本相同的 Nebula Console 和 Nebula Graph 兼容程度最高，版本不同的 Nebula Console 连接 Nebula Graph 时，可能会有兼容问题，或者无法连接并报错 `incompatible version between client and server`。

3.4.2 操作步骤

1. 在 Nebula Console 下载页面，确认需要的版本，单击 **Assets**。

   Note

   建议选择最新版本。

2. 在 **Assets** 区域找到机器运行所需的二进制文件，下载文件到机器上。
3. （可选）为方便使用，重命名文件为 `nebula-console`。

   Note

   在 Windows 系统中，请重命名为 `nebula-console.exe`。

4. 在运行 Nebula Console 的机器上执行如下命令，为用户授予 nebula-console 文件的执行权限。

   Note

   Windows 系统请跳过此步骤。

```
$ chmod 111 nebula-console
```

5. 在命令行界面中，切换工作目录至 nebula-console 文件所在目录。

2022 vesoft Inc.

6. 执行如下命令连接 Nebula Graph。

- Linux 或 macOS

```
$ ./nebula-console -addr <ip> -port <port> -u <username> -p <password>
  [-t 120] [-e "nGQL_statement" | -f filename.nGQL]
```

- Windows

```
> nebula-console.exe -addr <ip> -port <port> -u <username> -p <password>
  [-t 120] [-e "nGQL_statement" | -f filename.nGQL]
```

参数说明如下。

参数	说明
`-h/-help`	显示帮助菜单。
`-addr/-address`	设置要连接的 Graph 服务的 IP 地址。默认地址为 127.0.0.1。
`-P/-port`	设置要连接的 Graph 服务的端口。默认端口为 9669。
`-u/-user`	设置 Nebula Graph 账号的用户名。未启用身份认证时，可以使用任意已存在的用户名（默认为 `root`）。
`-p/-password`	设置用户名对应的密码。未启用身份认证时，密码可以填写任意字符。
`-t/-timeout`	设置整数类型的连接超时时间。单位为秒，默认值为 120。
`-e/-eval`	设置字符串类型的 nGQL 语句。连接成功后会执行一次该语句并返回结果，然后自动断开连接。
`-f/-file`	设置存储 nGQL 语句的文件的路径。连接成功后会执行该文件内的 nGQL 语句并返回结果，执行完毕后自动断开连接。
`-enable_ssl`	连接 Nebula Graph 时使用 SSL 加密。
`-ssl_root_ca_path`	指定 CA 证书的存储路径。
`-ssl_cert_path`	指定 CRT 证书的存储路径。
`-ssl_private_key_path`	指定私钥文件的存储路径。

更多参数参见项目仓库。

最后更新: June 20, 2023

2022 vesoft Inc.

3.5 注册 Storage 服务

首次连接到 Nebula Graph 后，需要先添加 Storage 主机，并确认主机都处于在线状态。

Compatibility

- 从 Nebula Graph 3.0.0 版本开始，必须先使用 `ADD HOSTS` 添加主机，才能正常通过 Storage 服务读写数据。
- 在此前的版本中，无需执行该操作。

3.5.1 前提条件

已连接 Nebula Graph 服务。

3.5.2 操作步骤

1. 添加 Storage 主机。

   执行如下命令添加主机：

   ```
   ADD HOSTS <ip>:<port> [,<ip>:<port> ...];
   ```

   示例：

   ```
   nebula> ADD HOSTS 192.168.10.100:9779, 192.168.10.101:9779, 192.168.10.102:9779;
   ```

   Caution

   请确保添加的主机 IP 和配置文件 `nebula-storaged.conf` 中 `local_ip` 配置的 IP 一致，否则会导致添加 Storage 主机失败。关于配置文件的详情，参见配置管理。

2. 检查主机状态，确认全部在线。

   ```
   nebula> SHOW HOSTS;
   +-----------------+------+----------+----------+--------------+----------------------+  +------------------------+---------+
   | Host            | Port | HTTP port | Status   | Leader count | Leader distribution  |  Partition distribution | Version |
   +-----------------+------+----------+----------+--------------+----------------------+  +------------------------+---------+
   | "192.168.10.100" | 9779 | 19669     | "ONLINE" | 0            | "No valid partition" | "No   valid partition"   | "3.1.0" |
   | "192.168.10.101" | 9779 | 19669     | "ONLINE" | 0            | "No valid partition" | "No   valid partition"   | "3.1.0" |
   | "192.168.10.102" | 9779 | 19669     | "ONLINE" | 0            | "No valid partition" | "No   valid partition"   | "3.1.0" |
   +-----------------+------+----------+----------+--------------+----------------------+  +------------------------+---------+
   ```

   在返回结果的 **Status** 列，可以看到所有 Storage 主机都在线。

最后更新: June 20, 2023

2022 vesoft Inc.

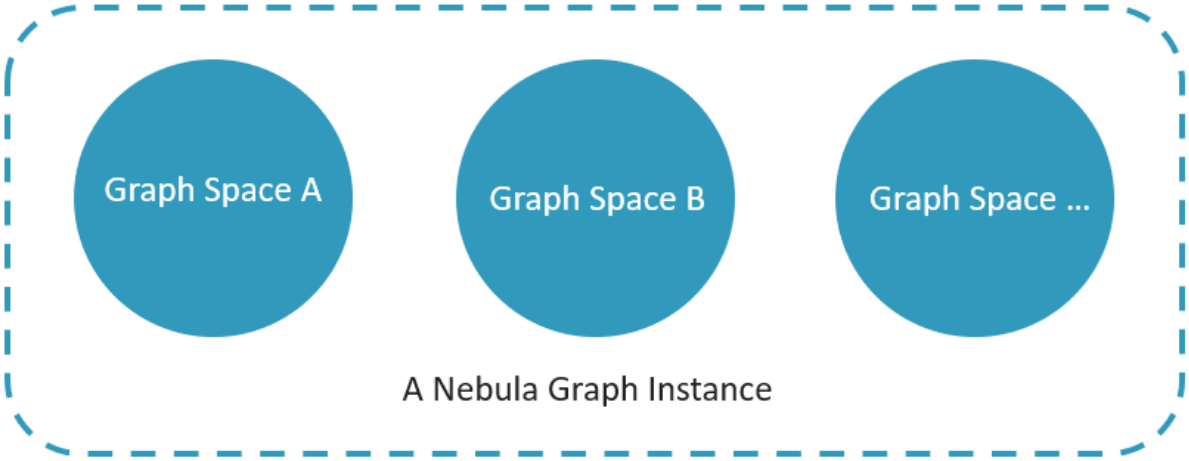
3.6 步骤 4：使用常用 nGQL（CRUD 命令）

本文介绍 Nebula Graph 查询语言的基础语法，包括用于 Schema 创建和常用增删改查操作的语句。

如需了解更多语句的用法，参见 nGQL 指南。

3.6.1 图空间和 Schema

一个 Nebula Graph 实例由一个或多个图空间组成。每个图空间都是物理隔离的，用户可以在同一个实例中使用不同的图空间存储不同的数据集。



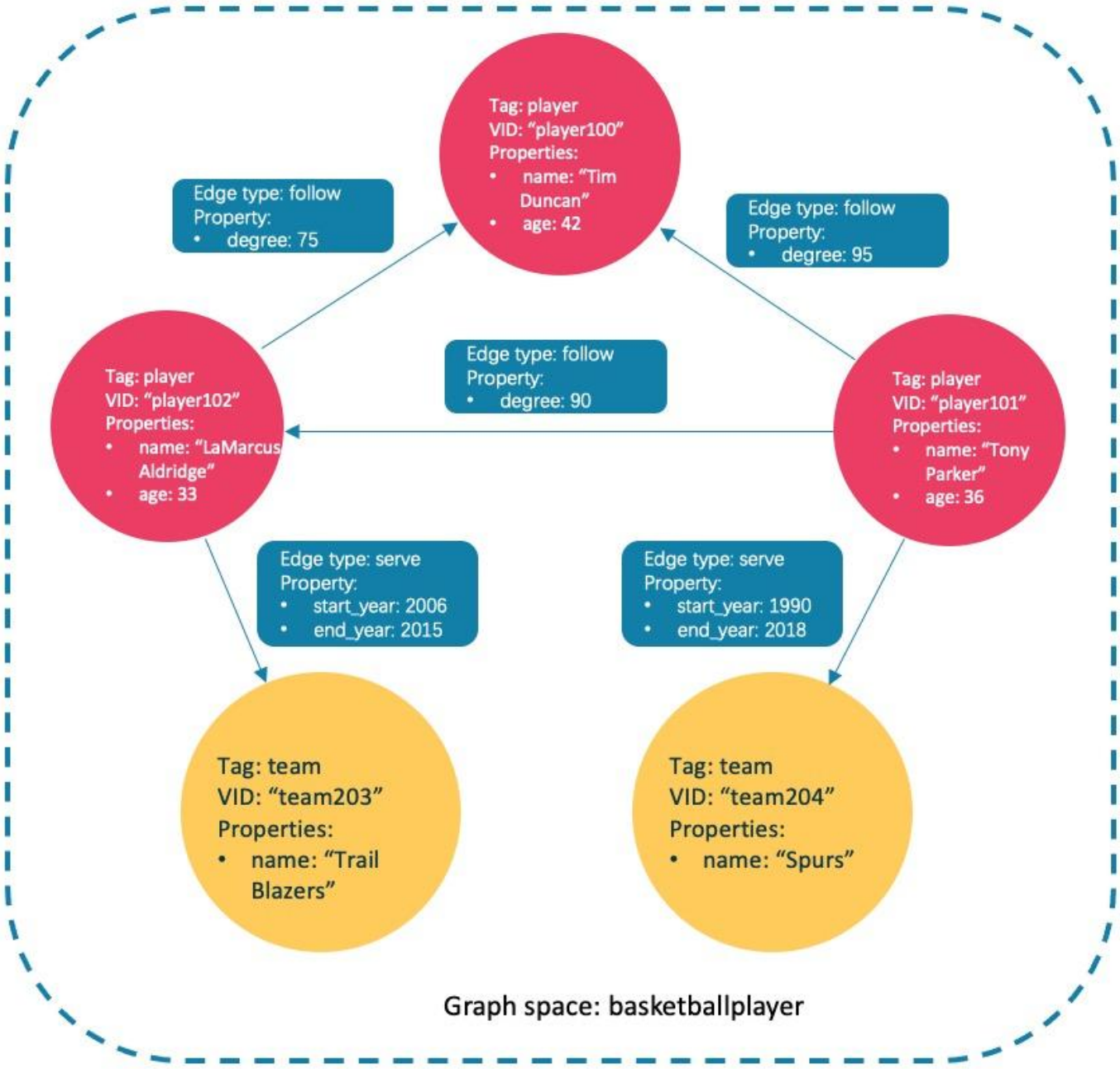
为了在图空间中插入数据，需要为图数据库定义一个 Schema。Nebula Graph 的 Schema 是由如下几部分组成。

组成部分	说明
点（Vertex）	表示现实世界中的实体。一个点可以有 0 到多个标签。
标签（Tag）	点的类型，定义了一组描述点类型的属性。
边（Edge）	表示两个点之间有方向的关系。
边类型（Edge type）	边的类型，定义了一组描述边的类型的属性。

更多信息，请参见数据结构。

本文将使用下图的数据集演示基础操作的语法。

2022 vesoft Inc.



2022 vesoft Inc.

异步实现创建和修改

Caution

Nebula Graph 中执行如下创建和修改操作，是异步实现的。要在下一个心跳周期之后才能生效，否则访问会报错。为确保数据同步，后续操作能顺利进行，请等待 2 个心跳周期（20 秒）。

- `CREATE SPACE`
- `CREATE TAG`
- `CREATE EDGE`
- `ALTER TAG`
- `ALTER EDGE`
- `CREATE TAG INDEX`
- `CREATE EDGE INDEX`

Note

默认心跳周期是 10 秒。修改心跳周期参数 `heartbeat_interval_secs`，请参见配置简介。

3.6.2 创建和选择图空间

nGQL 语法

- 创建图空间

```
CREATE SPACE [IF NOT EXISTS] <graph_space_name> (
[partition_num = <partition_number>,]
[replica_factor = <replica_number>,]
vid_type = {FIXED_STRING(<N>) | INT64}
)

[COMMENT = '<comment>'];
```

参数详情请参见 CREATE SPACE。

- 列出创建成功的图空间

```
nebula> SHOW SPACES;
```

- 选择数据库

```
USE <graph_space_name>;
```

示例

1. 执行如下语句创建名为 `basketballplayer` 的图空间。

```
nebula> CREATE SPACE basketballplayer(partition_num=15, replica_factor=1, vid_type=fixed_string(30));
```

2. 执行命令 `SHOW HOSTS` 检查分片的分布情况，确保平衡分布。

```
nebula> SHOW HOSTS;
+-------------+-----------+-----------+-----------+--------------+---------------------------------+------------------------+---------+
| Host        | Port      | HTTP port | Status    | Leader count | Leader distribution             | Partition distribution | Version |
+-------------+-----------+-----------+-----------+--------------+---------------------------------+------------------------+---------+
| "storaged0" | 9779      | 19669     | "ONLINE"  | 5            | "basketballplayer:5"            | "basketballplayer:5"   | "3.1.0" |
| "storaged1" | 9779      | 19669     | "ONLINE"  | 5            | "basketballplayer:5"            | "basketballplayer:5"   | "3.1.0" |
| "storaged2" | 9779      | 19669     | "ONLINE"  | 5            | "basketballplayer:5"            | "basketballplayer:5"   | "3.1.0" |
+-------------+-----------+-----------+-----------+--------------+---------------------------------+------------------------+---------+
```


2022 vesoft Inc.

如果 **Leader distribution** 分布不均匀，请执行命令 `BALANCE LEADER` 重新分配。更多信息，请参见 Storage 负载均衡。

3. 选择图空间 `basketballplayer` 。

```
nebula[(none)]> USE basketballplayer;
```

用户可以执行命令 `SHOW SPACES` 查看创建的图空间。

```
nebula> SHOW SPACES;
+--------------------+
| Name               |
+--------------------+
| "basketballplayer" |
+--------------------+
```

3.6.3 创建 Tag 和 Edge type

nGQL 语法

```
CREATE {TAG | EDGE} [IF NOT EXISTS] {<tag_name> | <edge_type_name>}
    (
      <prop_name> <data_type> [NULL | NOT NULL] [DEFAULT <default_value>] [COMMENT '<comment>']
      [{, <prop_name> <data_type> [NULL | NOT NULL] [DEFAULT <default_value>] [COMMENT '<comment>']} ...]
    )
    [TTL_DURATION = <ttl_duration>]
    [TTL_COL = <prop_name>]
    [COMMENT = '<comment>'];
```

参数详情请参见 CREATE TAG 和 CREATE EDGE。

示例

创建 Tag: `player` 和 `team` ，以及 Edge type: `follow` 和 `serve` 。说明如下表。

名称	类型	属性
player	Tag	name (string), age (int)
team	Tag	name (string)
follow	Edge type	degree (int)
serve	Edge type	start_year (int), end_year (int)

```
nebula> CREATE TAG player(name string, age int);

nebula> CREATE TAG team(name string);

nebula> CREATE EDGE follow(degree int);

nebula> CREATE EDGE serve(start_year int, end_year int);
```

3.6.4 插入点和边

用户可以使用 `INSERT` 语句，基于现有的 Tag 插入点，或者基于现有的 Edge type 插入边。

nGQL 语法

- 插入点

```
INSERT VERTEX [IF NOT EXISTS] [tag_props, [tag_props] ...]
VALUES <vid>: ([prop_value_list])

tag_props:
  tag_name ([prop_name_list])

prop_name_list:
   [prop_name [, prop_name] ...]
```


2022 vesoft Inc.

```
prop_value_list:
   [prop_value [, prop_value] ...]
```

`vid` 是 Vertex ID 的缩写，`vid` 在一个图空间中是唯一的。参数详情请参见 INSERT VERTEX。

- 插入边

```
INSERT EDGE [IF NOT EXISTS] <edge_type> ( <prop_name_list> ) VALUES
<src_vid> -> <dst_vid>[@<rank>] : ( <prop_value_list> )
[, <src_vid> -> <dst_vid>[@<rank>] : ( <prop_value_list> ), ...];

<prop_name_list> ::=
[ <prop_name> [, <prop_name> ] ...]

<prop_value_list> ::=
[ <prop_value> [, <prop_value> ] ...]
```

参数详情请参见 INSERT EDGE。

示例

- 插入代表球员和球队的点。

```
nebula> INSERT VERTEX player(name, age) VALUES "player100":("Tim Duncan", 42);

nebula> INSERT VERTEX player(name, age) VALUES "player101":("Tony Parker", 36);

nebula> INSERT VERTEX player(name, age) VALUES "player102":("LaMarcus Aldridge", 33);

nebula> INSERT VERTEX team(name) VALUES "team203":("Trail Blazers"), "team204":("Spurs");
```

- 插入代表球员和球队之间关系的边。

```
nebula> INSERT EDGE follow(degree) VALUES "player101" -> "player100":(95);

nebula> INSERT EDGE follow(degree) VALUES "player101" -> "player102":(90);

nebula> INSERT EDGE follow(degree) VALUES "player102" -> "player100":(75);

nebula> INSERT EDGE serve(start_year, end_year) VALUES "player101" -> "team204":(1999, 2018),"player102" -> "team203":(2006,  2015);
```

3.6.5 查询数据

- GO 语句可以根据指定的条件遍历数据库。`GO` 语句从一个或多个点开始，沿着一条或多条边遍历，返回 `YIELD` 子句中指定的信息。
- FETCH 语句可以获得点或边的属性。
- LOOKUP 语句是基于索引的，和 `WHERE` 子句一起使用，查找符合特定条件的数据。
- MATCH 语句是查询图数据最常用的，可以灵活的描述各种图模式，但是它依赖索引去匹配 Nebula Graph 中的数据模型，性能也还需要调优。

nGQL 语法

- `GO`

```
GO [[<M> TO] <N> STEPS ] FROM <vertex_list>
OVER <edge_type_list> [{REVERSELY | BIDIRECT}]
[ WHERE <conditions> ]
YIELD [DISTINCT] <return_list>
[{ SAMPLE <sample_list> | <limit_by_list_clause> }]
[| GROUP BY {<col_name> | expression> | <position>} YIELD <col_name>]
```


2022 vesoft Inc.

```
[| ORDER BY <expression> [{ASC | DESC}]]
[| LIMIT [<offset>,] <number_rows>];
```

- `FETCH`
- 查询 Tag 属性

```
FETCH PROP ON {<tag_name>[, tag_name ...] | *}
<vid> [, vid ...]
YIELD <return_list> [AS <alias>];
```

- 查询边属性

```
FETCH PROP ON <edge_type> <src_vid> -> <dst_vid>[@<rank>] [, <src_vid> -> <dst_vid> ...]
YIELD <output>;
```

- `LOOKUP`

```
LOOKUP ON {<vertex_tag> | <edge_type>}
[WHERE <expression> [AND <expression> ...]]
YIELD <return_list> [AS <alias>];

<return_list>
    <prop_name> [AS <col_alias>] [, <prop_name> [AS <prop_alias>] ...];
```

- `MATCH`

```
MATCH <pattern> [<clause_1>]  RETURN <output>  [<clause_2>];
```

`GO` 语句示例

- 从 VID 为 `player101` 的球员开始，沿着边 `follow` 找到连接的球员。

```
nebula> GO FROM "player101" OVER follow YIELD id($$);
+-------------+
| id($$)      |
+-------------+
| "player100" |
| "player102" |
```


2022 vesoft Inc.

```
| "player125" |
+-------------+
```

- 从 VID 为 `player101` 的球员开始，沿着边 `follow` 查找年龄大于或等于 35 岁的球员，并返回他们的姓名和年龄，同时重命名对应的列。

```
nebula> GO FROM "player101" OVER follow WHERE properties($$).age >= 35 \
        YIELD properties($$).name AS Teammate, properties($$).age AS Age;
+-----------------+-----+
| Teammate        | Age |
+-----------------+-----+
| "Tim Duncan"    | 42  |
| "Manu Ginobili" | 41  |
+-----------------+-----+
```

子句/符号	说明
`YIELD`	指定该查询需要返回的值或结果。
`$$`	表示边的终点。
`\`	表示换行继续输入。

- 从 VID 为 `player101` 的球员开始，沿着边 `follow` 查找连接的球员，然后检索这些球员的球队。为了合并这两个查询请求，可以使用管道符或临时变量。

- 使用管道符

```
nebula> GO FROM "player101" OVER follow YIELD dst(edge) AS id | \
        GO FROM $-.id OVER serve YIELD properties($$).name AS Team, \
        properties($^).name AS Player;
+-----------------+--------------------+
| Team            | Player             |
+-----------------+--------------------+
| "Spurs"         | "Tim Duncan"       |
| "Trail Blazers" | "LaMarcus Aldridge" |
| "Spurs"         | "LaMarcus Aldridge" |
| "Spurs"         | "Manu Ginobili"    |
+-----------------+--------------------+
```

子句/符号	说明
`$^`	表示边的起点。
`|`	组合多个查询的管道符，将前一个查询的结果集用于后一个查询。
`$-`	表示管道符前面的查询输出的结果集。

- 使用临时变量

> **Note**
>
> 当复合语句作为一个整体提交给服务器时，其中的临时变量会在语句结束时被释放。

```
nebula> $var = GO FROM "player101" OVER follow YIELD dst(edge) AS id; \
        GO FROM $var.id OVER serve YIELD properties($$).name AS Team, \
        properties($^).name AS Player;
+-----------------+--------------------+
| Team            | Player             |
+-----------------+--------------------+
| "Spurs"         | "Tim Duncan"       |
| "Trail Blazers" | "LaMarcus Aldridge" |
| "Spurs"         | "LaMarcus Aldridge" |
| "Spurs"         | "Manu Ginobili"    |
+-----------------+--------------------+
```

`FETCH` 语句示例

查询 VID 为 `player100` 的球员的属性。

2022 vesoft Inc.

```
nebula> FETCH PROP ON player "player100" YIELD properties(vertex);
+--------------------------------+
| properties(VERTEX)             |
+--------------------------------+
| {age: 42, name: "Tim Duncan"} |
+--------------------------------+
```

> **Note**
>
> LOOKUP 和 MATCH 的示例在下文的索引 部分查看。

3.6.6 修改点和边

用户可以使用 UPDATE 语句或 UPSERT 语句修改现有数据。

UPSERT 是 UPDATE 和 INSERT 的结合体。当使用 UPSERT 更新一个点或边，如果它不存在，数据库会自动插入一个新的点或边。

> **Note**
>
> 每个 partition 内部， UPSERT 操作是一个串行操作，所以执行速度比执行 INSERT 或 UPDATE 慢很多。其仅在多个 partition 之间有并发。

nGQL 语法

- UPDATE 点

```
UPDATE VERTEX <vid> SET <properties to be updated>
[WHEN <condition>] [YIELD <columns>];
```

- UPDATE 边

```
UPDATE EDGE <source vid> -> <destination vid> [@rank] OF <edge_type>
SET <properties to be updated> [WHEN <condition>] [YIELD <columns to be output>];
```

- UPSERT 点或边

```
UPSERT {VERTEX <vid> | EDGE <edge_type>} SET <update_columns>
[WHEN <condition>] [YIELD <columns>];
```

示例

- 用 UPDATE 修改 VID 为 player100 的球员的 name 属性，然后用 FETCH 语句检查结果。

```
nebula> UPDATE VERTEX "player100" SET player.name = "Tim";

nebula> FETCH PROP ON player "player100" YIELD properties(vertex);
+-------------------------+
| properties(VERTEX)      |
+-------------------------+
```


2022 vesoft Inc.

```
| {age: 42, name: "Tim"} |
+------------------------+
```

- 用 `UPDATE` 修改某条边的 `degree` 属性，然后用 `FETCH` 检查结果。

```
nebula> UPDATE EDGE "player101" -> "player100" OF follow SET degree = 96;

nebula> FETCH PROP ON follow "player101" -> "player100" YIELD properties(edge);
+------------------+
| properties(EDGE) |
+------------------+
| {degree: 96}     |
+------------------+
```

- 用 `INSERT` 插入一个 VID 为 `player111` 的点，然后用 `UPSERT` 更新它。

```
nebula> INSERT VERTEX player(name,age) VALUES "player111":("David West", 38);

nebula> UPSERT VERTEX "player111" SET player.name = "David", player.age = $^.player.age + 11 \
        WHEN $^.player.name == "David West" AND $^.player.age > 20 \
        YIELD $^.player.name AS Name, $^.player.age AS Age;
+---------+-----+
| Name    | Age |
+---------+-----+
| "David" | 49  |
+---------+-----+
```

3.6.7 删除点和边

nGQL 语法

- 删除点

```
DELETE VERTEX <vid1>[, <vid2>...]
```

- 删除边

```
DELETE EDGE <edge_type> <src_vid> -> <dst_vid>[@<rank>]
[, <src_vid> -> <dst_vid>...]
```

示例

- 删除点

```
nebula> DELETE VERTEX "player111", "team203";
```

- 删除边

```
nebula> DELETE EDGE follow "player101" -> "team204";
```

3.6.8 索引

用户可以通过 CREATE INDEX 语句为 Tag 和 Edge type 增加索引。

使用索引必读

`MATCH` 和 `LOOKUP` 语句的执行都依赖索引，但是索引会导致写性能大幅降低（降低 90%甚至更多）。请不要随意在生产环境中使用索引，除非很清楚使用索引对业务的影响。

必须为“已写入但未构建索引”的数据重建索引，否则无法在 `MATCH` 和 `LOOKUP` 语句中返回这些数据。参见重建索引。

2022 vesoft Inc.

nGQL 语法

- 创建索引

```
CREATE {TAG | EDGE} INDEX [IF NOT EXISTS] <index_name>
ON {<tag_name> | <edge_name>} ([<prop_name_list>]) [COMMENT = '<comment>'];
```

- 重建索引

```
REBUILD {TAG | EDGE} INDEX <index_name>;
```

> **Note**
>
> 为没有指定长度的变量属性创建索引时，需要指定索引长度。在 utf-8 编码中，一个中文字符占 3 字节，请根据变量属性长度设置合适的索引长度。例如 10 个中文字符，索引长度需要为 30。详情请参见创建索引。

基于索引的 **LOOKUP** 和 **MATCH** 示例

确保 LOOKUP 或 MATCH 有一个索引可用。如果没有，请先创建索引。

找到 Tag 为 player 的点的信息，它的 name 属性值为 Tony Parker 。

```
// 为 name 属性创建索引 player_index_1。
nebula> CREATE TAG INDEX IF NOT EXISTS player_index_1 ON player(name(20));

// 重建索引确保能对已存在数据生效。
nebula> REBUILD TAG INDEX player_index_1
+------------+
| New Job Id |
+------------+
| 31         |
+------------+

// 使用 LOOKUP 语句检索点的属性。
nebula> LOOKUP ON player WHERE player.name == "Tony Parker" \
        YIELD properties(vertex).name AS name, properties(vertex).age AS age;
+---------------+-----+
| name          | age |
+---------------+-----+
| "Tony Parker" | 36  |
+---------------+-----+

// 使用 MATCH 语句检索点的属性。
nebula> MATCH (v:player{name:"Tony Parker"}) RETURN v;
+-------------------------------------------------------+
| v                                                     |
+-------------------------------------------------------+
| ("player101" :player{age: 36, name: "Tony Parker"}) |
+-------------------------------------------------------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

3.7 nGQL 命令汇总

3.7.1 函数

2022 vesoft Inc.

- 数学函数

2022 vesoft Inc.

函数	说明
double abs(double x)	返回 x 的绝对值。
double floor(double x)	返回小于或等于 x 的最大整数。
double ceil(double x)	返回大于或等于 x 的最小整数。
double round(double x)	返回离 x 最近的整数值，如果 x 恰好在中间，则返回离 0 较远的整数。
double sqrt(double x)	返回 x 的平方根。
double cbrt(double x)	返回 x 的立方根。
double hypot(double x, double y)	返回直角三角形（直角边长为 x 和 y）的斜边长。
double pow(double x, double y)	返回 x^y 的值。
double exp(double x)	返回 e^x 的值。
double exp2(double x)	返回 2^x 的值。
double log(double x)	返回以自然数 e 为底 x 的对数。
double log2(double x)	返回以 2 为底 x 的对数。
double log10(double x)	返回以 10 为底 x 的对数。
double sin(double x)	返回 x 的正弦值。
double asin(double x)	返回 x 的反正弦值。
double cos(double x)	返回 x 的余弦值。
double acos(double x)	返回 x 的反余弦值。
double tan(double x)	返回 x 的正切值。
double atan(double x)	返回 x 的反正切值。
double rand()	返回 [0,1) 内的随机浮点数。
int rand32(int min, int max)	返回 `[min, max)` 内的一个随机 32 位整数。 用户可以只传入一个参数，该参数会判定为 `max`，此时 `min` 默认为 `0`。 如果不传入参数，此时会从带符号的 32 位 int 范围内随机返回。
int rand64(int min, int max)	返回 `[min, max)` 内的一个随机 64 位整数。 用户可以只传入一个参数，该参数会判定为 `max`，此时 `min` 默认为 `0`。 如果不传入参数，此时会从带符号的 64 位 int 范围内随机返回。
collect()	将收集的所有值放在一个列表中。
avg()	返回参数的平均值。
count()	返回参数的数量。
max()	返回参数的最大值。
min()	返回参数的最小值。
std()	返回参数的总体标准差。
sum()	返回参数的和。
bit_and()	逐位做 AND 操作。
bit_or()	逐位做 OR 操作。
bit_xor()	逐位做 XOR 操作。
int size()	返回列表或映射中元素的数量。

2022 vesoft Inc.

函数	说明
int range(int start, int end, int step)	返回 `[start,end]` 中指定步长的值组成的列表。步长 `step` 默认为 1。
int sign(double x)	返回 x 的正负号。 如果 x 为 `0`，则返回 `0`。 如果 x 为负数，则返回 `-1`。 如果 x 为正数，则返回 `1`。
double e()	返回自然对数的底 e（2.718281828459045）。
double pi()	返回数学常数π（3.141592653589793）。
double radians()	将角度转换为弧度。 `radians(180)` 返回 `3.141592653589793`。

- 字符串函数

函数	说明
int strcasecmp(string a, string b)	比较两个字符串（不区分大小写）。当 a=b 时，返回 0，当 a>b 是，返回大于 0 的数，当 a<b 时，返回小于 0 的数。
string lower(string a)	返回小写形式的字符串。
string toLower(string a)	和 `lower()` 相同。
string upper(string a)	返回大写形式的字符串。
string toUpper(string a)	和 `upper()` 相同。
int length(string a)	以字节为单位，返回给定字符串的长度。
string trim(string a)	删除字符串头部和尾部的空格。
string ltrim(string a)	删除字符串头部的空格。
string rtrim(string a)	删除字符串尾部的空格。
string left(string a, int count)	返回字符串左侧 `count` 个字符组成的子字符串。如果 `count` 超过字符串 a 的长度，则返回字符串 a。
string right(string a, int count)	返回字符串右侧 `count` 个字符组成的子字符串。如果 `count` 超过字符串 a 的长度，则返回字符串 a。
string lpad(string a, int size, string letters)	在字符串 a 的左侧填充 `letters` 字符串，并返回 `size` 长度的字符串。
string rpad(string a, int size, string letters)	在字符串 a 的右侧填充 `letters` 字符串，并返回 `size` 长度的字符串。
string substr(string a, int pos, int count)	从字符串 a 的指定位置 `pos` 开始（不包括 `pos` 位置的字符），提取右侧的 `count` 个字符，组成新的字符串并返回。
string substring(string a, int pos, int count)	和 `substr()` 相同。
string reverse(string)	逆序返回字符串。
string replace(string a, string b, string c)	将字符串 a 中的子字符串 b 替换为字符串 c。
list split(string a, string b)	在子字符串 b 处拆分字符串 a，返回一个字符串列表。
string toString()	将任意数据类型转换为字符串类型。
int hash()	获取任意对象的哈希值。

2022 vesoft Inc.

- 日期时间函数

函数	说明
int now()	根据当前系统返回当前时区的时间戳。
timestamp timestamp()	根据当前系统返回当前时区的时间戳。
date date()	根据当前系统返回当前日期（UTC 时间）。
time time()	根据当前系统返回当前时间（UTC 时间）。
datetime datetime()	根据当前系统返回当前日期和时间（UTC 时间）。

- Schema 函数

- 原生 nGQL 语句适用

函数	说明
id(vertex)	返回点 ID。数据类型和点 ID 的类型保持一致。
map properties(vertex)	返回点的所有属性。
map properties(edge)	返回边的所有属性。
string type(edge)	返回边的 Edge type。
src(edge)	返回边的起始点 ID。数据类型和点 ID 的类型保持一致。
dst(edge)	返回边的目的点 ID。数据类型和点 ID 的类型保持一致。
int rank(edge)	返回边的 rank。
vertex	返回点的信息。包括点 ID、Tag、属性和值。
edge	返回边的信息。包括 Edge type、起始点 ID、目的点 ID、rank、属性和值。
vertices	返回子图中的点的信息。详情参见 GET SUBGRAPH。
edges	返回子图中的边的信息。详情参见 GET SUBGRAPH。
path	返回路径信息。详情参见 FIND PATH。

- openCypher 兼容语句适用

函数	说明
id(<vertex>)	返回点 ID。数据类型和点 ID 的类型保持一致。
list tags(<vertex>)	返回点的 Tag，与 labels() 作用相同。
list labels(<vertex>)	返回点的 Tag，与 tags() 作用相同，用于兼容 openCypher 语法。
map properties(<vertex_or_edge>)	返回点或边的所有属性。
string type(<edge>)	返回边的 Edge type。
src(<edge>)	返回边的起始点 ID。数据类型和点 ID 的类型保持一致。
dst(<edge>)	返回边的目的点 ID。数据类型和点 ID 的类型保持一致。
vertex startNode(<path>)	获取一条边或一条路径并返回它的起始点 ID。
string endNode(<path>)	获取一条边或一条路径并返回它的目的点 ID。
int rank(<edge>)	返回边的 rank。

2022 vesoft Inc.

- 列表函数

函数	说明
keys(expr)	返回一个列表，包含字符串形式的点、边或映射的所有属性。
labels(vertex)	返回点的 Tag 列表。
nodes(path)	返回路径中所有点的列表。
range(start, end [, step])	返回 `[start,end]` 范围内固定步长的列表，默认步长 `step` 为 1。
relationships(path)	返回路径中所有关系的列表。
reverse(list)	返回将原列表逆序排列的新列表。
tail(list)	返回不包含原列表第一个元素的新列表。
head(list)	返回列表的第一个元素。
last(list)	返回列表的最后一个元素。
coalesce(list)	返回列表中第一个非空元素。
reduce()	请参见 reduce 函数。

- count 函数

函数	说明
count()	语法：`count({expr | *})` 。 `count()` 返回总行数（包括 NULL）。 `count(expr)` 返回满足表达式的非空值的总数。 `count()` 和 `size()` 是不同的。

- collect 函数

函数	说明
collect()	`collect()` 函数返回一个符合表达式返回结果的列表。该函数可以将多条记录或值合并进一个列表，实现数据聚合。

- reduce 函数

函数	语法	说明
reduce()	`reduce(<accumulator> = <initial>, <variable> IN <list>`	`reduce()` 将表达式逐个应用于列表中的元素，然后和累加器中的当前结果累加，最后返回完整结果。

- hash 函数

函数	说明
hash()	`hash()` 函数返回参数的哈希值。其参数可以是数字、字符串、列表、布尔值、NULL 等类型的值，或者计算结果为这些类型的表达式。`hash()` 函数采用 MurmurHash2 算法，种子（seed）为 `0xc70f6907UL` 。用户可以在 `MurmurHash2.h` 中查看其源代码。

- concat 函数

函数	说明
concat()	`concat()` 函数至少需要两个或以上字符串参数，并将所有参数连接成一个字符串。 语法：`concat(string1,string2,...)`

2022 vesoft Inc.

- concat_ws 函数

函数	说明
concat_ws()	`concat_ws()` 函数将两个或以上字符串参数与预定义的分隔符（separator）相连接。

- 谓词函数

谓词函数只返回 `true` 或 `false`，通常用于 `WHERE` 子句中。

```
<predicate>(<variable> IN <list> WHERE <condition>)
```

函数	说明
exists()	如果指定的属性在点、边或映射中存在，则返回 `true`，否则返回 `false`。
any()	如果指定的谓词适用于列表中的至少一个元素，则返回 `true`，否则返回 `false`。
all()	如果指定的谓词适用于列表中的每个元素，则返回 `true`，否则返回 `false`。
none()	如果指定的谓词不适用于列表中的任何一个元素，则返回 `true`，否则返回 `false`。
single()	如果指定的谓词适用于列表中的唯一一个元素，则返回 `true`，否则返回 `false`。

- CASE 表达式

`CASE` 表达式使用条件来过滤 nGQL 查询语句的结果，常用于 `YIELD` 和 `RETURN` 子句中。`CASE` 表达式会遍历所有条件，并在满足第一个条件时停止读取后续条件，然后返回结果。如果不满足任何条件，将通过 `ELSE` 子句返回结果。如果没有 `ELSE` 子句且不满足任何条件，则返回 `NULL`。

语法：

```
CASE <comparer>
WHEN <value> THEN <result>
[WHEN ...]
[ELSE <default>]
END
```

参数	说明
`comparer`	用于与 `value` 进行比较的值或者有效表达式。
`value`	和 `comparer` 进行比较，如果匹配，则满足此条件。
`result`	如果 `value` 匹配 `comparer`，则返回该 `result`。
`default`	如果没有条件匹配，则返回该 `default`。

3.7.2 通用查询语句

- MATCH

2022 vesoft Inc.

```
MATCH <pattern> [<clause_1>]  RETURN <output>  [<clause_2>];
```


2022 vesoft Inc.

模式	示例	说明
匹配点	`(v)`	用户可以在一对括号中使用自定义变量来表示模式中的点。例如 `(v)`。
匹配 Tag	`MATCH (v:player) RETURN v`	用户可以在点的右侧用 `:<tag_name>` 表示模式中的 Tag。
匹配多 Tag	`MATCH (v:player:team) RETURN v LIMIT 10`	用户可以用英文冒号（:）匹配多 Tag 的点。
匹配点的属性	`MATCH (v:player{name:"Tim Duncan"}) RETURN v`	用户可以在 Tag 的右侧用 `{<prop_name>: <prop_value>}` 表示模式中点的属性。
匹配单点 ID	`MATCH (v) WHERE id(v) == 'player101' RETURN v`	用户可以使用点 ID 去匹配点。`id()` 函数可以检索点的 ID。
匹配多点 ID	`MATCH (v:player { name: 'Tim Duncan' })--(v2) WHERE id(v2) IN ["player101", "player102"] RETURN v2`	要匹配多个点的 ID，可以用 `WHERE id(v) IN [vid_list]`。
匹配连接的点	`MATCH (v:player{name:"Tim Duncan"})--(v2) RETURN v2.player.name AS Name`	用户可以使用 `--` 符号表示两个方向的边，并匹配这些边连接的点。用户可以在 `--` 符号上增加 `<` 或 `>` 符号指定边的方向。
匹配路径	`MATCH p=(v:player{name:"Tim Duncan"})-->(v2) RETURN p`	连接起来的点和边构成了路径。用户可以使用自定义变量命名路径。
匹配边	`MATCH (v:player{name:"Tim Duncan"})-[e]-(v2) RETURN e` `MATCH ()<-[e]-()  RETURN e  LIMIT 3`	除了用 `--`、`-->`、`<--` 表示未命名的边之外，用户还可以在方括号中使用自定义变量命名边。例如 `-[e]-`。
匹配 Edge type	`MATCH ()-[e:follow]->() RETURN e LIMIT 5`	和点一样，用户可以用 `:<edge_type>` 表示模式中的 Edge type，例如 `-[e:follow]-`。
匹配边的属性	`MATCH (v:player{name:"Tim Duncan"})-[e:follow{degree:95}]->(v2) RETURN e`	用户可以用 `{<prop_name>: <prop_value>}` 表示模式中 Edge type 的属性，例如 `[e:follow{likeness:95}]`。
匹配多个 Edge type	`MATCH (v:player{name:"Tim Duncan"})-[e:follow | :serve]->(v2) RETURN e`	使用 `|` 可以匹配多个 Edge type，例如 `[e:follow | :serve]`。第一个 Edge type 前的英文冒号（:）不可省略，后续 Edge type 前的英文冒号可以省略，例如 `[e:follow | serve]`。
匹配多条边	`MATCH (v:player{name:"Tim Duncan"})-[]->(v2)<-[e:serve]-(v3) RETURN v2, v3`	用户可以扩展模式，匹配路径中的多条边。
匹配定长路径	`MATCH p=(v:player{name:"Tim Duncan"})-[e:follow*2]->(v2) RETURN DISTINCT v2 AS Friends`	用户可以在模式中使用 `:<edge_type>*<hop>` 匹配定长路径。`hop` 必须是一个非负整数。`e` 的数据类型是列表。
匹配变长路径	`MATCH p=(v:player{name:"Tim Duncan"})-[e:follow*1..3]->(v2) RETURN v2 AS Friends`	`minHop`：可选项。表示路径的最小长度。`minHop` 必须是一个非负整数，默认值为 1。 `maxHop`：可选项。表示路径的最大长度。`maxHop` 必须是一个非负整数，默认为无穷大。`e` 的数据类型是列表。
匹配多个 Edge type 的变长路径	`MATCH p=(v:player{name:"Tim Duncan"})-[e:follow | serve*2]->(v2) RETURN DISTINCT v2`	用户可以在变长或定长模式中指定多个 Edge type。`hop`、`minHop` 和 `maxHop` 对所有 Edge type 都生效。`e` 的数据类型是列表。
检索点或边的信息	`MATCH (v:player{name:"Tim Duncan"}) RETURN v` `MATCH (v:player{name:"Tim Duncan"})-[e]->(v2) RETURN e`	使用 `RETURN {<vertex_name> | <edge_name>}` 检索点或边的所有信息。
检索点 ID	`MATCH (v:player{name:"Tim Duncan"}) RETURN id(v)`	使用 `id()` 函数检索点 ID。
检索 Tag	`MATCH (v:player{name:"Tim Duncan"}) RETURN labels(v)`	使用 `labels()` 函数检索点上的 Tag 列表。 检索列表 `labels(v)` 中的第 N 个元素，可以使用 `labels(v)[n-1]`。

2022 vesoft Inc.

模式	示例	说明
检索点或边的单个属性	`MATCH (v:player{name:"Tim Duncan"}) RETURN v.player.age`	使用 `RETURN {<vertex_name> | <edge_name>}.<property>` 检索单个属性。使用 `AS` 设置属性的别名。
检索点或边的所有属性	`MATCH p=(v:player{name:"Tim Duncan"})-[]->(v2) RETURN properties(v2)`	使用 `properties()` 函数检索点或边的所有属性。
检索 Edge type	`MATCH p=(v:player{name:"Tim Duncan"})-[e]->() RETURN DISTINCT type(e)`	使用 `type()` 函数检索匹配的 Edge type。
检索路径	`MATCH p=(v:player{name:"Tim Duncan"})-[*3]->() RETURN p`	使用 `RETURN <path_name>` 检索匹配路径的所有信息。
检索路径中的点	`MATCH p=(v:player{name:"Tim Duncan"})-[]->(v2) RETURN nodes(p)`	使用 `nodes()` 函数检索路径中的所有点。
检索路径中的边	`MATCH p=(v:player{name:"Tim Duncan"})-[]->(v2) RETURN relationships(p)`	使用 `relationships()` 函数检索路径中的所有边。
检索路径长度	`MATCH p=(v:player{name:"Tim Duncan"})-[*..2]->(v2) RETURN p AS Paths, length(p) AS Length`	使用 `length()` 函数检索路径的长度。

- OPTIONAL MATCH

模式	示例	说明
作为MATCH语句的可选项去匹配图数据库中的模式	`MATCH (m)-[]->(n) WHERE id(m)=="player100" OPTIONAL MATCH (n)-[]->(l) WHERE id(n)=="player125" RETURN id(m),id(n),id(l)`	如果图数据库中没有对应的模式，对应的列返回NULL。

- LOOKUP

```
LOOKUP ON {<vertex_tag> | <edge_type>}
[WHERE <expression> [AND <expression> ...]]
YIELD <return_list> [AS <alias>]
```

模式	示例	说明
检索点	`LOOKUP ON player WHERE player.name == "Tony Parker" YIELD player.name AS name, player.age AS age`	返回 Tag 为 `player` 且 `name` 为 `Tony Parker` 的点。
检索边	`LOOKUP ON follow WHERE follow.degree == 90 YIELD follow.degree`	返回 Edge type 为 `follow` 且 `degree` 为 `90` 的边。
通过 Tag 列出所有点	`LOOKUP ON player YIELD properties(vertex),id(vertex)`	查找所有 Tag 为 `player` 的点 VID。
通过 Edge type 列出边	`LOOKUP ON like YIELD edge AS e`	查找 Edge type 为 `like` 的所有边的信息。
统计点	`LOOKUP ON player YIELD id(vertex)| YIELD COUNT(*) AS Player_Count`	统计 Tag 为 `player` 的点。
统计边	`LOOKUP ON like YIELD id(vertex)| YIELD COUNT(*) AS Like_Count`	统计 Edge type 为 `like` 的边。

2022 vesoft Inc.

- GO

```
GO [[<M> TO] <N> STEPS ] FROM <vertex_list>
OVER <edge_type_list> [{REVERSELY | BIDIRECT}]
[ WHERE <conditions> ]
YIELD [DISTINCT] <return_list>
[{SAMPLE <sample_list> | LIMIT <limit_list>}]
[| GROUP BY {col_name | expr | position} YIELD <col_name>]
[| ORDER BY <expression> [{ASC | DESC}]]
[| LIMIT [<offset_value>,] <number_rows>]
```

示例	说明
`GO FROM "player102" OVER serve YIELD dst(edge)`	返回 player102 所属队伍。
`GO 2 STEPS FROM "player102" OVER follow YIELD dst(edge)`	返回距离 player102 两跳的朋友。
`GO FROM "player100", "player102" OVER serve WHERE properties(edge).start_year > 1995 YIELD DISTINCT properties($$).name AS team_name, properties(edge).start_year AS start_year, properties($^).name AS player_name`	添加过滤条件。
`GO FROM "player100" OVER follow, serve YIELD properties(edge).degree, properties(edge).start_year`	遍历多个 Edge type。属性没有值时，会显示 `UNKNOWN_PROP` 。
`GO FROM "player100" OVER follow REVERSELY YIELD src(edge) AS destination`	返回 player100 入方向的邻居点。
`GO FROM "player100" OVER follow REVERSELY YIELD src(edge) AS id | GO FROM $-.id OVER serve WHERE properties($^).age > 20 YIELD properties($^).name AS FriendOf, properties($$).name AS Team`	查询 player100 的朋友和朋友所属队伍。
`GO FROM "player102" OVER follow YIELD dst(edge) AS both`	返回 player102 所有邻居点。
`GO 2 STEPS FROM "player100" OVER follow YIELD src(edge) AS src, dst(edge) AS dst, properties($$).age AS age | GROUP BY $-.dst YIELD $-.dst AS dst, collect_set($-.src) AS src, collect($-.age) AS age`	根据年龄分组。

2022 vesoft Inc.

- FETCH

- 获取点的属性值

```
FETCH PROP ON {<tag_name>[, tag_name ...] | *}
<vid> [, vid ...]
YIELD <return_list> [AS <alias>]
```

示例	说明
`FETCH PROP ON player "player100" YIELD properties(vertex)`	在 `FETCH` 语句中指定 Tag 获取对应点的属性值。
`FETCH PROP ON player "player100" YIELD player.name AS name`	使用 `YIELD` 子句指定返回的属性。
`FETCH PROP ON player "player101", "player102", "player103" YIELD properties(vertex)`	指定多个点 ID 获取多个点的属性值，点之间用英文逗号（,）分隔。
`FETCH PROP ON player, t1 "player100", "player103" YIELD properties(vertex)`	在 `FETCH` 语句中指定多个 Tag 获取属性值。Tag 之间用英文逗号（,）分隔。
`FETCH PROP ON * "player100", "player106", "team200" YIELD properties(vertex)`	在 `FETCH` 语句中使用 `*` 获取当前图空间所有标签里，点的属性值。

- 获取边的属性值

```
FETCH PROP ON <edge_type> <src_vid> -> <dst_vid>[@<rank>] [, <src_vid> -> <dst_vid> ...]
YIELD <output>;
```

示例	说明
`FETCH PROP ON serve "player100" -> "team204" YIELD properties(edge)`	获取连接 player100 和 team204 的边 serve 的所有属性值。
`FETCH PROP ON serve "player100" -> "team204" YIELD serve.start_year`	使用 `YIELD` 子句指定返回的属性。
`FETCH PROP ON serve "player100" -> "team204", "player133" -> "team202" YIELD properties(edge)`	指定多个边模式 (`<src_vid> -> <dst_vid>[@<rank>]`) 获取多个边的属性值。模式之间用英文逗号（,）分隔。
`FETCH PROP ON serve "player100" -> "team204"@1 YIELD properties(edge)`	要获取 rank 不为 0 的边，请在 FETCH 语句中设置 rank。
`GO FROM "player101" OVER follow YIELD follow._src AS s, follow._dst AS d | FETCH PROP ON follow $-.s -> $-.d YIELD follow.degree`	返回从点 player101 开始的 follow 边的 degree 值。
`$var = GO FROM "player101" OVER follow YIELD follow._src AS s, follow._dst AS d; FETCH PROP ON follow $var.s -> $var.d YIELD follow.degree`	自定义变量构建查询。

2022 vesoft Inc.

- SHOW

语句	语法	示例	说明
SHOW CHARSET	`SHOW CHARSET`	`SHOW CHARSET`	显示当前的字符集。
SHOW COLLATION	`SHOW COLLATION`	`SHOW COLLATION`	显示当前的排序规则。
SHOW CREATE SPACE	`SHOW CREATE SPACE <space_name>`	`SHOW CREATE SPACE basketballplayer`	显示指定图空间的创建语句。
SHOW CREATE TAG/EDGE	`SHOW CREATE {TAG <tag_name> | EDGE <edge_name>}`	`SHOW CREATE TAG player`	显示指定 Tag/Edge type 的基本信息。
SHOW HOSTS	`SHOW HOSTS [GRAPH | STORAGE | META]`	`SHOW HOSTS` `SHOW HOSTS GRAPH`	显示 Graph、Storage、Meta 服务主机信息、版本信息。
SHOW INDEX STATUS	`SHOW {TAG | EDGE} INDEX STATUS`	`SHOW TAG INDEX STATUS`	重建原生索引的作业状态，以便确定重建索引是否成功。
SHOW INDEXES	`SHOW {TAG | EDGE} INDEXES`	`SHOW TAG INDEXES`	列出当前图空间内的所有 Tag 和 Edge type（包括属性）的索引。
SHOW PARTS	`SHOW PARTS [<part_id>]`	`SHOW PARTS`	显示图空间中指定分片或所有分片的信息。
SHOW ROLES	`SHOW ROLES IN <space_name>`	`SHOW ROLES in basketballplayer`	显示分配给用户的角色信息。
SHOW SNAPSHOTS	`SHOW SNAPSHOTS`	`SHOW SNAPSHOTS`	显示所有快照信息。
SHOW SPACES	`SHOW SPACES`	`SHOW SPACES`	显示现存的图空间。
SHOW STATS	`SHOW STATS`	`SHOW STATS`	显示最近 `STATS` 作业收集的图空间统计信息。
SHOW TAGS/EDGES	`SHOW TAGS | EDGES`	`SHOW TAGS` 、`SHOW EDGES`	显示当前图空间内的所有 Tag/Edge type。
SHOW USERS	`SHOW USERS`	`SHOW USERS`	显示用户信息。
SHOW SESSIONS	`SHOW SESSIONS`	`SHOW SESSIONS`	显示所有会话信息。
SHOW SESSIONS	`SHOW SESSION <Session_Id>`	`SHOW SESSION 1623304491050858`	指定会话 ID 进行查看。
SHOW QUERIES	`SHOW [ALL] QUERIES`	`SHOW QUERIES`	查看当前 Session 中正在执行的查询请求信息。
SHOW META LEADER	`SHOW META LEADER`	`SHOW META LEADER`	显示当前 Meta 集群的 leader 信息。

2022 vesoft Inc.

3.7.3 子句和选项

子句	语法	示例	说明
GROUP BY	`GROUP BY <var> YIELD <var>, <aggregation_function(var)>`	`GO FROM "player100" OVER follow BIDIRECT YIELD $$.player.name as Name | GROUP BY $-.Name YIELD $-.Name as Player, count(*) AS Name_Count`	查找所有连接到 player100 的点，并根据他们的姓名进行分组，返回姓名的出现次数。
LIMIT	`YIELD <var> [| LIMIT [<offset_value>,] <number_rows>]`	`GO FROM "player100" OVER follow REVERSELY YIELD $$.player.name AS Friend, $$.player.age AS Age | ORDER BY $-.Age, $-.Friend | LIMIT 1, 3`	从排序结果中返回第 2 行开始的 3 行数据。
SKIP	`RETURN <var> [SKIP <offset>] [LIMIT <number_rows>]`	`MATCH (v:player{name:"Tim Duncan"}) --> (v2) RETURN v2.player.name AS Name, v2.player.age AS Age ORDER BY Age DESC SKIP 1`	用户可以单独使用 `SKIP <offset>` 设置偏移量，后面不需要添加 `LIMIT <number_rows>` 。
SAMPLE	`<go_statement> SAMPLE <sample_list>;`	`GO 3 STEPS FROM "player100" OVER * YIELD properties($$).name AS NAME, properties($$).age AS Age SAMPLE [1,2,3];`	在结果集中均匀取样并返回指定数量的数据。
ORDER BY	`<YIELD clause> ORDER BY <expression> [ASC | DESC] [, <expression> [ASC | DESC] ...]`	`FETCH PROP ON player "player100", "player101", "player102", "player103" YIELD player.age AS age, player.name AS name | ORDER BY $-.age ASC, $-.name DESC`	`ORDER BY` 子句指定输出结果的排序规则。
RETURN	`RETURN {<vertex_name>|<edge_name>|<vertex_name>.<property>|<edge_name>.<property>|...}`	`MATCH (v:player) RETURN v.player.name, v.player.age LIMIT 3`	返回点的属性为 `name` 和 `age` 的前三行值。
TTL	`CREATE TAG <tag_name>(<property_name_1> <property_value_1>, <property_name_2> <property_value_2>, ...) ttl_duration= <value_int>, ttl_col = <property_name>`	`CREATE TAG t2(a int, b int, c string) ttl_duration= 100, ttl_col = "a"`	创建 Tag 并设置 TTL 选项。
WHERE	`WHERE {<vertex|edge_alias>.<property_name> {>|==|<|...} <value>...}`	`MATCH (v:player) WHERE v.player.name == "Tim Duncan" XOR (v.player.age < 30 AND v.player.name == "Yao Ming") OR NOT (v.player.name == "Yao Ming" OR v.player.name == "Tim Duncan") RETURN v.player.name, v.player.age`	`WHERE` 子句可以根据条件过滤输出结果，通常用于 `GO` 和 `LOOKUP` 语句， `MATCH` 和 `WITH` 语句。
YIELD	`YIELD [DISTINCT] <col> [AS <alias>] [, <col> [AS <alias>] ...] [WHERE <conditions>];`	`GO FROM "player100" OVER follow YIELD dst(edge) AS ID | FETCH PROP ON player $-.ID YIELD player.age AS Age \| YIELD AVG($-.Age) as Avg_age, count(*)as Num_friends`	查找 player100 关注的 player，并计算他们的平均年龄。
WITH	`MATCH $expressions WITH {nodes()| labels()|...}`	`MATCH p=(v:player{name:"Tim Duncan"})--() WITH nodes(p) AS n UNWIND n AS n1 RETURN DISTINCT n1`	`WITH` 子句可以获取并处理查询前半部分的结果，并将处理结果作为输入传递给查询的后半部分。
UNWIND	`UNWIND <list> AS <alias> <RETURN clause>`	`UNWIND [1,2,3] AS n RETURN n`	拆分列表。

2022 vesoft Inc.

3.7.4 图空间语句

语句	语法	示例	说明
CREATE SPACE	`CREATE SPACE [IF NOT EXISTS] <graph_space_name> ( [partition_num = <partition_number>,] [replica_factor = <replica_number>,] vid_type = {FIXED_STRING(<N>) | INT[64]} ) [COMMENT = '<comment>']`	`CREATE SPACE my_space_1 (vid_type=FIXED_STRING(30))`	创建一个新的图空间。
CREATE SPACE	`CREATE SPACE <new_graph_space_name> AS <old_graph_space_name>`	`CREATE SPACE my_space_4 as my_space_3`	克隆现有图空间的 Schema。
USE	`USE <graph_space_name>`	`USE space1`	指定一个图空间，或切换到另一个图空间，将其作为后续查询的工作空间。
SHOW SPACES	`SHOW SPACES`	`SHOW SPACES`	列出 Nebula Graph 示例中的所有图空间。
DESCRIBE SPACE	`DESC[RIBE] SPACE <graph_space_name>`	`DESCRIBE SPACE basketballplayer`	显示指定图空间的信息。
CLEAR SPACE	`CLEAR SPACE [IF EXISTS] <graph_space_name>`	清空图空间中的点和边，但不会删除图空间本身以及其中的 Schema 信息。	
DROP SPACE	`DROP SPACE [IF EXISTS] <graph_space_name>`	`DROP SPACE basketballplayer`	删除指定图空间的所有内容。

3.7.5 TAG 语句

语句	语法	示例	说明
CREATE TAG	`CREATE TAG [IF NOT EXISTS] <tag_name> (<prop_name> <data_type> [NULL | NOT NULL] [DEFAULT <default_value>] [COMMENT '<comment>'] [{, <prop_name> <data_type> [NULL | NOT NULL] [DEFAULT <default_value>] [COMMENT '<comment>']} ...]) [TTL_DURATION = <ttl_duration>] [TTL_COL = <prop_name>] [COMMENT = '<comment>']`	`CREATE TAG woman(name string, age int, married bool, salary double, create_time timestamp) TTL_DURATION = 100, TTL_COL = "create_time"`	通过指定名称创建一个 Tag。
DROP TAG	`DROP TAG [IF EXISTS] <tag_name>`	`DROP TAG test;`	删除当前工作空间内所有点上的指定 Tag。
ALTER TAG	`ALTER TAG <tag_name> <alter_definition> [, alter_definition] ...] [ttl_definition [, ttl_definition] ... ] [COMMENT = '<comment>']`	`ALTER TAG t1 ADD (p3 int, p4 string)`	修改 Tag 的结构。例如增删属性、修改数据类型，也可以为属性设置、修改 TTL（Time-To-Live）。
SHOW TAGS	`SHOW TAGS`	`SHOW TAGS`	显示当前图空间内的所有 Tag 名称。
DESCRIBE TAG	`DESC[RIBE] TAG <tag_name>`	`DESCRIBE TAG player`	查看指定 Tag 的详细信息，例如字段名称、数据类型等。
DELETE TAG	`DELETE TAG <tag_name_list> FROM <VID>`	`DELETE TAG test1 FROM "test"`	删除指定点上的指定 Tag。

2022 vesoft Inc.

3.7.6 Edge type 语句

语句	语法	示例	说明
CREATE EDGE	`CREATE EDGE [IF NOT EXISTS] <edge_type_name> (<prop_name> <data_type> [NULL | NOT NULL] [DEFAULT <default_value>] [COMMENT '<comment>'] [{, <prop_name> <data_type> [NULL | NOT NULL] [DEFAULT <default_value>] [COMMENT '<comment>']} ...]) [TTL_DURATION = <ttl_duration>] [TTL_COL = <prop_name>] [COMMENT = '<comment>']`	`CREATE EDGE e1(p1 string, p2 int, p3 timestamp) TTL_DURATION = 100, TTL_COL = "p2"`	指定名称创建一个 Edge type。
DROP EDGE	`DROP EDGE [IF EXISTS] <edge_type_name>`	`DROP EDGE e1`	删除当前工作空间内的指定 Edge type。
ALTER EDGE	`ALTER EDGE <edge_type_name> <alter_definition> [, alter_definition] ...] [ttl_definition [, ttl_definition] ... ] [COMMENT = '<comment>']`	`ALTER EDGE e1 ADD (p3 int, p4 string)`	修改 Edge type 的结构。
SHOW EDGES	`SHOW EDGES`	`SHOW EDGES`	显示当前图空间内的所有 Edge type 名称。
DESCRIBE EDGE	`DESC[RIBE] EDGE <edge_type_name>`	`DESCRIBE EDGE follow`	查看指定 Edge type 的详细信息，例如字段名称、数据类型等。

3.7.7 点语句

语句	语法	示例	说明
INSERT VERTEX	`INSERT VERTEX [IF NOT EXISTS] [tag_props, [tag_props] ...] VALUES <vid>: ([prop_value_list])`	`INSERT VERTEX t2 (name, age) VALUES "13":("n3", 12), "14": ("n4", 8)`	在 Nebula Graph 实例的指定图空间中插入一个或多个点。
DELETE VERTEX	`DELETE VERTEX <vid> [, <vid> ...]`	`DELETE VERTEX "team1"`	删除点，以及点关联的出边和入边。
UPDATE VERTEX	`UPDATE VERTEX ON <tag_name> <vid> SET <update_prop> [WHEN <condition>] [YIELD <output>]`	`UPDATE VERTEX ON player "player101" SET age = age + 2`	修改点上 Tag 的属性值。
UPSERT VERTEX	`UPSERT VERTEX ON <tag> <vid> SET <update_prop> [WHEN <condition>] [YIELD <output>]`	`UPSERT VERTEX ON player "player667" SET age = 31`	结合 `UPDATE` 和 `INSERT`，如果点存在，会修改点的属性值；如果点不存在，会插入新的点。

2022 vesoft Inc.

3.7.8 边语句

语句	语法	示例	说明
INSERT EDGE	`INSERT EDGE [IF NOT EXISTS] <edge_type> ( <prop_name_list> ) VALUES <src_vid> -> <dst_vid>[@<rank>] : ( <prop_value_list> ) [, <src_vid> -> <dst_vid>[@<rank>] : ( <prop_value_list> ), ...]`	`INSERT EDGE e2 (name, age) VALUES "11"->"13": ("n1", 1)`	在 Nebula Graph 实例的指定图空间中插入一条或多条边。
DELETE EDGE	`DELETE EDGE <edge_type> <src_vid> -> <dst_vid>[@<rank>] [, <src_vid> -> <dst_vid>[@<rank>] ...]`	`DELETE EDGE serve "player100" -> "team204"@0`	删除边。一次可以删除一条或多条边。
UPDATE EDGE	`UPDATE EDGE ON <edge_type> <src_vid> -> <dst_vid> [@<rank>] SET <update_prop> [WHEN <condition>] [YIELD <output>]`	`UPDATE EDGE ON serve "player100" -> "team204"@0 SET start_year = start_year + 1`	修改边上 Edge type 的属性。
UPSERT EDGE	`UPSERT EDGE ON <edge_type> <src_vid> -> <dst_vid> [@rank] SET <update_prop> [WHEN <condition>] [YIELD <properties>]`	`UPSERT EDGE on serve "player666" -> "team200"@0 SET end_year = 2021`	结合 `UPDATE` 和 `INSERT`，如果边存在，会更新边的属性；如果边不存在，会插入新的边。

2022 vesoft Inc.

3.7.9 索引

- 原生索引

  索引配合 `LOOKUP` 和 `MATCH` 语句使用。

语句	语法	示例	说明
CREATE INDEX	`CREATE {TAG | EDGE} INDEX [IF NOT EXISTS] <index_name> ON {<tag_name> | <edge_name>} ([<prop_name_list>]) [COMMENT = '<comment>']`	`CREATE TAG INDEX player_index on player()`	对 Tag、EdgeType 或其属性创建原生索引。
SHOW CREATE INDEX	`SHOW CREATE {TAG | EDGE} INDEX <index_name>`	`show create tag index index_2`	创建 Tag 或者 Edge type 时使用的 nGQL 语句，其中包含索引的详细信息，例如其关联的属性。
SHOW INDEXES	`SHOW {TAG | EDGE} INDEXES`	`SHOW TAG INDEXES`	列出当前图空间内的所有 Tag 和 Edge type（包括属性）的索引。
DESCRIBE INDEX	`DESCRIBE {TAG | EDGE} INDEX <index_name>`	`DESCRIBE TAG INDEX player_index_0`	查看指定索引的信息，包括索引的属性名称（Field）和数据类型（Type）。
REBUILD INDEX	`REBUILD {TAG | EDGE} INDEX [<index_name_list>]`	`REBUILD TAG INDEX single_person_index`	重建索引。索引功能不会自动对其创建之前已存在的存量数据生效。
SHOW INDEX STATUS	`SHOW {TAG | EDGE} INDEX STATUS`	`SHOW TAG INDEX STATUS`	查看索引名称和对应的状态。
DROP INDEX	`DROP {TAG | EDGE} INDEX [IF EXISTS] <index_name>`	`DROP TAG INDEX player_index_0`	删除当前图空间中已存在的索引。

- 全文索引

语法	示例	说明
`SIGN IN TEXT SERVICE [(<elastic_ip:port> [,<username>, <password>]), (<elastic_ip:port>), ...]`	`SIGN IN TEXT SERVICE (127.0.0.1:9200)`	Nebula Graph 的全文索引是基于 Elasticsearch 实现，部署 Elasticsearch 集群之后，可以使用 `SIGN IN` 语句登录 Elasticsearch 客户端。
`SHOW TEXT SEARCH CLIENTS`	`SHOW TEXT SEARCH CLIENTS`	列出文本搜索客户端。
`SIGN OUT TEXT SERVICE`	`SIGN OUT TEXT SERVICE`	退出所有文本搜索客户端。
`CREATE FULLTEXT {TAG | EDGE} INDEX <index_name> ON {<tag_name> | <edge_name>} ([<prop_name_list>])`	`CREATE FULLTEXT TAG INDEX nebula_index_1 ON player(name)`	创建全文索引。
`SHOW FULLTEXT INDEXES`	`SHOW FULLTEXT INDEXES`	显示全文索引。
`REBUILD FULLTEXT INDEX`	`REBUILD FULLTEXT INDEX`	重建全文索引。
`DROP FULLTEXT INDEX <index_name>`	`DROP FULLTEXT INDEX nebula_index_1`	删除全文索引。
`LOOKUP ON {<tag> | <edge_type>} WHERE <expression> [YIELD <return_list>]`	`LOOKUP ON player WHERE FUZZY(player.name, "Tim Dunncan", AUTO, OR) YIELD player.name`	使用查询选项。

2022 vesoft Inc.

3.7.10 子图和路径

类型	语法	示例	说明
子图	`GET SUBGRAPH [WITH PROP] [<step_count> STEPS] FROM {<vid>, <vid>...} [{IN | OUT | BOTH} <edge_type>, <edge_type>...] YIELD [VERTICES AS <vertex_alias>] [,EDGES AS <edge_alias>]`	`GET SUBGRAPH 1 STEPS FROM "player100" YIELD VERTICES AS nodes, EDGES AS relationships`	指定 Edge type 的起始点可以到达的点和边的信息，返回子图信息。
路径	`FIND { SHORTEST | ALL | NOLOOP } PATH [WITH PROP] FROM <vertex_id_list> TO <vertex_id_list> OVER <edge_type_list> [REVERSELY | BIDIRECT] [<WHERE clause>] [UPTO <N> STEPS] YIELD path as <alias> [| ORDER BY $-.path] [| LIMIT <M>]`	`FIND SHORTEST PATH FROM "player102" TO "team204" OVER * YIELD path as p`	查找指定起始点和目的点之间的路径。返回的路径格式类似于 `(<vertex_id>)-[:<edge_type_name>@<rank>]->(<vertex_id)` 。

3.7.11 查询调优

类型	语法	示例	说明
EXPLAIN	`EXPLAIN [format="row" | "dot"] <your_nGQL_statement>`	`EXPLAIN format="row" SHOW TAGS` `EXPLAIN format="dot" SHOW TAGS`	输出 nGQL 语句的执行计划，但不会执行 nGQL 语句。
PROFILE	`PROFILE [format="row" | "dot"] <your_nGQL_statement>`	`PROFILE format="row" SHOW TAGS` `EXPLAIN format="dot" SHOW TAGS`	执行 nGQL 语句，然后输出执行计划和执行概要。

2022 vesoft Inc.

3.7.12 运维

- BALANCE

语法	说明
`BALANCE LEADER`	在当前图空间内启动任务均衡分布 leader。该命令会返回任务 ID。

- 作业管理

语法	说明
`SUBMIT JOB COMPACT`	触发 RocksDB 的长耗时 `compact` 操作。
`SUBMIT JOB FLUSH`	将内存中的 RocksDB memfile 写入硬盘。
`SUBMIT JOB STATS`	启动一个作业，该作业对当前图空间进行统计。作业完成后，用户可以使用 `SHOW STATS` 语句列出统计结果。
`SHOW JOB <job_id>`	显示当前图空间内指定作业和相关任务的信息。Meta 服务将 `SUBMIT JOB` 请求解析为多个任务，然后分配给进程 nebula-storaged。
`SHOW JOBS`	列出当前图空间内所有未过期的作业。
`STOP JOB`	停止当前图空间内未完成的作业。
`RECOVER JOB`	重新执行当前图空间内失败的作业，并返回已恢复的作业数量。

- 终止查询

语法	示例	说明
`KILL QUERY (session=<session_id>, plan=<plan_id>)`	`KILL QUERY(SESSION=1625553545984255,PLAN=163)`	在一个会话中执行命令终止另一个会话中的查询。

最后更新: June 20, 2023

2022 vesoft Inc.

4. nGQL 指南

4.1 nGQL 概述

4.1.1 什么是 nGQL

nGQL（Nebula Graph Query Language）是 Nebula Graph 使用的的声明式图查询语言，支持灵活高效的图模式，而且 nGQL 是为开发和运维人员设计的类 SQL 查询语言，易于学习。

nGQL 是一个进行中的项目，会持续发布新特性和优化，因此可能会出现语法和实际操作不一致的问题，如果遇到此类问题，请提交 issue 通知 Nebula Graph 团队。Nebula Graph 3.0 及更新版本正在支持 openCypher 9。

nGQL 可以做什么

- 支持图遍历
- 支持模式匹配
- 支持聚合
- 支持修改图
- 支持访问控制
- 支持聚合查询
- 支持索引
- 支持大部分 openCypher 9 图查询语法（不支持修改和控制语法）

示例数据 **Basketballplayer**

用户可以下载 Nebula Graph 示例数据 basketballplayer 文件，然后使用 Nebula Graph Console，使用选项 `-f` 执行脚本。

> **Note**
>
> 导入示例数据前，确保已执行 `ADD HOSTS` 命令将 Storage 主机增加至集群中。更多信息，请参见管理 Storage 主机。

占位标识符和占位符值

Nebula Graph 查询语言 nGQL 参照以下标准设计：

- (Draft) ISO/IEC JTC1 N14279 SC 32 - Database_Languages - GQL
- (Draft) ISO/IEC JTC1 SC32 N3228 - SQL_Property_Graph_Queries - SQLPGQ
- OpenCypher 9

在模板代码中，任何非关键字、字面值或标点符号的标记都是占位符标识符或占位符值。

2022 vesoft Inc.

本文中 nGQL 语法符号的说明如下。

符号	含义
< >	语法元素的名称。
:	定义元素的公式。
[ ]	可选元素。
{ }	显式的指定元素。
|	所有可选的元素。
...	可以重复多次。

例如创建点的 nGQL 语法:

```
INSERT VERTEX [IF NOT EXISTS] [tag_props, [tag_props] ...]
VALUES <vid>: ([prop_value_list])

tag_props:
  tag_name ([prop_name_list])

prop_name_list:
   [prop_name [, prop_name] ...]

prop_value_list:
   [prop_value [, prop_value] ...]
```

示例语句:

```
nebula> CREATE TAG IF NOT EXISTS player(name string, age int);
```

关于 openCypher 兼容性

原生 NGQL 和 OPENCYPHER 的关系

原生 nGQL 是由 Nebula Graph 自行创造和实现的图查询语言。openCypher 是由 openCypher Implementers Group 组织所开源和维护的图查询语言，最新版本为 openCypher 9。

由于 nGQL 语言部分兼容了 openCypher，这个部分在本文中称为 openCypher 兼容语句。

Note

`nGQL 语言 = 原生 nGQL 语句 + openCypher 兼容语句`

行为未定义

不要在同一个复合语句中，同时使用 `原生 nGQL 语句` 和 `openCypher 兼容语句`，其行为是未定义的。

NGQL 完全兼容 OPENCYPHER 9 吗?

不。

2022 vesoft Inc.

openCypher 兼容性

nGQL 设计目标仅为兼容部分的 DQL 语句(match, optional match, with等)。

不计划兼容任何 DDL，DML，DCL;

不计划兼容 Bolt 协议;

不计划兼容 APOC 与 GDS。

在本文搜索 "compatibility" 或者 “兼容性” 查看具体不兼容的细节。

在 Nebula Graph Issues 中已经列出已知的兼容错误。如果发现这种类型的新问题，请提交问题并附带 `incompatible` 标签。

NGQL 和 OPENCYPHER 9 的 主要差异有哪些?

类别	**openCypher 9**	**nGQL**
Schema	弱 Schema	强 Schema
相等运算符	`=`	`==`
数学求幂	`^`	使用 `pow(x, y)` 替代 `^`。
边 Rank	无此概念	用 `@rank` 设置。
语句	-	不支持 openCypher 9 的所有 DML 语句（如 `CREATE` 、`MERGE` 等），不支持所有的 DCL，和支持部分 `MATCH`, `OPTIONAL MATCH` 语法和函数。
语句文本换行	换行符	`\` + 换行符
Label 与 Tag 是不同的概念	Label 用于寻找点（点的索引）。	Tag 用于定义点的一种类型及相应的属性，无索引功能。
预编译与参数化查询	支持	仅支持参数化查询。

Compatibility

请注意 openCypher 9 和 Cypher 在语法和许可上有不同:

1. Cypher 要求所有 Cypher 语句必须“显式地在一个事务中”执行，而 openCypher 没有这样的要求。另外，nGQL 没有事务及隔离性。
2. Cypher 企业版功能有多种的约束（constraints），包括 Unique node property constraints、Node property existence constraints、Relationship property existence constraints、Node key constraints。 OpenCypher 标准中没有约束。 而 nGQL 是强 Schema 系统，前述的约束大多通过 Schema 定义可实现（包括 NOT NULL），唯一不能支持的功能是“属性值唯一性”（UNIQUE constraint）。
3. Cypher 有 APoC，openCypher 9 没有 APoC。Cypher 有 Blot 协议支持要求， openCypher 9 没有。

哪里可以找到更多 NGQL 的示例?

用户可以在 Nebula Graph GitHub 的 features 目录内查看超过 2500 条 nGQL 示例。

features 目录内包含很多。features 格式的文件，每个文件都记录了使用 nGQL 的场景和示例。例如:

```
Feature: Basic match

  Background:
    Given a graph with space named "basketballplayer"

  Scenario: Single node
    When executing query:
      """
      MATCH (v:player {name: "Yao Ming"}) RETURN v;
      """
    Then the result should be, in any order, with relax comparison:
      | v                                                |
      | ("player133" :player{age: 38, name: "Yao Ming"}) |
```


2022 vesoft Inc.

```
  Scenario: One step
    When executing query:
      """
      MATCH (v1:player{name: "LeBron James"}) -[r]-> (v2)
      RETURN type(r) AS Type, v2.player.name AS Name
      """
    Then the result should be, in any order:

      | Type     | Name        |
      | "follow" | "Ray Allen" |
      | "serve"  | "Lakers"    |
      | "serve"  | "Heat"      |
      | "serve"  | "Cavaliers" |

Feature:  Comparison of where clause

  Background:
    Given a graph with space named "basketballplayer"

    Scenario: push edge props filter down
      When profiling query:
        """
        GO FROM "player100" OVER follow
        WHERE properties(edge).degree IN [v IN [95,99] WHERE v > 0]
        YIELD dst(edge), properties(edge).degree
        """
      Then the result should be, in any order:
        | follow._dst | follow.degree |
        | "player101" | 95            |
        | "player125" | 95            |
      And the execution plan should be:
        | id | name         | dependencies | operator info                                                                |
        | 0  | Project      | 1            |                                                                              |
        | 1  | GetNeighbors | 2            | {"filter": "(properties(edge).degree IN [v IN [95,99] WHERE (v>0)])"} |
        | 2  | Start        |              |                                                                              |
```

示例中的关键字说明如下。

关键字	说明
`Feature`	描述当前文档的主题。
`Background`	描述当前文档的背景信息。
`Given`	描述执行示例语句的前提条件。
`Scenario`	描述具体场景。如果场景之前有 `@skip` 标识，表示这个场景下示例语句可能无法正常工作，请不要在生产环境中使用该示例语句。
`When`	描述要执行的 nGQL 示例语句。可以是 `executing query` 或 `profiling query`。
`Then`	描述执行 `When` 内语句的预期返回结果。如果返回结果和文档不同，请提交 issue 通知 Nebula Graph 团队。
`And`	描述执行 `When` 内语句的副作用或执行计划。
`@skip`	跳过这个示例。通常表示测试代码还没有准备好。

欢迎增加更多 tck case，在 CI/CD 中自动回归所使用的语句。

是否支持 **TINKERPOP GREMLIN**?

不支持。也没有计划。

是否支持 **W3C** 的 **RDF**（**SPARQL**） 或 **GRAPHQL** 等?

不支持。也没有计划。

Nebula Graph 的数据模型是属性图，是一个强 Schema 系统，不支持 RDF 标准。

nGQL 也不支持 SPARQL 和 GraphQL。

最后更新: June 20, 2023

2022 vesoft Inc.

4.1.2 模式

模式（pattern）和图模式匹配，是图查询语言的核心功能，本文介绍 Nebula Graph 设计的各种模式，部分还未实现。

单点模式

点用一对括号来描述，通常包含一个名称。例如：

```
(a)
```

示例为一个简单的模式，描述了单个点，并使用变量 `a` 命名该点。

多点关联模式

多个点通过边相连是常见的结构，模式用箭头来描述两个点之间的边。例如：

```
(a)-[]->(b)
```

示例为一个简单的数据结构：两个点和一条连接两个点的边，两个点分别为 `a` 和 `b`，边是有方向的，从 `a` 到 `b`。

这种描述点和边的方式可以扩展到任意数量的点和边，例如：

```
(a)-[]->(b)<-[]-(c)
```

这样的一系列点和边称为 `路径`（path）。

只有在涉及某个点时，才需要命名这个点。如果不涉及这个点，则可以省略名称，例如：

```
(a)-[]->()<-[]-(c)
```

Tag 模式

> **Note**
>
> nGQL 中的 `Tag` 概念与 openCypher 中的 `Label` 有一些不同。例如，必须创建一个 `Tag` 之后才能使用它，而且 `Tag` 还定义了属性的类型。

模式除了简单地描述图中的点之外，还可以描述点的 Tag。例如：

```
(a:User)-[]->(b)
```

模式也可以描述有多个 Tag 的点，例如：

```
(a:User:Admin)-[]->(b)
```

属性模式

点和边是图的基本结构。nGQL 在这两种结构上都可以增加属性，方便实现更丰富的模型。

在模式中，属性的表示方式为：用花括号括起一些键值对，用英文逗号分隔。例如一个点有两个属性：

```
(a {name: 'Andres', sport: 'Brazilian Ju-Jitsu'})
```

在这个点上可以有一条边是：

```
(a)-[{blocked: false}]->(b)
```


2022 vesoft Inc.

边模式

描述一条边最简单的方法是使用箭头连接两个点。

可以用以下方式描述边以及它的方向性。如果不关心边的方向，可以省略箭头，例如：

```
(a)-[]-(b)
```

和点一样，边也可以命名。一对方括号用于分隔箭头，变量放在两者之间。例如：

```
(a)-[r]->(b)
```

和点上的 Tag 一样，边也可以有类型。描述边的类型，例如：

```
(a)-[r:REL_TYPE]->(b)
```

和点上的 Tag 不同，一条边只能有一种 Edge type。但是如果我们想描述多个可选 Edge type，可以用管道符号（|）将可选值分开，例如：

```
(a)-[r:TYPE1|TYPE2]->(b)
```

和点一样，边的名称可以省略，例如：

```
(a)-[:REL_TYPE]->(b)
```

变长模式

在图中指定边的长度来描述多条边（以及中间的点）组成的一条长路径，不需要使用多个点和边来描述。例如：

```
(a)-[*2]->(b)
```

该模式描述了 3 点 2 边组成的图，它们都在一条路径上（长度为 2），等价于：

```
(a)-[]->()-[]->(b)
```

也可以指定长度范围，这样的边模式称为`variable-length edges`，例如：

```
(a)-[*3..5]->(b)
```

`*3..5`表示最小长度为 3，最大长度为 5。

该模式描述了 4 点 3 边、5 点 4 边或 6 点 5 边组成的图。

也可以忽略最小长度，只指定最大长度，例如：

```
(a)-[*..5]->(b)
```

> **Note**
>
> 必须指定最大长度，不支持仅指定最小长度（`(a)-[*3..]->(b)`）或都不指定（`(a)-[*]->(b)`）。

路径变量

一系列连接的点和边称为`路径`。nGQL 允许使用变量来命名路径，例如：

```
p = (a)-[*3..5]->(b)
```

可以在 MATCH 语句中使用路径变量。

最后更新: June 20, 2023

2022 vesoft Inc.

4.1.3 注释

本文介绍 nGQL 中的注释方式。

版本兼容性

- Nebula Graph 1.x 支持四种注释方式：`#`、`--`、`//`、`/* */`。
- Nebula Graph 2.x 中，`--` 不再是注释符。

Examples

```
nebula> # 这行什么都不做。
nebula> RETURN 1+1;     # 这条注释延续到行尾。
nebula> RETURN 1+1;     // 这条注释延续到行尾。
nebula> RETURN 1 /* 这是一条行内注释 */ + 1 == 2;
nebula> RETURN 11 +             \
/* 多行注释       \
用反斜线来换行。   \
*/ 12;
```

nGQL 语句中的反斜线（\）代表换行。

OpenCypher 兼容性

- 在 nGQL 中，用户必须在行末使用反斜线（\）来换行，即使是在使用 `/* */` 符号的多行注释内。
- 在 openCypher 中不需要使用反斜线换行。

```
/* openCypher 风格：
这条注释
延续了不止
一行 */
MATCH (n:label)
RETURN n;
```

```
/* 原生 nGQL 风格：  \
这条注释       \
延续了不止     \
一行 */       \
MATCH (n:tag) \
RETURN n;
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.1.4 大小写区分

标识符区分大小写

以下语句会出现错误，因为 `my_space` 和 `MY_SPACE` 是两个不同的图空间。

```
nebula> CREATE SPACE IF NOT EXISTS my_space (vid_type=FIXED_STRING(30));
nebula> use MY_SPACE;
[ERROR (-1005)]: SpaceNotFound:
```

关键字不区分大小写

以下语句是等价的，因为 `show` 和 `spaces` 是关键字。

```
nebula> show spaces;
nebula> SHOW SPACES;
nebula> SHOW spaces;
nebula> show SPACES;
```

函数不区分大小写

函数名称不区分大小写，例如 `count()` 、`COUNT()` 、`couNT()` 是等价的。

```
nebula> WITH [NULL, 1, 1, 2, 2] As a \
        UNWIND a AS b \
        RETURN count(b), COUNT(*), couNT(DISTINCT b);
+----------+----------+-------------------+
| count(b) | COUNT(*) | couNT(distinct b) |
+----------+----------+-------------------+
| 4        | 5        | 2                 |
+----------+----------+-------------------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.1.5 关键字

关键字在 nGQL 中有重要意义，分为保留关键字和非保留关键字。建议不要在 Schema 中使用关键字。

如果必须使用关键字：

- 非保留关键字作为标识符时可以不使用引号。
- 保留关键字或特殊字符作为标识符时，需要用反引号（`）包围，例如 `AND`。

> **Note**
>
> 关键字不区分大小写。

```
nebula> CREATE TAG TAG(name string);
[ERROR (-1004)]: SyntaxError: syntax error near `TAG'

nebula> CREATE TAG `TAG` (name string);
Execution succeeded

nebula> CREATE TAG SPACE(name string);
Execution succeeded

nebula> CREATE TAG 中文(简体 string);
Execution succeeded

nebula> CREATE TAG `￥%特殊 字符&*+-*/` (`q~! () =  wer` string);
Execution succeeded
```

保留关键字

```
ACROSS
ADD
ALTER
AND
AS
ASC
ASCENDING
BALANCE
BOOL
BY
CASE
CHANGE
COMPACT
CREATE
DATE
DATETIME
DELETE
DESC
DESCENDING
DESCRIBE
DISTINCT
DOUBLE
DOWNLOAD
DROP
EDGE
EDGES
EXISTS
EXPLAIN
FETCH
FIND
FIXED_STRING
FLOAT
FLUSH
FORMAT
FROM
GET
GO
GRANT
IF
IGNORE_EXISTED_INDEX
IN
INDEX
INDEXES
INGEST
INSERT
INT
INT16
INT32
INT64
```


2022 vesoft Inc.

```
INT8
INTERSECT
IS
LIMIT
LIST
LOOKUP
MAP
MATCH
MINUS
NO
NOT
NOT_IN
NULL
OF
OFFSET
ON
OR
ORDER
OVER
OVERWRITE
PROFILE
PROP
REBUILD
RECOVER
REMOVE
RESTART
RETURN
REVERSELY
REVOKE
SET
SHOW
STEP
STEPS
STOP
STRING
SUBMIT
TAG
TAGS
TIME
TIMESTAMP
TO
UNION
UPDATE
UPSERT
UPTO
USE
VERTEX
VERTICES
WHEN
WHERE
WITH
XOR
YIELD
```

非保留关键字

```
ACCOUNT
ADMIN
ALL
ANY
ATOMIC_EDGE
AUTO
BIDIRECT
BOTH
CHARSET
CLIENTS
COLLATE
COLLATION
COMMENT
CONFIGS
CONTAINS
DATA
DBA
DEFAULT
ELASTICSEARCH
ELSE
END
ENDS
ENDS_WITH
FORCE
FULLTEXT
FUZZY
GOD
GRAPH
GROUP
GROUPS
GUEST
HDFS
HOST
HOSTS
INTO
```


2022 vesoft Inc.

```
IS_EMPTY
IS_NOT_EMPTY
IS_NOT_NULL
IS_NULL
JOB
JOBS
KILL
LEADER
LISTENER
META
NOLOOP
NONE
NOT_CONTAINS
NOT_ENDS_WITH
NOT_STARTS_WITH
OPTIONAL
OUT
PART
PARTITION_NUM
PARTS
PASSWORD
PATH
PLAN
PREFIX
QUERIES
QUERY
REDUCE
REGEXP
REPLICA_FACTOR
RESET
ROLE
ROLES
SAMPLE
SEARCH
SERVICE
SESSION
SESSIONS
SHORTEST
SIGN
SINGLE
SKIP
SNAPSHOT
SNAPSHOTS
SPACE
SPACES
STARTS
STARTS_WITH
STATS
STATUS
STORAGE
SUBGRAPH
TEXT
TEXT_SEARCH
THEN
TOP
TTL_COL
TTL_DURATION
UNWIND
USER
USERS
UUID
VALUE
VALUES
VID_TYPE
WILDCARD
ZONE
ZONES
FALSE
TRUE
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.1.6 nGQL 风格指南

nGQL 没有严格的构建格式要求，但根据恰当而统一的风格创建 nGQL 语句有利于提高可读性、避免歧义。在同一组织或项目中使用相同的 nGQL 风格有利于降低维护成本，规避因格式混乱或误解造成的问题。本文为写作 nGQL 语句提供了风格参考。

Compatibility

nGQL 风格与 Cypher Style Guide 不同。

换行

1. 换行写子句。

   不推荐:

   ```
   GO FROM "player100" OVER follow REVERSELY YIELD src(edge) AS id;
   ```

   推荐:

   ```
   GO FROM "player100" \
   OVER follow REVERSELY \
   YIELD src(edge) AS id;
   ```

2. 换行写复合语句中的不同语句。

   不推荐:

   ```
   GO FROM "player100" OVER follow REVERSELY YIELD src(edge) AS id | GO FROM $-.id \
   OVER serve WHERE properties($^).age > 20 YIELD properties($^).name AS FriendOf, properties($$).name AS Team;
   ```

   推荐:

   ```
   GO FROM "player100" \
   OVER follow REVERSELY \
   YIELD src(edge) AS id | \
   GO FROM $-.id OVER serve \
   WHERE properties($^).age > 20 \
   YIELD properties($^).name AS FriendOf, properties($$).name AS Team;
   ```

3. 子句长度超过 80 个字符时，在合适的位置换行。

   不推荐:

   ```
   MATCH (v:player{name:"Tim Duncan"})-[e]->(v2) \
   WHERE (v2.player.name STARTS WITH "Y" AND v2.player.age > 35 AND v2.player.age < v.player.age) OR (v2.player.name STARTS WITH "T" AND v2.player.age < 45 AND v2.player.age > v.player.age) \
   RETURN v2;
   ```

   推荐:

   ```
   MATCH (v:player{name:"Tim Duncan"})-[e]->(v2) \
   WHERE (v2.player.name STARTS WITH "Y" AND v2.player.age > 35 AND v2.player.age < v.player.age) \
   OR (v2.player.name STARTS WITH "T" AND v2.player.age < 45 AND v2.player.age > v.player.age) \
   RETURN v2;
   ```

Note

即使子句不超过 80 个字符，如需换行后有助于理解，也可将子句再次分行。

2022 vesoft Inc.

标识符命名

在 nGQL 语句中，关键字、标点符号、空格以外的字符内容都是标识符。推荐的标识符命名方式如下。

1. 使用单数名词命名 Tag，用原型动词或动词短语构成 Edge type。

   不推荐：

   ```
   MATCH p=(v:players)-[e:are_following]-(v2) \
   RETURN nodes(p);
   ```

   推荐：

   ```
   MATCH p=(v:player)-[e:follow]-(v2) \
   RETURN nodes(p);
   ```

2. 标识符用蛇形命名法，以下划线（_）连接单词，且所有字母小写。

   不推荐：

   ```
   MATCH (v:basketballTeam) \
   RETURN v;
   ```

   推荐：

   ```
   MATCH (v:basketball_team) \
   RETURN v;
   ```

3. 语法关键词大写，变量小写。

   不推荐：

   ```
   match (V:player) return V limit 5;
   ```

   推荐：

   ```
   MATCH (v:player) RETURN v LIMIT 5;
   ```

Pattern

1. 分行写 Pattern 时，在表示边的箭头右侧换行，而不是左侧。

   不推荐：

   ```
   MATCH (v:player{name: "Tim Duncan", age: 42}) \
   -[e:follow]->()-[e2:serve]->()<--(v2) \
   RETURN v, e, v2;
   ```

   推荐：

   ```
   MATCH (v:player{name: "Tim Duncan", age: 42})-[e:follow]-> \
   ()-[e2:serve]->()<--(v2) \
   RETURN v, e, v2;
   ```

2. 将无需查询的点和边匿名化。

   不推荐：

   ```
   MATCH (v:player)-[e:follow]->(v2) \
   RETURN v;
   ```

   推荐：

   ```
   MATCH (v:player)-[:follow]->() \
   RETURN v;
   ```

3. 将非匿名点放在匿名点的前面。

   不推荐：

2022 vesoft Inc.

```
MATCH ()-[:follow]->(v) \
RETURN v;
```

推荐：

```
MATCH (v)<-[:follow]-() \
RETURN v;
```

字符串

字符串用双引号包围。

不推荐：

```
RETURN 'Hello Nebula!';
```

推荐：

```
RETURN "Hello Nebula!\"123\"";
```

> **Note**

字符串中需要嵌套单引号或双引号时，用反斜线（\）转义。例如：

```
RETURN "\"Nebula Graph is amazing,\" the user says.";
```

结束语句

1. 用英文分号（;）结束 nGQL 语句。

   不推荐：

   ```
   FETCH PROP ON player "player100" YIELD properties(vertex);
   ```

   推荐：

   ```
   FETCH PROP ON player "player100" YIELD properties(vertex);
   ```

2. 使用管道符（|）分隔的复合语句，仅在最后一行末用英文分号结尾。在管道符前使用英文分号会导致语句执行失败。

   不支持：

   ```
   GO FROM "player100" \
   OVER follow \
   YIELD dst(edge) AS id; | \
   GO FROM $-.id \
   OVER serve \
   YIELD properties($$).name AS Team, properties($^).name AS Player;
   ```

   支持：

   ```
   GO FROM "player100" \
   OVER follow \
   YIELD dst(edge) AS id | \
   GO FROM $-.id \
   OVER serve \
   YIELD properties($$).name AS Team, properties($^).name AS Player;
   ```

3. 在包含自定义变量的复合语句中，用英文分号结束定义变量的语句。不按规则加分号或使用管道符结束该语句会导致执行失败。

   不支持：

   ```
   $var = GO FROM "player100" \
   OVER follow \
   YIELD dst(edge) AS id \
   GO FROM $var.id \
   OVER serve \
   YIELD properties($$).name AS Team, properties($^).name AS Player;
   ```


2022 vesoft Inc.

也不支持:

```
$var = GO FROM "player100" \
OVER follow \
YIELD dst(edge) AS id | \
GO FROM $var.id \
OVER serve \
YIELD properties($$).name AS Team, properties($^).name AS Player;
```

支持:

```
$var = GO FROM "player100" \
OVER follow \
YIELD dst(edge) AS id; \
GO FROM $var.id \
OVER serve \
YIELD properties($$).name AS Team, properties($^).name AS Player;
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.2 数据类型

4.2.1 数值

nGQL 支持整数和浮点数。

整数

nGQL 支持带符号的 64 位整数（INT64）、32 位整数（INT32）、16 位整数（INT16）和 8 位整数（INT8）。

类型	声明关键字	范围
INT64	`INT64` 或 `INT`	-9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807
INT32	`INT32`	-2,147,483,648 ~ 2,147,483,647
INT16	`INT16`	-32,768 ~ 32,767
INT8	`INT8`	-128 ~ 127

浮点数

nGQL 支持单精度浮点（FLOAT）和双精度浮点（DOUBLE）。

类型	声明关键字	范围	精度
FLOAT	`FLOAT`	3.4E +/- 38	6~7 位
DOUBLE	`DOUBLE`	1.7E +/- 308	15~16 位

nGQL 支持科学计数法，例如 `1e2`、`1.1e2`、`.3e4`、`1.e4`、`-1234E-10`。

Note

不支持 MySQL 中的 DECIMAL 数据类型。

数值的读写

在写入和读取不同类型的数据时，nGQL 的行为遵守以下规则：

数值类型	设置为 **VID**	设置为属性类型	读取该类型的属性值得到的类型
INT64	支持	支持	INT64
INT32	不支持	支持	INT64
INT16	不支持	支持	INT64
INT8	不支持	支持	INT64
FLOAT	不支持	支持	DOUBLE
DOUBLE	不支持	支持	DOUBLE

2022 vesoft Inc.

例如，nGQL 不支持设置 INT8 类型的 VID，但支持将 TAG 或 Edge type 的某个属性类型设置为 INT8。当使用 nGQL 语句读取 INT8 类型的属性时，获取到的值的类型为 INT64。

- Nebula Graph 支持写入多种进制的数值：
- 十进制，例如 `123456`。
- 十六进制，例如 `0x1e240`。
- 八进制，例如 `0361100`。

  但 Nebula Graph 会将写入的非十进制数值解析为十进制的值保存。读取到的值为十进制。

  例如，属性 `score` 的类型为 `INT`，通过 INSERT 语句为其赋值 `0xb`，使用 FETCH 等语句查询该属性值获取到的结果是 `11`，即将十六进制的 `0xb` 转换为十进制后的值。

- 将 FLOAT/DOUBLE 类型的数值插入 INT 类型的列，会将数值四舍五入取整。

最后更新: June 20, 2023

2022 vesoft Inc.

4.2.2 布尔

Nebula Graph 使用关键字 `BOOL` 声明布尔数据类型，可选值为 `true` 或 `false` 。

nGQL 支持以如下方式使用布尔值:

- 将属性值的数据类型定义为布尔。
- 在 `WHERE` 子句中用布尔值作为判断条件。

最后更新: June 20, 2023

2022 vesoft Inc.

4.2.3 字符串

Nebula Graph 支持定长字符串和变长字符串。

声明与表示方式

nGQL 中的字符串声明方式如下：

- 使用关键字 `STRING` 声明变长字符串。
- 使用关键字 `FIXED_STRING(<length>)` 声明定长字符串，`<length>` 为字符串长度，例如 `FIXED_STRING(32)` 。

字符串的表示方式为用双引号或单引号包裹，例如 `"Hello, Cooper"` 或 `'Hello, Cooper'` 。

字符串读写

nGQL 支持以如下方式使用字符串：

- 将 VID 的数据类型定义为定长字符串。
- 将变长字符串设置为 Schema 名称，包括图空间、Tag、Edge type 和属性的名称。
- 将属性值的数据类型定义为定长或变长字符串。

例如：

- 将属性值的类型定义为定长字符串

```
nebula> CREATE TAG IF NOT EXISTS t1 (p1 FIXED_STRING(10));
```

- 将属性值的类型定义为变长字符串

```
nebula> CREATE TAG IF NOT EXISTS t2 (p2 STRING);
```

如果尝试写入的定长字符串超出长度限制：

- 当该定长字符串为属性值时，写入会成功，Nebula Graph 将截断字符串，仅存入符合长度限制的部分。
- 当该定长字符串为 VID 时，写入会失败，Nebula Graph 将报错。

转义字符

字符串中不支持直接换行，可以使用转义字符实现，例如：

- `"\n\t\r\b\f"`
- `"\110ello world"`

OpenCypher 兼容性

openCypher、Cypher 和 nGQL 之间有一些细微区别，例如下面 openCypher 的示例，不能将单引号替换为双引号。

```
# File: Literals.feature
Feature: Literals

Background:
    Given any graph
 Scenario: Return a single-quoted string
    When executing query:
      """
      RETURN '' AS literal
      """
    Then the result should be, in any order:
      | literal |
      | ''      |    # Note: it should return single-quotes as openCypher required.
    And no side effects
```


2022 vesoft Inc.

Cypher 的返回结果同时支持单引号和双引号，nGQL 遵循 Cypher 的方式。

```
nebula > YIELD '' AS quote1, "" AS quote2, "'" AS quote3, '"' AS quote4
+--------+--------+--------+--------+
| quote1 | quote2 | quote3 | quote4 |
+--------+--------+--------+--------+
| ""     | ""     | "'"    | """    |
+--------+--------+--------+--------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.2.4 日期和时间类型

本文介绍日期和时间的类型，包括 `DATE` 、 `TIME` 、 `DATETIME` 、 `TIMESTAMP` 和 `DURATION` 。

注意事项

- 在插入时间类型的属性值时，Nebula Graph 会根据配置文件中 `timezone_name` 参数指定的时区，将该 `DATE` 、 `TIME` 、 `DATETIME` 转换成相应的世界协调时间（UTC）时间。

  > **Note**
  >
  > 如需修改当前时区，请同时修改所有服务的配置文件中的 `timezone_name` 参数。

- 函数 `date()` 、 `time()` 和 `datetime()` 可以指定时区进行转换，例如 `datetime("2017-03-04 22:30:40.003000+08:00")` 或 `datetime("2017-03-04T22:30:40.003000[Asia/Shanghai]")` 。
- 函数 `date()` 、 `time()` 、 `datetime()` 和 `timestamp()` 可以用空值获取当前的日期或时间。
- 函数 `date()` 、 `time()` 、 `datetime()` 和 `duration()` 可以用属性名称获取自身的某一个具体属性值，例如 `date().month` 获取当前月份、 `time("02:59:40").minute` 获取传入时间的分钟数。

> **Note**
>
> 设置时间的年份为负数时，需要使用 Map 类型数据。

openCypher 兼容性

- 支持年、月、日、时、分、秒、毫秒、微秒，不支持纳秒。
- 不支持函数 `localdatetime()` 。
- 不支持大部分字符串时间格式，支持 `YYYY-MM-DDThh:mm:ss` 和 `YYYY-MM-DD hh:mm:ss` 。
- 支持单个数字的字符串时间格式，例如 `time("1:1:1")` 。

DATE

`DATE` 包含日期，但是不包含时间。Nebula Graph 检索和显示 `DATE` 的格式为 `YYYY-MM-DD` 。支持的范围是 `-32768-01-01` 到 `32767-12-31` 。

`date()` 支持的属性名称包括 `year` 、 `month` 和 `day` 。 `date()` 支持输入 `YYYY` 、 `YYYY-MM` 或 `YYYY-MM-DD` ，未输入的月份或日期默认为 `01` 。

```
nebula> RETURN DATE({year:-123, month:12, day:3});
+-----------------------------------+
| date({year:-(123),month:12,day:3}) |
+-----------------------------------+
| -123-12-03                        |
+-----------------------------------+
nebula> RETURN DATE("23333");
+---------------+
| date("23333") |
+---------------+
| 23333-01-01   |
+---------------+
```

TIME

`TIME` 包含时间，但是不包含日期。Nebula Graph 检索和显示 `TIME` 的格式为 `hh:mm:ss.msmsmsususus` 。支持的范围是 `00:00:00.000000` 到 `23:59:59.999999` 。

`time()` 支持的属性名称包括 `hour` 、 `minute` 和 `second` 。

2022 vesoft Inc.

DATETIME

`DATETIME` 包含日期和时间。Nebula Graph 检索和显示 `DATETIME` 的格式为 `YYYY-MM-DDThh:mm:ss.msmsmsususus` 。支持的范围是 `-32768-01-01T00:00:00.000000` 到 `32767-12-31T23:59:59.999999` 。

`datetime()` 支持的属性名称包括 `year` 、 `month` 、 `day` 、 `hour` 、 `minute` 和 `second` 。

TIMESTAMP

`TIMESTAMP` 包含日期和时间。支持的范围是 UTC 时间的 `1970-01-01T00:00:01` 到 `2262-04-11T23:47:16` 。

`TIMESTAMP` 还有以下特点：

- 以时间戳形式存储和显示。例如 `1615974839` ，表示 `2021-03-17T17:53:59` 。
- 查询 `TIMESTAMP` 的方式包括时间戳和 `timestamp()` 函数。
- 插入 `TIMESTAMP` 的方式包括时间戳、 `timestamp()` 函数和 `now()` 函数。
- `timestamp()` 函数支持传入空值获取当前时区的时间戳，还接受 string 类型的参数。

```
# 传入当前时间。
nebula> RETURN timestamp();
+-------------+
| timestamp() |
+-------------+
| 1625469277  |
+-------------+

# 传入指定时间。
nebula> RETURN timestamp("2022-01-05T06:18:43");
+-----------------------------------+
| timestamp("2022-01-05T06:18:43") |
+-----------------------------------+
| 1641363523                        |
+-----------------------------------+
```

> 旧版本兼容性
>
> 在 Nebula Graph 3.0.0 版本前，传入 timestamp() 函数的时间字符串可包含毫秒和微秒；从 3.0.0 版本起，传入 timestamp() 函数的时间字符串不支持包含毫秒和微秒。

- 底层存储的数据格式为 **64** 位 **int**。

DURATION

`DURATION` 是一段连续的时间，由 `years` 、 `months` 、 `days` 、 `hours` 、 `minutes` 、 `seconds` 六个Key自由组合成的Map类型数据表示。例如 `duration({years: 12, months: 5, days: 14, hours: 16, minutes: 12, seconds: 70})` 。

`DURATION` 还有以下特点：

- 不支持为 `DURATION` 类型数据创建索引。
- 可以用于对指定时间进行计算。

示例

1. 创建 Tag，名称为 `date1` ，包含 `DATE` 、 `TIME` 和 `DATETIME` 三种类型。

```
nebula> CREATE TAG IF NOT EXISTS date1(p1 date, p2 time, p3 datetime);
```

2. 插入点，名称为 `test1` 。

```
nebula> INSERT VERTEX date1(p1, p2, p3) VALUES "test1":(date("2021-03-17"), time("17:53:59"), datetime("2017-03-04T22:30:40.003000[Asia/Shanghai]"));
```


2022 vesoft Inc.

3. 获取 `test1` 的属性 `p1` 的月份。

```
nebula> CREATE TAG INDEX IF NOT EXISTS date1_index ON date1(p1);
nebula> REBUILD TAG INDEX date1_index;
nebula> MATCH (v:date1) RETURN v.date1.p1.month;
+-----------------+
| v.date1.p1.month |
+-----------------+
| 3               |
+-----------------+
```

4. 创建 Tag，名称为 `school`，包含 `TIMESTAMP` 类型。

```
nebula> CREATE TAG IF NOT EXISTS school(name string , found_time timestamp);
```

5. 插入点，名称为 `DUT`，存储时间为 `"1988-03-01T08:00:00"`。

```
# 时间戳形式插入，1988-03-01T08:00:00 对应的时间戳为 573177600，转换为 UTC 时间为 573206400。
nebula> INSERT VERTEX school(name, found_time) VALUES "DUT":("DUT", 573206400);

# 日期和时间格式插入。
nebula> INSERT VERTEX school(name, found_time) VALUES "DUT":("DUT", timestamp("1988-03-01T08:00:00"));
```

6. 插入点，名称为 `dut`，用 `now()` 或 `timestamp()` 函数存储时间。

```
# 用 now() 函数存储时间
nebula> INSERT VERTEX school(name, found_time) VALUES "dut":("dut", now());

# 用 timestamp() 函数存储时间
nebula> INSERT VERTEX school(name, found_time) VALUES "dut":("dut", timestamp());
```

还可以使用 `WITH` 语句设置具体日期时间或进行计算，例如:

```
nebula> WITH time({hour: 12, minute: 31, second: 14, millisecond:111, microsecond: 222}) AS d RETURN d;
+-----------------+
| d               |
+-----------------+
| 12:31:14.111222 |
+-----------------+

nebula> WITH date({year: 1984, month: 10, day: 11}) AS x RETURN x + 1;
+------------+
| (x+1)      |
+------------+
| 1984-10-12 |
+------------+

nebula> WITH date('1984-10-11') as x, duration({years: 12, days: 14, hours: 99, minutes: 12}) as d \
        RETURN x + d AS sum, x - d AS diff;
+------------+------------+
| sum        | diff       |
+------------+------------+
| 1996-10-29 | 1972-09-23 |
+------------+------------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.2.5 NULL

默认情况下，插入点或边时，属性值可以为 `NULL`，用户也可以设置属性值不允许为 `NULL`（`NOT NULL`），即插入点或边时必须设置该属性的值，除非创建属性时已经设置默认值。

NULL 的逻辑操作

`AND`、`OR`、`XOR` 和 `NOT` 的真值表如下。

a	b	a AND b	a OR b	a XOR b	NOT a
false	false	false	false	false	true
false	null	false	null	null	true
false	true	false	true	true	true
true	false	false	true	true	false
true	null	null	true	null	false
true	true	true	true	false	false
null	false	false	null	null	null
null	null	null	null	null	null
null	true	null	true	null	null

OpenCypher 兼容性

Nebula Graph 中，NULL 的比较和操作与 openCypher 不同，后续也可能会有变化。

NULL 的比较

Nebula Graph 中，NULL 的比较操作不兼容 openCypher。

NULL 的操作和返回

Nebula Graph 中，对 NULL 的操作以及返回结果不兼容 openCypher。

示例

使用 NOT NULL

创建 Tag，名称为 `player`，指定属性 `name` 为 `NOT NULL`。

```
nebula> CREATE TAG IF NOT EXISTS player(name string NOT NULL, age int);
```

使用 `SHOW` 命令查看创建 Tag 语句，属性 `name` 为 `NOT NULL`，属性 `age` 为默认的 `NULL`。

```
nebula> SHOW CREATE TAG player;
+-----------+-----------------------------------+
| Tag       | Create Tag                        |
+-----------+-----------------------------------+
| "student" | "CREATE TAG `player` (            |
|           |  `name` string NOT NULL,          |
|           |  `age` int64 NULL                 |
|           | ) ttl_duration = 0, ttl_col = """ |
+-----------+-----------------------------------+
```

插入点 `Kobe`，属性 `age` 可以为 `NULL`。

```
nebula> INSERT VERTEX player(name, age) VALUES "Kobe":("Kobe",null);
```


2022 vesoft Inc.

使用 **NOT NULL** 并设置默认值

创建 Tag，名称为 `player`，指定属性 `age` 为 `NOT NULL`，并设置默认值 `18`。

```
nebula> CREATE TAG IF NOT EXISTS player(name string, age int NOT NULL DEFAULT 18);
```

插入点 `Kobe`，只设置属性 `name`。

```
nebula> INSERT VERTEX player(name) VALUES "Kobe":("Kobe");
```

查询点 `Kobe`，属性 `age` 为默认值 `18`。

```
nebula> FETCH PROP ON player "Kobe" YIELD properties(vertex);
+--------------------------+
| properties(VERTEX)       |
+--------------------------+
| {age: 18, name: "Kobe"}  |
+--------------------------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.2.6 列表

列表（List）是复合数据类型，一个列表是一组元素的序列，可以通过元素在序列中的位置访问列表中的元素。

列表用左方括号（[）和右方括号（]）包裹多个元素，各个元素之间用英文逗号（,）隔开。元素前后的空格在列表中被忽略，因此可以使用换行符、制表符和空格调整格式。

OpenCypher 兼容性

复合数据类型（例如 List、Set、Map）不能存储为点或边的属性。

列表操作

对列表进行操作可以使用预设的列表函数，也可以使用下标表达式过滤列表内的元素。

下标表达式语法

```
[M]
[M..N]
[M..]
[..N]
```

nGQL 的下标支持从前往后查询，从 0 开始，0 表示第一个元素，1 表示第二个元素，以此类推；也支持从后往前查询，从-1 开始，-1 表示最后一个元素，-2 表示倒数第二个元素，以此类推。

- [M]：表示下标为 M 的元素。
- [M..N]：表示 `M ≤ 下标 < N` 的元素。`N` 为 0 时，返回为空。
- [M..]：表示 `M ≤ 下标` 的元素。
- [..N]：表示 `下标 < N` 的元素。`N` 为 0 时，返回为空。

> **Note**

- 越界的下标返回为空，未越界的可以正常返回。
- `M ≥ N` 时，返回为空。
- 查询单个元素时，如果 `M` 为 null，返回报错 `BAD_TYPE`；范围查询时，`M` 或 `N` 为 null，返回为 `null`。

示例

```
# 返回列表 [1,2,3]
nebula> RETURN list[1, 2, 3] AS a;
+-----------+
| a         |
+-----------+
| [1, 2, 3] |
+-----------+

# 返回列表 [1,2,3,4,5] 中位置下标为 3 的元素。列表的位置下标是从 0 开始，因此返回的元素为 4。
nebula> RETURN range(1,5)[3];
+---------------+
| range(1,5)[3] |
+---------------+
| 4             |
+---------------+

# 返回列表 [1,2,3,4,5] 中位置下标为-2 的元素。列表的最后一个元素的位置下标是-1，因此-2 是指倒数第二个元素，即 4。
nebula> RETURN range(1,5)[-2];
+------------------+
| range(1,5)[-(2)] |
+------------------+
| 4                |
+------------------+

# 返回列表 [1,2,3,4,5] 中下标位置从 0 到 3（不包括 3）的元素。
nebula> RETURN range(1,5)[0..3];
+------------------+
| range(1,5)[0..3] |
+------------------+
```


2022 vesoft Inc.

```
| [1, 2, 3]       |
+-----------------+

# 返回列表 [1,2,3,4,5] 中位置下标大于 2 的元素。
nebula> RETURN range(1,5)[3..] AS a;
+--------+
| a      |
+--------+
| [4, 5] |
+--------+

# 返回列表内下标小于 3 的元素。
nebula> WITH list[1, 2, 3, 4, 5] AS a \
        RETURN a[..3] AS r;
+-----------+
| r         |
+-----------+
| [1, 2, 3] |
+-----------+

# 筛选列表 [1,2,3,4,5] 中大于 2 的元素，将这些元素分别做运算并返回。
nebula> RETURN [n IN range(1,5) WHERE n > 2 | n + 10] AS a;
+--------------+
| a            |
+--------------+
| [13, 14, 15] |
+--------------+

# 返回列表内第一个至倒数第二个（包括）的元素。
nebula> YIELD list[1, 2, 3][0..-1] AS a;
+--------+
| a      |
+--------+
| [1, 2] |
+--------+

# 返回列表内倒数第三个至倒数第一个（不包括）的元素。
nebula> YIELD list[1, 2, 3, 4, 5][-3..-1] AS a;
+--------+
| a      |
+--------+
| [3, 4] |
+--------+

# 设置变量，返回列表内下标为 1、2 的元素。
nebula> $var = YIELD 1 AS f, 3 AS t; \
        YIELD list[1, 2, 3][$var.f..$var.t] AS a;
+--------+
| a      |
+--------+
| [2, 3] |
+--------+

# 越界的下标返回为空，未越界的可以正常返回。
nebula> RETURN list[1, 2, 3, 4, 5] [0..10] AS a;
+-----------------+
| a               |
+-----------------+
| [1, 2, 3, 4, 5] |
+-----------------+

nebula> RETURN list[1, 2, 3] [-5..5] AS a;
+-----------+
| a         |
+-----------+
| [1, 2, 3] |
+-----------+

# [0..0] 时返回为空。
nebula> RETURN list[1, 2, 3, 4, 5] [0..0] AS a;
+----+
| a  |
+----+
| [] |
+----+

# M ≥ N 时，返回为空。
nebula> RETURN list[1, 2, 3, 4, 5] [3..1] AS a;
+----+
| a  |
+----+
| [] |
+----+

# 范围查询时，下标有 null 时，返回为 null。
nebula> WITH list[1,2,3] AS a \
        RETURN a[0..null] as r;
+----------+
| r        |
+----------+
| __NULL__ |
+----------+

# 将列表 [1,2,3,4,5] 中的元素分别做运算，然后将列表去掉表头并返回。
```


2022 vesoft Inc.

```
nebula> RETURN tail([n IN range(1, 5) | 2 * n - 10]) AS a;
+-----------------+
| a               |
+-----------------+
| [-6, -4, -2, 0] |
+-----------------+

# 将列表 [1,2,3] 中的元素判断为真，然后返回。
nebula> RETURN [n IN range(1, 3) WHERE true | n] AS r;
+-----------+
| r         |
+-----------+
| [1, 2, 3] |
+-----------+

# 返回列表 [1,2,3] 的长度。
nebula> RETURN size(list[1,2,3]);
+---------------+
| size([1,2,3]) |
+---------------+
| 3             |
+---------------+

# 将列表 [92,90] 中的元素做运算，然后在 where 子句中进行条件判断。
nebula> GO FROM "player100" OVER follow WHERE properties(edge).degree NOT IN [x IN [92, 90] | x + $$.player.age] \
        YIELD dst(edge) AS id, properties(edge).degree AS degree;
+-------------+--------+
| id          | degree |
+-------------+--------+
| "player101" | 95     |
| "player102" | 90     |
+-------------+--------+

# 将 MATCH 语句的查询结果作为列表中的元素进行运算并返回。
nebula> MATCH p = (n:player{name:"Tim Duncan"})-[:follow]->(m) \
        RETURN [n IN nodes(p) | n.age + 100] AS r;
+------------+
| r          |
+------------+
| [142, 136] |
| [142, 133] |
+------------+
```

OpenCypher 兼容性

- 在 openCypher 中，查询越界元素时返回 `null`，而在 nGQL 中，查询单个越界元素时返回 `OUT_OF_RANGE`。

  ```
  nebula> RETURN range(0,5)[-12];
  +-------------------+
  | range(0,5)[-(12)] |
  +-------------------+
  | OUT_OF_RANGE      |
  +-------------------+
  ```

- 复合数据类型（例如 set、map、list）不能存储为点或边的属性。
- 建议修改图建模方式：将复合数据类型建模为点的邻边，而不是该点的自身属性，每条邻边可以动态增删，并且可以设置邻边的 Rank 值来控制邻边的顺序。
- List 中不支持 pattern，例如 `[(src)-[]->(m) | m.name]`。

最后更新: June 20, 2023

2022 vesoft Inc.

4.2.7 集合

集合（Set）是复合数据类型，集合中是一组元素，与列表（List）不同的是，集合中的元素是无序的，且不允许重复。

集合用左花括号（{）和右花括号（}）包裹多个元素，各个元素之间用英文逗号（,）隔开。元素前后的空格在集合中被忽略，因此可以使用换行符、制表符和空格调整格式。

OpenCypher 兼容性

- 复合数据类型（例如 List、Set、Map）不能存储为点或边的属性。
- 在 OpenCypher 中，集合不是一个数据类型，而在 nGQL 中，用户可以使用集合。

示例

```
# 返回集合 {1,2,3}。
nebula> RETURN set{1, 2, 3} AS a;
+-----------+
| a         |
+-----------+
| {3, 2, 1} |
+-----------+

# 返回集合 {1,2,1}，因为集合不允许重复元素，会返回 {1,2}，且顺序是无序的。
nebula> RETURN set{1, 2, 1} AS a;
+--------+
| a      |
+--------+
| {2, 1} |
+--------+

# 判断集合中是否有指定元素 1。
nebula> RETURN 1 IN set{1, 2} AS a;
+------+
| a    |
+------+
| true |
+------+

# 计算集合中的元素数量。
nebula> YIELD size(set{1, 2, 1}) AS a;
+---+
| a |
+---+
| 2 |
+---+

# 返回目标点属性值组成的集合。
nebula> GO FROM "player100" OVER follow \
        YIELD set{properties($$).name,properties($$).age} as a;
+-----------------------+
| a                     |
+-----------------------+
| {36, "Tony Parker"}   |
| {41, "Manu Ginobili"} |
+-----------------------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.2.8 映射

映射（Map）是复合数据类型。一个映射是一组键值对（Key-Value）的无序集合。在映射中，Key 是字符串类型，Value 可以是任何数据类型。用户可以通过 `map['<key>']` 的方法获取映射中的元素。

映射用左花括号（{）和右花括号（}）包裹多个键值对，各个键值对之间用英文逗号（,）隔开。键值对前后的空格在映射中被忽略，因此可以使用换行符、制表符和空格调整格式。

OpenCypher 兼容性

- 复合数据类型（例如 List、Set、Map）不能存储为点或边的属性。
- 不支持映射投影（map projection）。

示例

```
# 返回简单的映射。
nebula> YIELD map{key1: 'Value1', Key2: 'Value2'} as a;
+----------------------------------+
| a                                |
+----------------------------------+
| {Key2: "Value2", key1: "Value1"} |
+----------------------------------+

# 返回列表类型的映射。
nebula> YIELD map{listKey: [{inner: 'Map1'}, {inner: 'Map2'}]} as a;
+-----------------------------------------------+
| a                                             |
+-----------------------------------------------+
| {listKey: [{inner: "Map1"}, {inner: "Map2"}]} |
+-----------------------------------------------+

# 返回混合类型的映射。
nebula> RETURN map{a: LIST[1,2], b: SET{1,2,1}, c: "hee"} as a;
+----------------------------------+
| a                                |
+----------------------------------+
| {a: [1, 2], b: {2, 1}, c: "hee"} |
+----------------------------------+

# 返回映射中的指定元素。
nebula> RETURN map{a: LIST[1,2], b: SET{1,2,1}, c: "hee"}["b"] AS b;
+--------+
| b      |
+--------+
| {2, 1} |
+--------+

# 判断映射中是否有指定key，暂不支持判断value。
nebula> RETURN "a" IN MAP{a:1, b:2} AS a;
+------+
| a    |
+------+
| true |
+------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.2.9 类型转换

类型转换是指将表达式的类型转换为另一个类型。

类型强制转换函数

函数	说明
toBoolean()	将字符串转换为布尔。
toFloat()	将整数或字符串转换为浮点数。
toInteger()	将浮点或字符串转换为整数。
toString()	将非复合类型的数据，例如数字、布尔值等，转换为字符串。
toSet()	将列表或集合转换为集合。
type()	返回字符串格式的关系类型。

示例

```
nebula> UNWIND [true, false, 'true', 'false', NULL] AS b \
        RETURN toBoolean(b) AS b;
+---------+
| b       |
+---------+
| true    |
+---------+
| false   |
+---------+
| true    |
+---------+
| false   |
+---------+
| __NULL__ |
+---------+

nebula> RETURN toFloat(1), toFloat('1.3'), toFloat('1e3'), toFloat('not a number');
+-----------+---------------+---------------+-------------------------+
| toFloat(1) | toFloat("1.3") | toFloat("1e3") | toFloat("not a number") |
+-----------+---------------+---------------+-------------------------+
| 1.0       | 1.3           | 1000.0        | __NULL__                |
+-----------+---------------+---------------+-------------------------+

nebula> RETURN toInteger(1), toInteger('1'), toInteger('1e3'), toInteger('not a number');
+-------------+--------------+-----------------+---------------------------+
| toInteger(1) | toInteger("1") | toInteger("1e3") | toInteger("not a number") |
+-------------+--------------+-----------------+---------------------------+
| 1           | 1            | 1000            | __NULL__                  |
+-------------+--------------+-----------------+---------------------------+

nebula> MATCH (a:player)-[e]-() \
        RETURN type(e);
+---------+
| type(e)  |
+---------+
| "follow" |
+---------+
| "follow" |

nebula> MATCH (a:player {name: "Tim Duncan"}) \
        WHERE toInteger(right(id(a),3)) == 100 \
        RETURN a;
+-------------------------------------------------+
| a                                               |
+-------------------------------------------------+
| ("player100" :player{age: 42, name: "Tim Duncan"}) |
+-------------------------------------------------+

nebula> MATCH (n:player) \
        WITH n LIMIT toInteger(ceil(1.8)) \
        RETURN count(*) AS count;
+-------+
| count |
+-------+
| 2     |
+-------+

nebula> RETURN toString(9669) AS int2str, toString(null) AS null2str;
+---------+----------+
```


2022 vesoft Inc.

```
| int2str | null2str |
+---------+----------+
| "9669"  | __NULL__ |
+---------+----------+

nebula> RETURN toSet(list[1,2,3,1,2]) AS list2set;
+-----------+
| list2set  |
+-----------+
| {3, 1, 2} |
+-----------+

nebula> RETURN toSet(set{1,2,3,1,2}) AS set2set;
+-----------+
| set2set   |
+-----------+
| {3, 2, 1} |
+-----------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.2.10 地理位置

地理位置（GEOGRAPHY）是由经纬度构成的表示地理空间信息的数据类型。Nebula Graph 当前支持简单地理要素中的 Point、LineString 和 Polygon 三种地理形状。支持 SQL-MM 3 中的部分核心 geo 解析、构造、格式设置、转换、谓词和度量等函数。

GEOGRAPHY

GEOGRAPHY 的基本类型是点，由经纬度确定一个点，例如 `"POINT(3 8)"` 表示经度为 `3°`，纬度为 `8°`。多个点可以构成线段或多边形。

类型	示例	说明
Point	`"POINT(3 8)"`	点类型
LineString	`"LINESTRING(3 8, 4.7 73.23)"`	线段类型
Polygon	`"POLYGON((0 1, 1 2, 2 3, 0 1))"`	多边形类型

示例

geo 相关函数请参见 geo 函数。

```
//创建 Tag，允许存储任意形状地理位置数据类型。
nebula> CREATE TAG IF NOT EXISTS any_shape(geo geography);

//创建 Tag，只允许存储点形状地理位置数据类型。
nebula> CREATE TAG IF NOT EXISTS only_point(geo geography(point));

//创建 Tag，只允许存储线段形状地理位置数据类型。
nebula> CREATE TAG IF NOT EXISTS only_linestring(geo geography(linestring));

//创建 Tag，只允许存储多边形形状地理位置数据类型。
nebula> CREATE TAG IF NOT EXISTS only_polygon(geo geography(polygon));

//创建 Edge type，允许存储任意形状地理位置数据类型。
nebula> CREATE EDGE IF NOT EXISTS any_shape_edge(geo geography);

//创建存储多边形地理位置的点。
nebula> INSERT VERTEX any_shape(geo) VALUES "103":(ST_GeogFromText("POLYGON((0 1, 1 2, 2 3, 0 1))"));

//创建存储多边形地理位置的边。
nebula> INSERT EDGE any_shape_edge(geo) VALUES "201"->"302":(ST_GeogFromText("POLYGON((0 1, 1 2, 2 3, 0 1))"));

//查询点 103 的属性 geo。
nebula> FETCH PROP ON any_shape "103" YIELD ST_ASText(any_shape.geo);
+---------------------------------+
| ST_ASText(any_shape.geo)        |
+---------------------------------+
| "POLYGON((0 1, 1 2, 2 3, 0 1))" |
+---------------------------------+

//查询边 201->302 的属性 geo。
nebula> FETCH PROP ON any_shape_edge "201"->"302" YIELD ST_ASText(any_shape_edge.geo);
+---------------------------------+
| ST_ASText(any_shape_edge.geo)   |
+---------------------------------+
| "POLYGON((0 1, 1 2, 2 3, 0 1))" |
+---------------------------------+

//为 geo 属性创建索引并使用 LOOKUP 查询。
nebula> CREATE TAG INDEX IF NOT EXISTS any_shape_geo_index ON any_shape(geo);
nebula> REBUILD TAG INDEX any_shape_geo_index;
nebula> LOOKUP ON any_shape YIELD ST_ASText(any_shape.geo);
+---------------------------------+
| ST_ASText(any_shape.geo)        |
+---------------------------------+
| "POLYGON((0 1, 1 2, 2 3, 0 1))" |
+---------------------------------+
```


2022 vesoft Inc.

为 geo 属性创建索引时，还可以指定 geo 索引的参数。说明如下。

参数	默认值	说明
`s2_max_level`	`30`	S2 cell 用于填充的最大等级。取值：`1`～`30`。设置为小于默认值时，意味着会使用较大的单元格进行填充。
`s2_max_cells`	`8`	S2 cell 用于填充的最大数量，可以限制填充时的工作量。取值：`1`～`30`。对于复杂形状的区域（例如细矩形），可以使用更大的值。

Note

指定如上两个参数对 Point 类型属性没有影响，Point 类型属性的 `s2_max_level` 强制为 `30`。

```
nebula> CREATE TAG INDEX IF NOT EXISTS any_shape_geo_index ON any_shape(geo) with (s2_max_level=30, s2_max_cells=8);
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.3 变量和复合查询

4.3.1 复合查询（子句结构）

复合查询将来自不同请求的数据放在一起，然后进行过滤、分组或者排序等，最后返回结果。

Nebula Graph 支持三种方式进行复合查询（或子查询）：

- （opencypher 兼容语句）连接各个子句，让它们在彼此之间提供中间结果集。
- （原生 nGQL）多个查询可以合并处理，以英文分号（;）分隔，返回最后一个查询的结果。
- （原生 nGQL）可以用管道符（|）将多个查询连接起来，上一个查询的结果可以作为下一个查询的输入。

OpenCypher 兼容性

在复合查询中，请不要混用 opencypher 兼容语句和原生 nGQL 语句，例如 `MATCH ... | GO ... | YIELD ...`，混用两种语句，行为是未定义的。

- 如果使用 openCypher 兼容语句（`MATCH`、`RETURN`、`WITH` 等），请不要使用管道符或分号组合子句。
- 如果使用原生 nGQL 语句（`FETCH`、`GO`、`LOOKUP` 等），必须使用管道符或分号组合子句。

行为未定义

不要混用 openCypher 兼容语句和原生 nGQL 语句，行为是未定义的。

复合查询不支持事务

例如一个查询由三个子查询 A、B、C 组成，A 是一个读操作，B 是一个计算操作，C 是一个写操作，如果在执行过程中，任何一个操作执行失败，则整个结果是未定义的：没有回滚，而且写入的内容取决于执行程序。

Note

openCypher 没有事务要求。

示例

- opencypher 兼容语句

```
# 子句连接多个查询。
nebula> MATCH p=(v:player{name:"Tim Duncan"})--() \
        WITH nodes(p) AS n \
```


2022 vesoft Inc.

```
        UNWIND n AS n1 \
        RETURN DISTINCT n1;
```

- 原生 nGQL（分号）

```
# 只返回边。
nebula> SHOW TAGS; SHOW EDGES;

# 插入多个点。
nebula> INSERT VERTEX player(name, age) VALUES "player100":("Tim Duncan", 42); \
        INSERT VERTEX player(name, age) VALUES "player101":("Tony Parker", 36); \
        INSERT VERTEX player(name, age) VALUES "player102":("LaMarcus Aldridge", 33);
```

- 原生 nGQL（管道符）

```
# 管道符连接多个查询。
nebula> GO FROM "player100" OVER follow YIELD dst(edge) AS id | \
        GO FROM $-.id OVER serve YIELD properties($$).name AS Team, \
        properties($^).name AS Player;
+-----------+-----------------+
| Team      | Player          |
+-----------+-----------------+
| "Spurs"   | "Tony Parker"   |
| "Hornets" | "Tony Parker"   |
| "Spurs"   | "Manu Ginobili" |
+-----------+-----------------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.3.2 自定义变量

Nebula Graph 允许将一条语句的结果作为自定义变量传递给另一条语句。

OpenCypher 兼容性

当引用一个变量的点、边或路径，需要先给它命名。例如：

```
nebula> MATCH (v:player{name:"Tim Duncan"}) RETURN v;
+-------------------------------------------------+
| v                                               |
+-------------------------------------------------+
| ("player100" :player{name: "Tim Duncan", age: 42}) |
+-------------------------------------------------+
```

示例中的 `v` 就是自定义变量。

原生 nGQL

nGQL 扩展的自定义变量可以表示为 `$var_name`， `var_name` 由字母、数字或下划线（_）构成，不允许使用其他字符。

自定义变量仅在当前执行（本复合查询中）有效，执行结束后变量也会释放，不能在其他客户端、执行、session 中使用之前的自定义变量。

用户可以在复合查询中使用自定义变量。复合查询的详细信息请参见复合查询。

> **Note**
>
> 自定义变量区分大小写。

示例

```
nebula> $var = GO FROM "player100" OVER follow YIELD dst(edge) AS id; \
        GO FROM $var.id OVER serve YIELD properties($$).name AS Team, \
        properties($^).name AS Player;
+-----------+-----------------+
| Team      | Player          |
+-----------+-----------------+
| "Spurs"   | "Tony Parker"   |
| "Hornets" | "Tony Parker"   |
| "Spurs"   | "Manu Ginobili" |
+-----------+-----------------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.3.3 引用属性

用户可以在 `WHERE` 和 `YIELD` 子句中引用点或边的属性。

> **Note**
>
> 本功能仅适用于原生 nGQL 的 GO 语句。

引用点的属性

起始点

```
$^.<tag_name>.<prop_name>
```

参数	说明
`$^`	起始点
`tag_name`	点的 Tag 名称
`prop_name`	Tag 内的属性名称

目的点

```
$$.<tag_name>.<prop_name>
```

参数	说明
`$$`	目的点
`tag_name`	点的 Tag 名称
`prop_name`	Tag 内的属性名称

引用边的属性

引用自定义的边属性

```
<edge_type>.<prop_name>
```

参数	说明
`edge_type`	Edge type
`prop_name`	Edge type 的属性名称

引用内置的边属性

除了自定义的边属性，每条边还有如下四种内置属性：

参数	说明
`_src`	边的起始点
`_dst`	边的目的点
`_type`	边的类型内部编码，正负号表示方向：正数为正向边，负数为逆向边
`_rank`	边的 rank 值

示例

2022 vesoft Inc.

```
# 返回起始点的 Tag player 的 name 属性值和目的点的 Tag player 的 age 属性值。
nebula> GO FROM "player100" OVER follow YIELD $^.player.name AS startName, $$.player.age AS endAge;
+--------------+--------+
| startName    | endAge |
+--------------+--------+
| "Tim Duncan" | 36     |
| "Tim Duncan" | 41     |
+--------------+--------+

# 返回 Edge type follow 的 degree 属性值。
nebula> GO FROM "player100" OVER follow YIELD follow.degree;
+---------------+
| follow.degree |
+---------------+
| 95            |
+---------------+

# 返回 EdgeType 是 follow 的起始点 VID、目的点 VID、EdgeType 编码（正数为正向边，负数为逆向边），和边的 rank 值。
nebula> GO FROM "player100" OVER follow YIELD follow._src, follow._dst, follow._type, follow._rank;
+-------------+-------------+--------------+--------------+
| follow._src | follow._dst | follow._type | follow._rank |
+-------------+-------------+--------------+--------------+
| "player100" | "player101" | 17           | 0            |
| "player100" | "player125" | 17           | 0            |
+-------------+-------------+--------------+--------------+
```

!!! 版本兼容性

从 Nebula Graph 2.6.0 起支持了新的 Schema 函数。以上示例在 Nebula Graph 3.1.0 中的近似写法如下：

```
GO FROM "player100" OVER follow YIELD properties($^).name AS startName, properties($$).age AS endAge;
GO FROM "player100" OVER follow YIELD properties(edge).degree;
GO FROM "player100" OVER follow YIELD src(edge), dst(edge), type(edge), rank(edge);
```

Nebula Graph 3.1.0 兼容新旧语法。

最后更新: June 20, 2023

2022 vesoft Inc.

4.4 运算符

4.4.1 比较符

Nebula Graph 支持的比较符如下。

符号	说明
`=`	赋值
`+`	加法
`-`	减法
`*`	乘法
`/`	除法
`==`	相等
`!=`, `<>`	不等于
`>`	大于
`>=`	大于等于
`<`	小于
`<=`	小于等于
`%`	取模
`-`	负数符号
`IS NULL`	为 NULL
`IS NOT NULL`	不为 NULL
`IS EMPTY`	不存在
`IS NOT EMPTY`	存在

比较操作的结果是 `true` 或者 `false`。

Note

- 比较不同类型的值通常没有定义，结果可能是 `NULL` 或其它。
- `EMPTY` 当前仅用于判断，不支持函数或者运算操作，包括且不限于 `GROUP BY` 、`count()` 、`sum()` 、`max()` 、`hash()` 、`collect()` 、`+` 、`*`。

OpenCypher 兼容性

openCypher 中没有 `EMPTY`，因此不支持在 MATCH 语句中使用 `EMPTY`。

示例

`==`

字符串比较时，会区分大小写。不同类型的值不相等。

2022 vesoft Inc.

> **Note**
>
> nGQL 中的相等符号是`==`，openCypher 中的相等符号是`=`。

```
nebula> RETURN 'A' == 'a', toUpper('A') == toUpper('a'), toLower('A') == toLower('a');
+-----------+-----------------------------+-----------------------------+
| ("A"=="a") | (toUpper("A")==toUpper("a")) | (toLower("A")==toLower("a")) |
+-----------+-----------------------------+-----------------------------+
| false     | true                        | true                        |
+-----------+-----------------------------+-----------------------------+

nebula> RETURN '2' == 2, toInteger('2') == 2;
+---------+--------------------+
| ("2"==2) | (toInteger("2")==2) |
+---------+--------------------+
| false    | true               |
+---------+--------------------+
```

`>`

```
nebula> RETURN 3 > 2;
+-------+
| (3>2) |
+-------+
| true  |
+-------+

nebula> WITH 4 AS one, 3 AS two \
        RETURN one > two AS result;
+--------+
| result |
+--------+
| true   |
+--------+
```

`>=`

```
nebula> RETURN 2 >= "2", 2 >= 2;
+----------+--------+
| (2>="2") | (2>=2) |
+----------+--------+
| __NULL__ | true   |
+----------+--------+
```

`<`

```
nebula> YIELD 2.0 < 1.9;
+---------+
| (2<1.9) |
+---------+
| false   |
+---------+
```

`<=`

```
nebula> YIELD 0.11 <= 0.11;
+--------------+
| (0.11<=0.11) |
+--------------+
| true         |
+--------------+
```

`!=`

```
nebula> YIELD 1 != '1';
+----------+
| (1!="1") |
+----------+
| true     |
+----------+
```

`IS [NOT] NULL`

```
nebula> RETURN null IS NULL AS value1, null == null AS value2, null != null AS value3;
+--------+----------+----------+
| value1 | value2   | value3   |
+--------+----------+----------+
| true   | __NULL__ | __NULL__ |
+--------+----------+----------+
```


2022 vesoft Inc.

```
nebula> RETURN length(NULL), size(NULL), count(NULL), NULL IS NULL, NULL IS NOT NULL, sin(NULL), NULL + NULL, [1, NULL] IS NULL;
+--------------+------------+-------------+--------------+------------------+-----------+-------------+------------------+
| length(NULL) | size(NULL) | count(NULL) | NULL IS NULL | NULL IS NOT NULL | sin(NULL) | (NULL+NULL) | [1,NULL] IS NULL |
+--------------+------------+-------------+--------------+------------------+-----------+-------------+------------------+
| __NULL__     | __NULL__   | 0           | true         | false            | __NULL__  | __NULL__    | false            |
+--------------+------------+-------------+--------------+------------------+-----------+-------------+------------------+

nebula> WITH {name: null} AS `map` \
        RETURN `map`.name IS NOT NULL;
+----------------------+
| map.name IS NOT NULL |
+----------------------+
| false                |
+----------------------+

nebula> WITH {name: 'Mats', name2: 'Pontus'} AS map1, \
        {name: null} AS map2, {notName: 0, notName2: null } AS map3 \
        RETURN map1.name IS NULL, map2.name IS NOT NULL, map3.name IS NULL;
+-------------------+-----------------------+-------------------+
| map1.name IS NULL | map2.name IS NOT NULL | map3.name IS NULL |
+-------------------+-----------------------+-------------------+
| false             | false                 | true              |
+-------------------+-----------------------+-------------------+

nebula> MATCH (n:player) \
        RETURN n.player.age IS NULL, n.player.name IS NOT NULL, n.player.empty IS NULL;
+----------------------+--------------------------+------------------------+
| n.player.age IS NULL | n.player.name IS NOT NULL | n.player.empty IS NULL |
+----------------------+--------------------------+------------------------+
| false                | true                     | true                   |
| false                | true                     | true                   |
...
```

IS [NOT] EMPTY

```
nebula> RETURN null IS EMPTY;
+---------------+
| NULL IS EMPTY |
+---------------+
| false         |
+---------------+

nebula> RETURN "a" IS NOT EMPTY;
+------------------+
| "a" IS NOT EMPTY |
+------------------+
| true             |
+------------------+

nebula> GO FROM "player100" OVER * WHERE properties($$).name IS NOT EMPTY YIELD dst(edge);
+-------------+
| dst(EDGE)   |
+-------------+
| "team204"   |
| "player101" |
| "player125" |
+-------------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.4.2 布尔符

Nebula Graph 支持的布尔符如下。

符号	说明
AND	逻辑与
OR	逻辑或
NOT	逻辑非
XOR	逻辑异或

对于以上运算的优先级，请参见运算优先级。

对于带有 `NULL` 的逻辑运算，请参见 NULL。

Note

非 0 数字不能转换为布尔值。

最后更新: June 20, 2023

2022 vesoft Inc.

4.4.3 管道符

nGQL 支持使用管道符（|）将多个查询组合起来。

openCypher 兼容性

管道符仅适用于原生 nGQL。

语法

nGQL 和 SQL 之间的一个主要区别是子查询的组成方式。

- 在 SQL 中，子查询是嵌套在查询语句中的。
- 在 nGQL 中，子查询是通过类似 shell 中的管道符（|）实现的。

示例

```
nebula> GO FROM "player100" OVER follow \
        YIELD dst(edge) AS dstid, properties($$).name AS Name | \
        GO FROM $-.dstid OVER follow YIELD dst(edge);

+-------------+
| dst(EDGE)   |
+-------------+
| "player100" |
| "player102" |
| "player125" |
| "player100" |
+-------------+
```

用户可以使用 `YIELD` 显式声明需要返回的结果，如果不使用 `YIELD`，默认返回目标点 ID。

必须在 `YIELD` 子句中为需要的返回结果设置别名，才能在管道符右侧使用引用符 `$-`，例如示例中的 `$-.dstid`。

性能提示

Nebula Graph 中的管道对性能有影响，以 `A | B` 为例，体现在以下几个方面：

1. 管道是同步操作。也即需要管道之前的子句 `A` 执行完毕后，数据才能整体进入管道子句。
2. 管道本身是需要序列化和反序列化的，这个是单线程执行的。
3. 如果 `A` 发大量数据给 `|`，整个查询请求的总体时延可能会非常大。此时可以尝试拆分这个语句：

a. 应用程序发送 `A`，
b. 将收到的返回结果在应用程序拆分，
c. 并发发送给多个 graphd，
d. 每个 graphd 执行部分 B。

这样通常比单个 graphd 执行完整地 `A | B` 要快很多。

最后更新: June 20, 2023

2022 vesoft Inc.

4.4.4 引用符

nGQL 提供引用符来表示 `WHERE` 和 `YIELD` 子句中的属性，或者复合查询中管道符之前的语句输出结果。

openCypher 兼容性

引用符仅适用于原生 nGQL。

引用符列表

引用符	说明
`$^`	引用起始点。更多信息请参见引用属性。
`$$`	引用目的点。更多信息请参见引用属性。
`$-`	引用复合查询中管道符之前的语句输出结果。更多信息请参见管道符。

示例

```
# 返回起始点和目的点的年龄。
nebula> GO FROM "player100" OVER follow YIELD properties($^).age AS SrcAge, properties($$).age AS DestAge;
+--------+---------+
| SrcAge | DestAge |
+--------+---------+
| 42     | 36      |
| 42     | 41      |
+--------+---------+

# 返回 player100 追随的 player 的名称和团队。
nebula> GO FROM "player100" OVER follow \
        YIELD dst(edge) AS id | \
        GO FROM $-.id OVER serve \
        YIELD properties($^).name AS Player, properties($$).name AS Team;
+-----------------+-----------+
| Player          | Team      |
+-----------------+-----------+
| "Tony Parker"   | "Spurs"   |
| "Tony Parker"   | "Hornets" |
| "Manu Ginobili" | "Spurs"   |
+-----------------+-----------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.4.5 集合运算符

合并多个请求时，可以使用集合运算符，包括 `UNION` 、`UNION ALL` 、`INTERSECT` 和 `MINUS` 。

所有集合运算符的优先级相同，如果一个 nGQL 语句中有多个集合运算符，Nebula Graph 会从左到右进行计算，除非用括号指定顺序。

openCypher 兼容性

集合运算符仅适用于原生 nGQL。

UNION、UNION DISTINCT、UNION ALL

```
<left> UNION [DISTINCT | ALL] <right> [ UNION [DISTINCT | ALL] <right> ...]
```

- 运算符 `UNION DISTINCT` （或使用缩写 `UNION` ）返回两个集合 A 和 B 的并集，不包含重复的元素。
- 运算符 `UNION ALL` 返回两个集合 A 和 B 的并集，包含重复的元素。
- `left` 和 `right` 必须有相同数量的列和数据类型。如果需要转换数据类型，请参见类型转换。

示例

```
# 返回两个查询结果的并集，不包含重复的元素。
nebula> GO FROM "player102" OVER follow YIELD dst(edge) \
        UNION \
        GO FROM "player100" OVER follow YIELD dst(edge);
+-------------+
| dst(EDGE)   |
+-------------+
| "player100" |
| "player101" |
| "player125" |
+-------------+

# 返回两个查询结果的并集，包含重复的元素。
nebula> GO FROM "player102" OVER follow YIELD dst(edge) \
        UNION ALL \
        GO FROM "player100" OVER follow YIELD dst(edge);
+-------------+
| dst(EDGE)   |
+-------------+
| "player100" |
| "player101" |
| "player101" |
| "player125" |
+-------------+

# UNION 也可以和 YIELD 语句一起使用，去重时会检查每一行的所有列，每列都相同时才会去重。
nebula> GO FROM "player102" OVER follow \
        YIELD dst(edge) AS id, properties(edge).degree AS Degree, properties($$).age AS Age \
        UNION /* DISTINCT */ \
        GO FROM "player100" OVER follow \
        YIELD dst(edge) AS id, properties(edge).degree AS Degree, properties($$).age AS Age;
+-------------+--------+-----+
| id          | Degree | Age |
+-------------+--------+-----+
| "player100" | 75     | 42  |
| "player101" | 75     | 36  |
| "player101" | 95     | 36  |
| "player125" | 95     | 41  |
+-------------+--------+-----+
```

INTERSECT

```
<left> INTERSECT <right>
```

- 运算符 `INTERSECT` 返回两个集合 A 和 B 的交集。
- `left` 和 `right` 必须有相同数量的列和数据类型。如果需要转换数据类型，请参见类型转换。

示例

```
nebula> GO FROM "player102" OVER follow \
        YIELD dst(edge) AS id, properties(edge).degree AS Degree, properties($$).age AS Age \
```


2022 vesoft Inc.

```
        INTERSECT \
        GO FROM "player100" OVER follow \
        YIELD dst(edge) AS id, properties(edge).degree AS Degree, properties($$).age AS Age;
+----+--------+-----+
| id | Degree | Age |
+----+--------+-----+
+----+--------+-----+
```

MINUS

```
<left> MINUS <right>
```

运算符 `MINUS` 返回两个集合 A 和 B 的差异，即 `A-B` 。请注意 `left` 和 `right` 的顺序， `A-B` 表示在集合 A 中，但是不在集合 B 中的元素。

示例

```
nebula> GO FROM "player100" OVER follow YIELD dst(edge) \
        MINUS \
        GO FROM "player102" OVER follow YIELD dst(edge);
+-------------+
| dst(EDGE)   |
+-------------+
| "player125" |
+-------------+

nebula> GO FROM "player102" OVER follow YIELD dst(edge) \
        MINUS \
        GO FROM "player100" OVER follow YIELD dst(edge);
+-------------+
| dst(EDGE)   |
+-------------+
| "player100" |
+-------------+
```

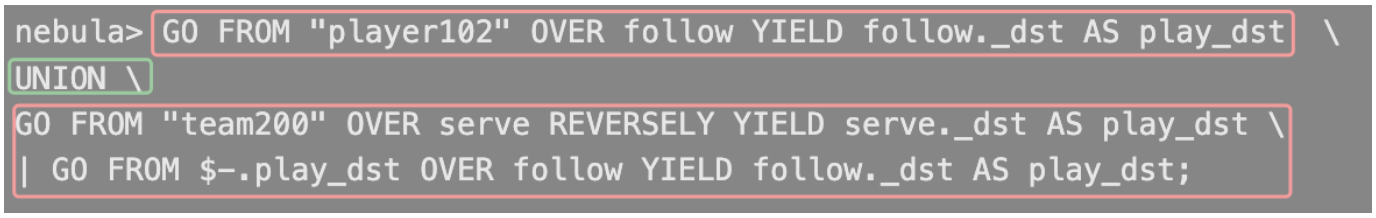
集合运算符和管道符的优先级

当查询包含集合运算符和管道符（|）时，管道符的优先级高。例如 `GO FROM 1 UNION GO FROM 2 | GO FROM 3` 相当于 `GO FROM 1 UNION (GO FROM 2 | GO FROM 3)` 。

示例

```
nebula> GO FROM "player102" OVER follow \
        YIELD dst(edge) AS play_dst  \
        UNION \
        GO FROM "team200" OVER serve REVERSELY \
        YIELD src(edge) AS play_src \
        | GO FROM $-.play_src OVER follow YIELD dst(edge) AS play_dst;

+-------------+
| play_dst    |
+-------------+
| "player100" |
| "player101" |
| "player117" |
| "player105" |
+-------------+
```



该查询会先执行红框内的语句，然后执行绿框的 `UNION` 操作。

圆括号可以修改执行的优先级，例如：

```
nebula> (GO FROM "player102" OVER follow \
        YIELD dst(edge) AS play_dst  \
        UNION \
        GO FROM "team200" OVER serve REVERSELY \
        YIELD src(edge) AS play_dst) \
        | GO FROM $-.play_dst OVER follow YIELD dst(edge) AS play_dst;
```


2022 vesoft Inc.

该查询中，圆括号包裹的部分先执行，即先执行 `UNION` 操作，再将结果结合管道符进行下一步操作。

最后更新: June 20, 2023

2022 vesoft Inc.

4.4.6 字符串运算符

Nebula Graph 支持使用字符串运算符进行连接、搜索、匹配运算。支持的运算符如下。

名称	说明
+	连接字符串。
CONTAINS	在字符串中执行搜索。
(NOT) IN	字符串是否匹配某个值。
(NOT) STARTS WITH	在字符串的开头执行匹配。
(NOT) ENDS WITH	在字符串的结尾执行匹配。
正则表达式	通过正则表达式匹配字符串。

Note

所有搜索或匹配都区分大小写。

示例

`+`

```
nebula> RETURN 'a' + 'b';
+----------+
| ("a"+"b") |
+----------+
| "ab"     |
+----------+
nebula> UNWIND 'a' AS a UNWIND 'b' AS b RETURN a + b;
+-------+
| (a+b) |
+-------+
| "ab"  |
+-------+
```

`CONTAINS`

`CONTAINS` 要求待运算的左右两边都是字符串类型。

```
nebula> MATCH (s:player)-[e:serve]->(t:team) WHERE id(s) == "player101" \
        AND t.team.name CONTAINS "ets" RETURN s.player.name, e.start_year, e.end_year, t.team.name;
+---------------+--------------+------------+-------------+
| s.player.name | e.start_year | e.end_year | t.team.name |
+---------------+--------------+------------+-------------+
| "Tony Parker" | 2018         | 2019       | "Hornets"   |
+---------------+--------------+------------+-------------+

nebula> GO FROM "player101" OVER serve WHERE (STRING)properties(edge).start_year CONTAINS "19" AND \
        properties($^).name CONTAINS "ny" \
        YIELD properties($^).name, properties(edge).start_year, properties(edge).end_year, properties($$).name;
+--------------------+-----------------------------+---------------------------+--------------------+
| properties($^).name | properties(EDGE).start_year | properties(EDGE).end_year | properties($$).name |
+--------------------+-----------------------------+---------------------------+--------------------+
| "Tony Parker"      | 1999                        | 2018                      | "Spurs"            |
+--------------------+-----------------------------+---------------------------+--------------------+

nebula> GO FROM "player101" OVER serve WHERE !(properties($$).name CONTAINS "ets") \
        YIELD properties($^).name, properties(edge).start_year, properties(edge).end_year, properties($$).name;
+--------------------+-----------------------------+---------------------------+--------------------+
| properties($^).name | properties(EDGE).start_year | properties(EDGE).end_year | properties($$).name |
+--------------------+-----------------------------+---------------------------+--------------------+
| "Tony Parker"      | 1999                        | 2018                      | "Spurs"            |
+--------------------+-----------------------------+---------------------------+--------------------+
```

`(NOT) IN`

```
nebula> RETURN  1 IN [1,2,3], "Yao" NOT IN ["Yi", "Tim", "Kobe"], NULL IN ["Yi", "Tim", "Kobe"];
+---------------+---------------------------------------+-------------------------------+
| (1 IN [1,2,3]) | ("Yao" NOT IN ["Yi","Tim","Kobe"]) | (NULL IN ["Yi","Tim","Kobe"]) |
+---------------+---------------------------------------+-------------------------------+
```


2022 vesoft Inc.

```
| true           | true                              | __NULL__                     |
+----------------+-----------------------------------+------------------------------+
```

(NOT) STARTS WITH

```
nebula> RETURN 'apple' STARTS WITH 'app', 'apple' STARTS WITH 'a', 'apple' STARTS WITH toUpper('a');
+---------------------------+--------------------------+---------------------------------+
| ("apple" STARTS WITH "app") | ("apple" STARTS WITH "a") | ("apple" STARTS WITH toUpper("a")) |
+---------------------------+--------------------------+---------------------------------+
| true                        | true                     | false                             |
+---------------------------+--------------------------+---------------------------------+

nebula> RETURN 'apple' STARTS WITH 'b','apple' NOT STARTS WITH 'app';
+--------------------------+---------------------------------+
| ("apple" STARTS WITH "b") | ("apple" NOT STARTS WITH "app") |
+--------------------------+---------------------------------+
| false                    | false                           |
+--------------------------+---------------------------------+
```

(NOT) ENDS WITH

```
nebula> RETURN 'apple' ENDS WITH 'app', 'apple' ENDS WITH 'e', 'apple' ENDS WITH 'E', 'apple' ENDS WITH 'b';
+-------------------------+-------------------------+-------------------------+-------------------------+
| ("apple" ENDS WITH "app") | ("apple" ENDS WITH "e") | ("apple" ENDS WITH "E") | ("apple" ENDS WITH "b") |
+-------------------------+-------------------------+-------------------------+-------------------------+
| false                   | true                    | false                   | false                   |
+-------------------------+-------------------------+-------------------------+-------------------------+
```

正则表达式

> **Note**
>
> 当前仅 opencypher 兼容语句（`MATCH` 、`WITH` 等）支持正则表达式，原生 nGQL 语句（`FETCH` 、`GO` 、`LOOKUP` 等）不支持正则表达式。

Nebula Graph 支持使用正则表达式进行过滤，正则表达式的语法是继承自 `std::regex` ，用户可以使用语法 `=~ '<regexp>'` 进行正则表达式匹配。例如：

```
nebula> RETURN "384748.39" =~ "\\d+(\\.\\d{2})?";
+---------------------------------+
| ("384748.39"=~"\d+(\.\d{2})?") |
+---------------------------------+
| true                            |
+---------------------------------+

nebula> MATCH (v:player) WHERE v.player.name =~ 'Tony.*' RETURN v.player.name;
+---------------+
| v.player.name |
+---------------+
| "Tony Parker" |
+---------------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.4.7 列表运算符

Nebula Graph 支持使用列表（List）运算符进行运算。支持的运算符如下。

名称	说明
+	连接列表。
IN	元素是否存在于列表中。
[]	使用下标操作符访问列表中的元素。

示例

```
nebula> YIELD [1,2,3,4,5]+[6,7] AS myList;
+-----------------------+
| myList                |
+-----------------------+
| [1, 2, 3, 4, 5, 6, 7] |
+-----------------------+

nebula> RETURN size([NULL, 1, 2]);
+------------------+
| size([NULL,1,2]) |
+------------------+
| 3                |
+------------------+

nebula> RETURN NULL IN [NULL, 1];
+--------------------+
| (NULL IN [NULL,1]) |
+--------------------+
| __NULL__           |
+--------------------+

nebula> WITH [2, 3, 4, 5] AS numberlist \
    UNWIND numberlist AS number \
    WITH number \
    WHERE number IN [2, 3, 8] \
    RETURN number;
+--------+
| number |
+--------+
| 2      |
| 3      |
+--------+

nebula> WITH ['Anne', 'John', 'Bill', 'Diane', 'Eve'] AS names RETURN names[1] AS result;
+--------+
| result |
+--------+
| "John" |
+--------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.4.8 运算符优先级

nGQL 运算符的优先级从高到低排列如下（同一行的运算符优先级相同）：

- `-`（负数）
- `!`、`NOT`
- `*`、`/`、`%`
- `-`、`+`
- `==`、`>=`、`>`、`<=`、`<`、`<>`、`!=`
- `AND`
- `OR`、`XOR`
- `=`（赋值）

如果表达式中有相同优先级的运算符，运算是从左到右进行，只有赋值操作是例外（从右到左运算）。

运算符的优先级决定运算的顺序，要显式修改运算顺序，可以使用圆括号。

示例

```
nebula> RETURN 2+3*5;
+-----------+
| (2+(3*5)) |
+-----------+
| 17        |
+-----------+

nebula> RETURN (2+3)*5;
+-----------+
| ((2+3)*5) |
+-----------+
| 25        |
+-----------+
```

openCypher 兼容性

在 openCypher 中，比较操作可以任意连接，例如 `x < y <= z` 等价于 `x < y AND y <= z`。

在 nGQL 中，`x < y <= z` 等价于 `(x < y) <= z`，`(x < y)` 的结果是一个布尔值，再将布尔值和 `z` 比较，最终结果是 `NULL`。

最后更新: June 20, 2023

2022 vesoft Inc.

4.5 函数和表达式

2022 vesoft Inc.

4.5.1 内置数学函数

函数说明

Nebula Graph 支持以下内置数学函数。

2022 vesoft Inc.

函数	说明
double abs(double x)	返回 x 的绝对值。
double floor(double x)	返回小于或等于 x 的最大整数。
double ceil(double x)	返回大于或等于 x 的最小整数。
double round(double x, int y)	返回 x 四舍五入后的值，y 为小数位数。y 小于 0 时 ， 在小数点左侧做四舍五入。 极端情况下请注意浮点数的精度问题。
double sqrt(double x)	返回 x 的平方根。
double cbrt(double x)	返回 x 的立方根。
double hypot(double x, double y)	返回直角三角形（直角边长为 x 和 y）的斜边长。
double pow(double x, double y)	返回 x^y 的值。
double exp(double x)	返回 e^x 的值。
double exp2(double x)	返回 2^x 的值。
double log(double x)	返回以自然数 e 为底 x 的对数。
double log2(double x)	返回以 2 为底 x 的对数。
double log10(double x)	返回以 10 为底 x 的对数。
double sin(double x)	返回 x 的正弦值。
double asin(double x)	返回 x 的反正弦值。
double cos(double x)	返回 x 的余弦值。
double acos(double x)	返回 x 的反余弦值。
double tan(double x)	返回 x 的正切值。
double atan(double x)	返回 x 的反正切值。
double rand()	返回 [0,1) 内的随机浮点数。
int rand32(int min, int max)	返回 `[min, max)` 内的一个随机 32 位整数。 用户可以只传入一个参数，该参数会判定为 `max`，此时 `min` 默认为 `0` 。 如果不传入参数，此时会从带符号的 32 位 int 范围内随机返回。
int rand64(int min, int max)	返回 `[min, max)` 内的一个随机 64 位整数。 用户可以只传入一个参数，该参数会判定为 `max`，此时 `min` 默认为 `0` 。 如果不传入参数，此时会从带符号的 64 位 int 范围内随机返回。
collect()	将收集的所有值放在一个列表中。
avg()	返回参数的平均值。
count()	返回参数的数量。
max()	返回参数的最大值。
min()	返回参数的最小值。
std()	返回参数的总体标准差。
sum()	返回参数的和。
bit_and()	逐位做 AND 操作。
bit_or()	逐位做 OR 操作。
bit_xor()	逐位做 XOR 操作。

2022 vesoft Inc.

函数	说明
int size()	返回列表或映射中元素的数量。
int range(int start, int end, int step)	返回 `[start,end]` 中指定步长的值组成的列表。步长 `step` 默认为 1。
int sign(double x)	返回 x 的正负号。 如果 x 为 `0`，则返回 `0`。 如果 x 为负数，则返回 `-1`。 如果 x 为正数，则返回 `1`。
double e()	返回自然对数的底 e（2.718281828459045）。
double pi()	返回数学常数π（3.141592653589793）。
double radians()	将角度转换为弧度。`radians(180)` 返回 `3.141592653589793`。

Note

如果参数为 `NULL`，则输出结果是未定义的。

示例

```
# 支持聚合函数
nebula>  GO FROM "player100" OVER follow YIELD dst(edge) AS dst, properties($$).age AS age \
         | GROUP BY $-.dst \
         YIELD \
         $-.dst AS dst, \
         toInteger((sum($-.age)/count($-.age)))+avg(distinct $-.age+1)+1 AS statistics;
+-------------+------------+
| dst         | statistics |
+-------------+------------+
| "player125" | 84.0       |
| "player101" | 74.0       |
+-------------+------------+
Got 2 rows (time spent 4739/5064 us)
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.5.2 内置字符串函数

Nebula Graph 支持以下内置字符串函数。

> **Note**

和 SQL 一样，nGQL 的字符索引（位置）从 `1` 开始。但是 C 语言的字符索引是从 `0` 开始的。

函数	说明
int strcasecmp(string a, string b)	比较两个字符串（不区分大小写）。当 a=b 时，返回 0，当 a>b 是，返回大于 0 的数，当 a<b 时，返回小于 0 的数。
string lower(string a)	返回小写形式的字符串。
string toLower(string a)	和 `lower()` 相同。
string upper(string a)	返回大写形式的字符串。
string toUpper(string a)	和 `upper()` 相同。
int length(string a)	以字节为单位，返回给定字符串的长度。
string trim(string a)	删除字符串头部和尾部的空格。
string ltrim(string a)	删除字符串头部的空格。
string rtrim(string a)	删除字符串尾部的空格。
string left(string a, int count)	返回字符串左侧 `count` 个字符组成的子字符串。如果 `count` 超过字符串 a 的长度，则返回字符串 a。
string right(string a, int count)	返回字符串右侧 `count` 个字符组成的子字符串。如果 `count` 超过字符串 a 的长度，则返回字符串 a。
string lpad(string a, int size, string letters)	在字符串 a 的左侧填充 `letters` 字符串，并返回 `size` 长度的字符串。
string rpad(string a, int size, string letters)	在字符串 a 的右侧填充 `letters` 字符串，并返回 `size` 长度的字符串。
string substr(string a, int pos, int count)	从字符串 a 的指定位置 `pos` 开始（不包括 `pos` 位置的字符），提取右侧的 `count` 个字符，组成新的字符串并返回。
string substring(string a, int pos, int count)	和 `substr()` 相同。
string reverse(string)	逆序返回字符串。
string replace(string a, string b, string c)	将字符串 a 中的子字符串 b 替换为字符串 c。
list split(string a, string b)	在子字符串 b 处拆分字符串 a，返回一个字符串列表。
string toString()	将任意数据类型转换为字符串类型。
int hash()	获取任意对象的哈希值。

> **Note**

如果参数为 `NULL`，则输出结果是未定义的。

2022 vesoft Inc.

`substr()` 和 **`substring()`** 的返回说明

- 字符索引（位置）从 `0` 开始。
- 如果 `pos` 为 0，则返回整个字符串。
- 如果 `pos` 大于最大字符索引，则返回空字符串。
- 如果 `pos` 是负数，则返回 `BAD_DATA` 。
- 如果省略 `count` ，则返回从 `pos` 位置开始到字符串末尾的子字符串。
- 如果 `count` 为 0，则返回空字符串。
- 使用 `NULL` 作为任何参数会出现错误。

> **openCypher** 兼容性

- 在 openCypher 中，如果字符串 `a` 为 `null` ，会返回 `null` 。
- 在 openCypher 中，如果 `pos` 为 0，会返回从第一个字符开始 `count` 个字符的子字符串。
- 在 openCypher 中，如果 `pos` 或 `count` 为 `null` 或负整数，会出现错误。

最后更新: June 20, 2023

2022 vesoft Inc.

4.5.3 内置日期时间函数

Nebula Graph 支持以下内置日期时间函数。

函数	说明
int now()	根据当前系统返回当前时区的时间戳。
timestamp timestamp()	根据当前系统返回当前时区的时间戳。
date date()	根据当前系统返回当前日期（UTC 时间）。
time time()	根据当前系统返回当前时间（UTC 时间）。
datetime datetime()	根据当前系统返回当前日期和时间（UTC 时间）。
map duration()	持续时间。可以用于对指定时间进行计算。

详细信息参见日期和时间类型。

示例

```
nebula>  RETURN now(), timestamp(), date(), time(), datetime();
+------------+-------------+------------+-----------------+----------------------------+
| now()      | timestamp() | date()     | time()          | datetime()                 |
+------------+-------------+------------+-----------------+----------------------------+
| 1640057560 | 1640057560  | 2021-12-21 | 03:32:40.351000 | 2021-12-21T03:32:40.351000 |
+------------+-------------+------------+-----------------+----------------------------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.5.4 Schema 函数

Nebula Graph 支持以下 Schema 函数。

原生 **nGQL** 语句适用

> **Note**

YIELD 和 WHERE 子句中可以使用如下函数。

函数	说明
id(vertex)	返回点 ID。数据类型和点 ID 的类型保持一致。
map properties(vertex)	返回点的所有属性。
map properties(edge)	返回边的所有属性。
string type(edge)	返回边的 Edge type。
src(edge)	返回边的起始点 ID。数据类型和点 ID 的类型保持一致。
dst(edge)	返回边的目的点 ID。数据类型和点 ID 的类型保持一致。
int rank(edge)	返回边的 rank。
vertex	返回点的信息。包括点 ID、Tag、属性和值。
edge	返回边的信息。包括 Edge type、起始点 ID、目的点 ID、rank、属性和值。
vertices	返回子图中的点的信息。详情参见 GET SUBGRAPH。
edges	返回子图中的边的信息。详情参见 GET SUBGRAPH。
path	返回路径信息。详情参见 FIND PATH。

> **Note**

由于 vertex、edge、vertices、edges、path 属于关键字，使用时需要用 `AS <alias>` 设置别名才能正常使用。例如 `GO FROM "player100" OVER follow YIELD edge AS e;`。

2022 vesoft Inc.

openCypher 兼容语句适用

函数	说明
id(<vertex>)	返回点 ID。数据类型和点 ID 的类型保持一致。
list tags(<vertex>)	返回点的 Tag，与 labels() 作用相同。
list labels(<vertex>)	返回点的 Tag，与 tags() 作用相同，用于兼容 openCypher 语法。
map properties(<vertex_or_edge>)	返回点或边的所有属性。
string type(<edge>)	返回边的 Edge type。
src(<edge>)	返回边的起始点 ID。数据类型和点 ID 的类型保持一致。
dst(<edge>)	返回边的目的点 ID。数据类型和点 ID 的类型保持一致。
vertex startNode(<path>)	获取一条路径并返回它的起始点 ID。
string endNode(<path>)	获取一条路径并返回它的目的点 ID。
int rank(<edge>)	返回边的 rank。

示例

```
nebula> GO FROM "player100" OVER follow REVERSELY \
        YIELD src(edge) AS destination;
+-------------+
| destination |
+-------------+
| "player101" |
| "player102" |
...

nebula> LOOKUP ON player WHERE player.age  > 45 YIELD id(vertex);
+-------------+
| id(VERTEX)  |
+-------------+
| "player144" |
| "player140" |
+-------------+

nebula> MATCH (a:player) WHERE id(a) == "player100" \
        RETURN tags(a), labels(a), properties(a);
+------------+------------+--------------------------------+
| tags(a)    | labels(a)  | properties(a)                  |
+------------+------------+--------------------------------+
| ["player"] | ["player"] | {age: 42, name: "Tim Duncan"} |
+------------+------------+--------------------------------+

nebula> MATCH p = (a :player {name : "Tim Duncan"})-[r:serve]-(t) \
        RETURN type(r), rank(r);
+---------+---------+
| type(r) | rank(r) |
+---------+---------+
| "serve" | 0       |
+---------+---------+

nebula> MATCH p = (a :player {name : "Tim Duncan"})-[r:serve]-(t) \
        RETURN startNode(p), endNode(p);
+--------------------------------------------------+---------------------------------+
| startNode(p)                                     | endNode(p)                      |
+--------------------------------------------------+---------------------------------+
| ("player100" :player{age: 42, name: "Tim Duncan"}) | ("team204" :team{name: "Spurs"}) |
+--------------------------------------------------+---------------------------------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.5.5 CASE 表达式

`CASE` 表达式使用条件来过滤 nGQL 查询语句的结果，常用于 `YIELD` 和 `RETURN` 子句中。和 openCypher 一样，nGQL 提供两种形式的 `CASE` 表达式：简单形式和通用形式。

`CASE` 表达式会遍历所有条件，并在满足第一个条件时停止读取后续条件，然后返回结果。如果不满足任何条件，将通过 `ELSE` 子句返回结果。如果没有 `ELSE` 子句且不满足任何条件，则返回 `NULL` 。

简单形式

语法

```
CASE <comparer>
WHEN <value> THEN <result>
[WHEN ...]
[ELSE <default>]
END
```

Caution

`CASE` 表达式一定要用 `END` 结尾。

参数	说明
`comparer`	用于与 `value` 进行比较的值或者有效表达式。
`value`	和 `comparer` 进行比较，如果匹配，则满足此条件。
`result`	如果 `value` 匹配 `comparer` ，则返回该 `result` 。
`default`	如果没有条件匹配，则返回该 `default` 。

示例

```
nebula> RETURN \
        CASE 2+3 \
        WHEN 4 THEN 0 \
        WHEN 5 THEN 1 \
        ELSE -1 \
        END \
        AS result;
+--------+
| result |
+--------+
| 1      |
+--------+
```

```
nebula> GO FROM "player100" OVER follow \
        YIELD properties($$).name AS Name, \
        CASE properties($$).age > 35 \
        WHEN true THEN "Yes" \
        WHEN false THEN "No" \
        ELSE "Nah" \
        END \
        AS Age_above_35;
+-----------------+--------------+
| Name            | Age_above_35 |
+-----------------+--------------+
| "Tony Parker"   | "Yes"        |
| "Manu Ginobili" | "Yes"        |
+-----------------+--------------+
```

通用形式

语法

```
CASE
WHEN <condition> THEN <result>
[WHEN ...]
```


2022 vesoft Inc.

```
[ELSE <default>]
END
```

参数	说明
`condition`	如果条件 `condition` 为 true，表示满足此条件。
`result`	`condition` 为 true，则返回此 `result` 。
`default`	如果没有条件匹配，则返回该 `default` 。

示例

```
nebula> YIELD \
        CASE WHEN 4 > 5 THEN 0 \
        WHEN 3+4==7 THEN 1 \
        ELSE 2 \
        END \
        AS result;
+--------+
| result |
+--------+
| 1      |
+--------+
```

```
nebula> MATCH (v:player) WHERE v.player.age > 30 \
        RETURN v.player.name AS Name,  \
        CASE \
        WHEN v.player.name STARTS WITH "T" THEN "Yes" \
        ELSE "No" \
        END \
        AS Starts_with_T;
+---------------------+---------------+
| Name                | Starts_with_T |
+---------------------+---------------+
| "Tim"               | "Yes"         |
| "LaMarcus Aldridge" | "No"          |
| "Tony Parker"       | "Yes"         |
+---------------------+---------------+
```

简单形式和通用形式的区别

为了避免误用简单形式和通用形式，用户需要了解它们的差异。请参见如下示例：

```
nebula> GO FROM "player100" OVER follow \
        YIELD properties($$).name AS Name, properties($$).age AS Age, \
        CASE properties($$).age \
        WHEN properties($$).age > 35 THEN "Yes" \
        ELSE "No" \
        END \
        AS Age_above_35;
+-----------------+-----+--------------+
| Name            | Age | Age_above_35 |
+-----------------+-----+--------------+
| "Tony Parker"   | 36  | "No"         |
| "Manu Ginobili" | 41  | "No"         |
+-----------------+-----+--------------+
```

示例本意为当玩家年龄大于 35 时输出 `Yes` 。但是查看输出结果，年龄为 36 时输出的却是 `No` 。

这是因为查询使用了简单形式的 `CASE` 表达式，比较对象是 `$$.player.age` 和 `$$.player.age > 35` 。当年龄为 36 时：

- `$$.player.age` 的值为 `36` ，数据类型为 int。
- `$$.player.age > 35` 的值为 `true` ，数据类型为 boolean。

这两种数据类型无法匹配，不满足条件，因此返回 `No` 。

最后更新: June 20, 2023

2022 vesoft Inc.

4.5.6 列表函数

Nebula Graph 支持以下列表（List）函数。

函数	说明
keys(expr)	返回一个列表，包含字符串形式的点、边或映射的所有属性。
labels(vertex)	返回点的 Tag 列表。
nodes(path)	返回路径中所有点的列表。
range(start, end [, step])	返回 `[start,end]` 范围内固定步长的列表，默认步长 `step` 为 1。
relationships(path)	返回路径中所有关系的列表。
reverse(list)	返回将原列表逆序排列的新列表。
tail(list)	返回不包含原列表第一个元素的新列表。
head(list)	返回列表的第一个元素。
last(list)	返回列表的最后一个元素。
coalesce(item0 [, item1, item2])	返回多个参数中第一个非空元素。
reduce()	请参见 reduce 函数。

Note

如果参数为 `NULL`，则输出结果是未定义的。

示例

```
nebula> WITH [NULL, 4923, 'abc', 521, 487] AS ids \
        RETURN reverse(ids), tail(ids), head(ids), last(ids), coalesce(ids[0], ids[1], ids[2]), head([x IN ids WHERE x IS NOT NULL | x]);
+-----------------------------------+-------------------------+-----------+-----------+-------------------------------+-------------------------------------------------------------+
| reverse(ids)                      | tail(ids)               | head(ids) | last(ids) | coalesce(ids[0],ids[1],ids[2]) | head([__VAR_0 IN ids WHERE $__VAR_0 IS NOT NULL | $__VAR_0]) |
+-----------------------------------+-------------------------+-----------+-----------+-------------------------------+-------------------------------------------------------------+
| [487, 521, "abc", 4923, __NULL__] | [4923, "abc", 521, 487] | __NULL__  | 487       | 4923                          | 4923                                                        |
+-----------------------------------+-------------------------+-----------+-----------+-------------------------------+-------------------------------------------------------------+

nebula> MATCH (a:player)-[r]->() \
        WHERE id(a) == "player100" \
        RETURN labels(a),  keys(r);
+------------+----------------------------+
| labels(a)  | keys(r)                    |
+------------+----------------------------+
| ["player"] | ["end_year", "start_year"] |
| ["player"] | ["degree"]                 |
| ["player"] | ["degree"]                 |
+------------+----------------------------+
nebula> MATCH p = (a:player)-[]->(b)-[]->(c:team) \
        WHERE a.player.name == "Tim Duncan" AND c.team.name == "Spurs" \
        RETURN nodes(p);
+-----------------------------------------------------------------------------------------------------------------------------------------------------+
| nodes(p)                                                                                                                                            |
+-----------------------------------------------------------------------------------------------------------------------------------------------------+
| [("player100" :player{age: 42, name: "Tim Duncan"}), ("player101" :player{age: 36, name: "Tony Parker"}), ("team204" :team{name: "Spurs"})]   |
| [("player100" :player{age: 42, name: "Tim Duncan"}), ("player125" :player{age: 41, name: "Manu Ginobili"}), ("team204" :team{name: "Spurs"})] |
+-----------------------------------------------------------------------------------------------------------------------------------------------------+

nebula> MATCH p = (a:player)-[]->(b)-[]->(c:team) WHERE a.player.name == "Tim Duncan" AND c.team.name == "Spurs" RETURN relationships(p);
+-----------------------------------------------------------------------------------------------------------------------------+
| relationships(p)                                                                                                            |
+-----------------------------------------------------------------------------------------------------------------------------+
| [[:follow "player100"->"player101" @0 {degree: 95}], [:serve "player101"->"team204" @0 {end_year: 2018, start_year: 1999}]] |
| [[:follow "player100"->"player125" @0 {degree: 95}], [:serve "player125"->"team204" @0 {end_year: 2018, start_year: 2002}]] |
+-----------------------------------------------------------------------------------------------------------------------------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.5.7 count 函数

`count()` 函数可以计数指定的值或行数。

- （原生 nGQL）用户可以同时使用 `count()` 和 `GROUP BY` 对指定的值进行分组和计数，再使用 `YIELD` 返回结果。
- （openCypher 方式）用户可以使用 `count()` 对指定的值进行计数，再使用 `RETURN` 返回结果。不需要使用 `GROUP BY` 。

语法

```
count({expr | *})
```

- count(*) 返回总行数（包括 NULL）。
- count(expr) 返回满足表达式的非空值的总数。
- `count()` 和 `size()` 是不同的。

示例

```
nebula> WITH [NULL, 1, 1, 2, 2] As a UNWIND a AS b \
        RETURN count(b), count(*), count(DISTINCT b);
+----------+----------+------------------+
| count(b) | count(*) | count(distinct b) |
+----------+----------+------------------+
| 4        | 5        | 2                |
+----------+----------+------------------+
```

```
# 返回 player101 follow 的人，以及 follow player101 的人，即双向查询。
nebula> GO FROM "player101" OVER follow BIDIRECT \
        YIELD properties($$).name AS Name \
        | GROUP BY $-.Name YIELD $-.Name, count(*);
+---------------------+----------+
| $-.Name             | count(*) |
+---------------------+----------+
| "LaMarcus Aldridge" | 2        |
| "Tim Duncan"        | 2        |
| "Marco Belinelli"   | 1        |
| "Manu Ginobili"     | 1        |
| "Boris Diaw"        | 1        |
| "Dejounte Murray"   | 1        |
+---------------------+----------+
```

上述示例的返回结果有两列：

- `$-.Name`： 查询结果包含的姓名。
- `count(*)`： 姓名出现的次数。

因为测试数据集 `basketballplayer` 中没有重复的姓名， `count(*)` 列中数字 `2` 表示该行的人和 `player101` 是互相 `follow` 的关系。

```
# 方法一：统计数据库中的年龄分布情况。
nebula> LOOKUP ON player \
        YIELD player.age As playerage \
        | GROUP BY $-.playerage \
        YIELD $-.playerage as age, count(*) AS number \
        | ORDER BY $-.number DESC, $-.age DESC;
+-----+--------+
| age | number |
+-----+--------+
| 34  | 4      |
| 33  | 4      |
| 30  | 4      |
| 29  | 4      |
| 38  | 3      |
+-----+--------+
...

# 方法二：统计数据库中的年龄分布情况。
nebula> MATCH (n:player) \
        RETURN n.player.age as age, count(*) as number \
        ORDER BY number DESC, age DESC;
+-----+--------+
| age | number |
+-----+--------+
| 34  | 4      |
```


2022 vesoft Inc.

```
| 33  | 4      |
| 30  | 4      |
| 29  | 4      |
| 38  | 3      |
+-----+--------+
...
```

```
# 统计 Tim Duncan 关联的边数。
nebula> MATCH (v:player{name:"Tim Duncan"}) -- (v2) \
        RETURN count(DISTINCT v2);
+-------------------+
| count(distinct v2) |
+-------------------+
| 11                |
+-------------------+

# 多跳查询，统计 Tim Duncan 关联的边数，返回两列（不去重和去重）。
nebula> MATCH (n:player {name : "Tim Duncan"})-[]->(friend:player)-[]->(fof:player) \
        RETURN count(fof), count(DISTINCT fof);
+------------+---------------------+
| count(fof) | count(distinct fof) |
+------------+---------------------+
| 4          | 3                   |
+------------+---------------------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.5.8 collect 函数

`collect()` 函数返回一个符合表达式返回结果的列表。该函数可以将多条记录或值合并进一个列表，实现数据聚合。

`collect()` 是一个聚合函数，类似 SQL 中的 `GROUP BY` 。

示例

```
nebula> UNWIND [1, 2, 1] AS a \
        RETURN a;
+---+
| a |
+---+
| 1 |
| 2 |
| 1 |
+---+

nebula> UNWIND [1, 2, 1] AS a \
        RETURN collect(a);
+------------+
| collect(a) |
+------------+
| [1, 2, 1]  |
+------------+

nebula> UNWIND [1, 2, 1] AS a \
        RETURN a, collect(a), size(collect(a));
+---+------------+------------------+
| a | collect(a) | size(collect(a)) |
+---+------------+------------------+
| 2 | [2]        | 1                |
| 1 | [1, 1]     | 2                |
+---+------------+------------------+

# 降序排列，限制输出行数为 3，然后将结果输出到列表中。
nebula> UNWIND ["c", "b", "a", "d" ] AS p \
        WITH p AS q \
        ORDER BY q DESC LIMIT 3 \
        RETURN collect(q);
+-----------------+
| collect(q)      |
+-----------------+
| ["d", "c", "b"] |
+-----------------+

nebula> WITH [1, 1, 2, 2] AS coll \
        UNWIND coll AS x \
        WITH DISTINCT x \
        RETURN collect(x) AS ss;
+--------+
| ss     |
+--------+
| [1, 2] |
+--------+

nebula> MATCH (n:player) \
        RETURN collect(n.player.age);
+-------------------------------------------------------------+
| collect(n.player.age)                                       |
+-------------------------------------------------------------+
| [32, 32, 34, 29, 41, 40, 33, 25, 40, 37, ...
...

# 基于年龄聚合姓名。
nebula> MATCH (n:player) \
        RETURN n.player.age AS age, collect(n.player.name);
+-----+-------------------------------------------------------------------------+
| age | collect(n.player.name)                                                  |
+-----+-------------------------------------------------------------------------+
| 24  | ["Giannis Antetokounmpo"]                                               |
| 20  | ["Luka Doncic"]                                                         |
| 25  | ["Joel Embiid", "Kyle Anderson"]                                        |
+-----+-------------------------------------------------------------------------+
...
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.5.9 reduce 函数

`reduce()` 将表达式逐个应用于列表中的元素，然后和累加器中的当前结果累加，最后返回完整结果。该函数将遍历给定列表中的每个元素 `e`，在 `e` 上运行表达式并和累加器的当前结果累加，将新的结果存储在累加器中。这个函数类似于函数式语言（如 Lisp 和 Scala）中的 fold 或 reduce 方法。

openCypher 兼容性

在 openCypher 中，`reduce()` 函数没有定义。nGQL 使用了 Cypher 方式实现 `reduce()` 函数。

语法

```
reduce(<accumulator> = <initial>, <variable> IN <list> | <expression>)
```

参数	说明
accumulator	在遍历列表时保存累加结果。
initial	为 `accumulator` 提供初始值的表达式或值。
variable	为列表引入一个变量，决定使用列表中的哪个元素。
list	列表或列表表达式。
expression	该表达式将对列表中的每个元素运行一次，并将结果累加至 `accumulator` 。

Note

返回值的类型取决于提供的参数，以及表达式的语义。

示例

```
nebula> RETURN reduce(totalNum = 10, n IN range(1, 3) | totalNum + n) AS r;
+----+
| r  |
+----+
| 16 |
+----+

nebula> RETURN reduce(totalNum = -4 * 5, n IN [1, 2] | totalNum + n * 2) AS r;
+-----+
| r   |
+-----+
| -14 |
+-----+

nebula> MATCH p = (n:player{name:"LeBron James"})<-[:follow]-(m) \
        RETURN  nodes(p)[0].age AS src1,  nodes(p)[1].age AS dst2,  \
        reduce(totalAge = 100, n IN nodes(p) | totalAge + n.age) AS sum;
+------+------+-----+
| src1 | dst2 | sum |
+------+------+-----+
| 34   | 31   | 165 |
| 34   | 29   | 163 |
| 34   | 33   | 167 |
| 34   | 26   | 160 |
| 34   | 34   | 168 |
| 34   | 37   | 171 |
+------+------+-----+

nebula> LOOKUP ON player WHERE player.name == "Tony Parker" YIELD id(vertex) AS VertexID \
        |  GO FROM $-.VertexID over follow \
        WHERE properties(edge).degree != reduce(totalNum = 5, n IN range(1, 3) | properties($$).age + totalNum + n) \
        YIELD properties($$).name AS id, properties($$).age AS age, properties(edge).degree AS degree;
+--------------------+-----+--------+
| id                 | age | degree |
+--------------------+-----+--------+
| "Tim Duncan"       | 42  | 95     |
| "LaMarcus Aldridge" | 33  | 90     |
| "Manu Ginobili"    | 41  | 95     |
+--------------------+-----+--------+
```


2022 vesoft Inc.

最后更新: June 20, 2023

2022 vesoft Inc.

4.5.10 hash 函数

`hash()` 函数返回参数的哈希值。其参数可以是数字、字符串、列表、布尔值、NULL 等类型的值，或者计算结果为这些类型的表达式。

`hash()` 函数采用 MurmurHash2 算法，种子（seed）为 `0xc70f6907UL`。用户可以在 `MurmurHash2.h` 中查看其源代码。

在 Java 中的调用方式如下：

```
MurmurHash2.hash64("to_be_hashed".getBytes(),"to_be_hashed".getBytes().length, 0xc70f6907)
```

历史版本兼容性

nGQL 1.0 不支持字符串类型的 VID，一种常用的处理方式是用 hash 函数获取字符串的哈希值，然后将该值设置为 VID。但从 nGQL 2.0 开始，同时支持了字符串和整数类型的 VID，所以无需再使用这种方式设置 VID。

计算数字的哈希值

```
nebula> YIELD hash(-123);
+--------------+
| hash(-(123)) |
+--------------+
| -123         |
+--------------+
```

计算字符串的哈希值

```
nebula> YIELD hash("to_be_hashed");
+----------------------+
| hash("to_be_hashed") |
+----------------------+
| -1098333533029391540 |
+----------------------+
```

计算列表的哈希值

```
nebula> YIELD hash([1,2,3]);
+----------------+
| hash([1,2,3])  |
+----------------+
| 11093822460243 |
+----------------+
```

计算布尔值的哈希值

```
nebula> YIELD hash(true);
+------------+
| hash(true) |
+------------+
| 1          |
+------------+

nebula> YIELD hash(false);
+-------------+
| hash(false) |
+-------------+
| 0           |
+-------------+
```

计算 **NULL** 的哈希值

```
nebula> YIELD hash(NULL);
+------------+
| hash(NULL) |
+------------+
| -1         |
+------------+
```


2022 vesoft Inc.

计算表达式的哈希值

```
nebula> YIELD hash(toLower("HELLO NEBULA"));
+-------------------------------+
| hash(toLower("HELLO NEBULA")) |
+-------------------------------+
| -8481157362655072082          |
+-------------------------------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.5.11 concat 函数

`concat()` 和 `concat_ws()` 函数返回一个或多个字符串连接产生的字符串。

concat() 函数

`concat()` 函数至少需要两个或以上字符串参数，并将所有参数连接成一个字符串。

- 如果字符串参数只有一个，则返回该字符串参数本身。
- 如果任何一个的字符串参数为 `NULL`，则 `concat()` 函数返回值为 `NULL`。

语法

```
concat(string1,string2,...)
```

示例

```
//连接 1, 2, 3
nebula> RETURN concat("1","2","3") AS r;
+-------+
| r     |
+-------+
| "123" |
+-------+

//字符串参数有 NULL
nebula> RETURN concat("1","2",NULL) AS r;
+----------+
| r        |
+----------+
| __NULL__ |
+----------+

nebula> GO FROM "player100" over follow \
        YIELD concat(src(edge), properties($^).age, properties($$).name, properties(edge).degree) AS A;
+------------------------------+
| A                            |
+------------------------------+
| "player10042Tony Parker95"   |
| "player10042Manu Ginobili95" |
+------------------------------+
```

concat_ws() 函数

`concat_ws()` 函数将两个或以上字符串参数与预定义的分隔符（separator）相连接。

- 如果分隔符为 `NULL` 时，`concat_ws()` 函数才返回 `NULL`。
- 如果分隔符不为 `NULL`，字符串参数只有一个，则返回该字符串参数本身。
- 字符串参数存在 `NULL` 值时，忽略 `NULL` 值，继续连接下一个参数。

语法

```
concat_ws(separator,string1,string2,... )
```

示例

```
//分隔符为+, 连接 a, b, c
nebula> RETURN concat_ws("+","a","b","c") AS r;
+---------+
| r       |
+---------+
| "a+b+c" |
+---------+

//分隔符为 NULL
neubla> RETURN concat_ws(NULL,"a","b","c") AS r;
+----------+
| r        |
+----------+
| __NULL__ |
+----------+
```


2022 vesoft Inc.

```
//分隔符为+，字符串参数有 NULL
nebula> RETURN concat_ws("+","a",NULL,"b","c") AS r;
+---------+
| r       |
+---------+
| "a+b+c" |
+---------+

//分隔符为+。字符串参数只有一个
nebula> RETURN concat_ws("+","a") AS r;
+-----+
| r   |
+-----+
| "a" |
+-----+

nebula> GO FROM "player100" over follow \
        YIELD concat_ws(" ",src(edge), properties($^).age, properties($$).name, properties(edge).degree) AS A;
+---------------------------------+
| A                               |
+---------------------------------+
| "player100 42 Tony Parker 95"   |
| "player100 42 Manu Ginobili 95" |
+---------------------------------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.5.12 谓词函数

谓词函数只返回 `true` 或 `false`，通常用于 `WHERE` 子句中。

Nebula Graph 支持以下谓词函数。

函数	说明
exists()	如果指定的属性在点、边或映射中存在，则返回 `true`，否则返回 `false`。
any()	如果指定的谓词适用于列表中的至少一个元素，则返回 `true`，否则返回 `false`。
all()	如果指定的谓词适用于列表中的每个元素，则返回 `true`，否则返回 `false`。
none()	如果指定的谓词不适用于列表中的任何一个元素，则返回 `true`，否则返回 `false`。
single()	如果指定的谓词适用于列表中的唯一一个元素，则返回 `true`，否则返回 `false`。

Note

如果列表为空，或者列表中的所有元素都为空，则返回 NULL。

Compatibility

在 openCypher 中只定义了函数 `exists()`，其他几个函数依赖于具体实现。

语法

```
<predicate>(<variable> IN <list> WHERE <condition>)
```

示例

```
nebula> RETURN any(n IN [1, 2, 3, 4, 5, NULL] \
        WHERE n > 2) AS r;
+------+
| r    |
+------+
| true |
+------+

nebula> RETURN single(n IN range(1, 5) \
        WHERE n == 3) AS r;
+------+
| r    |
+------+
| true |
+------+

nebula> RETURN none(n IN range(1, 3) \
        WHERE n == 0) AS r;
+------+
| r    |
+------+
| true |
+------+

nebula> WITH [1, 2, 3, 4, 5, NULL] AS a \
        RETURN any(n IN a WHERE n > 2);
+-------------------------+
| any(n IN a WHERE (n>2)) |
+-------------------------+
| true                    |
+-------------------------+

nebula> MATCH p = (n:player{name:"LeBron James"})<-[:follow]-(m) \
        RETURN nodes(p)[0].name AS n1, nodes(p)[1].name AS n2, \
        all(n IN nodes(p) WHERE n.name NOT STARTS WITH "D") AS b;
+----------------+--------------------+-------+
| n1             | n2                 | b     |
+----------------+--------------------+-------+
| "LeBron James" | "Danny Green"      | false |
```


2022 vesoft Inc.

```
| "LeBron James" | "Dejounte Murray" | false |
| "LeBron James" | "Chris Paul"      | true  |
| "LeBron James" | "Kyrie Irving"    | true  |
| "LeBron James" | "Carmelo Anthony" | true  |
| "LeBron James" | "Dwyane Wade"     | false |
+---------------+-------------------+-------+

nebula> MATCH p = (n:player{name:"LeBron James"})-[:follow]->(m) \
        RETURN single(n IN nodes(p) WHERE n.age > 40) AS b;
+------+
| b    |
+------+
| true |
+------+

nebula> MATCH (n:player) \
        RETURN exists(n.player.id), n IS NOT NULL;
+-------------+--------------+
| exists(n.id) | n IS NOT NULL |
+-------------+--------------+
| false        | true          |
...

nebula> MATCH (n:player) \
        WHERE exists(n['name']) \
        RETURN n;
+-------------------------------------------------------------+
| n                                                           |
+-------------------------------------------------------------+
| ("player105" :player{age: 31, name: "Danny Green"})         |
| ("player109" :player{age: 34, name: "Tiago Splitter"})      |
| ("player111" :player{age: 38, name: "David West"})          |
...
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.5.13 geo 函数

geo 函数用于生成地理位置（GEOGRAPHY）数据类型的值或对其执行操作。

关于地理位置数据类型说明请参见地理位置。

函数说明

函数	返回类型	说明
ST_Point(longitude, latitude)	`GEOGRAPHY`	创建包含一个点的地理位置。
ST_GeogFromText(wkt_string)	`GEOGRAPHY`	返回与传入的 WKT 字符串形式相对应的 GEOGRAPHY。
ST_ASText(geography)	`STRING`	返回传入的 GEOGRAPHY 的 WKT 字符串形式。
ST_Centroid(geography)	`GEOGRAPHY`	以单点 GEOGRAPHY 的形式返回传入的 GEOGRAPHY 的形心。
ST_ISValid(geography)	`BOOL`	返回传入的 GEOGRAPHY 是否有效。
ST_Intersects(geography_1, geography_2)	`BOOL`	返回传入的两个 GEOGRAPHY 是否有交集。
ST_Covers(geography_1, geography_2)	`BOOL`	返回 geography_1 是否完全包含 geography_2。如果 geography_2 中没有位于 geography_1 外部的点，返回 True。
ST_CoveredBy(geography_1, geography_2)	`BOOL`	返回 geography_2 是否完全包含 geography_1。如果 geography_1 中没有位于 geography_2 外部的点，返回 True。
ST_DWithin(geography_1, geography_2, distance)	`BOOL`	如果 geography_1 中至少有一个点与 geography_2 中的一个点的距离小于或等于 distance 参数（以米为单位）指定的距离，则返回 True。
ST_Distance(geography_1, geography_2)	`FLOAT`	返回两个非空 GEOGRAPHY 之间的最短距离（以米为单位）。
S2_CellIdFromPoint(point_geography)	`INT`	返回覆盖点 GEOGRAPHY 的 S2 单元ID。
S2_CoveringCellIds(geography)	`ARRAY<INT64>`	返回覆盖传入的 GEOGRAPHY 的 S2 单元 ID 的数组。

示例

```
nebula> RETURN ST_ASText(ST_Point(1,1));
+--------------------------+
| ST_ASText(ST_Point(1,1)) |
+--------------------------+
| "POINT(1 1)"             |
+--------------------------+

nebula> RETURN ST_ASText(ST_GeogFromText("POINT(3 8)"));
+----------------------------------------+
| ST_ASText(ST_GeogFromText("POINT(3 8)")) |
+----------------------------------------+
| "POINT(3 8)"                           |
+----------------------------------------+

nebula> RETURN ST_ASTEXT(ST_Centroid(ST_GeogFromText("LineString(0 1,1 0)")));
+-----------------------------------------------------------+
| ST_ASTEXT(ST_Centroid(ST_GeogFromText("LineString(0 1,1 0)"))) |
+-----------------------------------------------------------+
| "POINT(0.5000380800773782 0.5000190382261059)"            |
+-----------------------------------------------------------+

nebula> RETURN ST_ISValid(ST_GeogFromText("POINT(3 8)"));
+----------------------------------------+
| ST_ISValid(ST_GeogFromText("POINT(3 8)")) |
+----------------------------------------+
| true                                   |
+----------------------------------------+

nebula> RETURN ST_Intersects(ST_GeogFromText("LineString(0 1,1 0)"),ST_GeogFromText("LineString(0 0,1 1)"));
+-------------------------------------------------------------------------------------------+
| ST_Intersects(ST_GeogFromText("LineString(0 1,1 0)"),ST_GeogFromText("LineString(0 0,1 1)")) |
+-------------------------------------------------------------------------------------------+
| true                                                                                      |
```


2022 vesoft Inc.

```
+-------------------------------------------------------------------------------------------+

nebula> RETURN ST_Covers(ST_GeogFromText("POLYGON((0 0,10 0,10 10,0 10,0 0))"),ST_Point(1,2));
+-----------------------------------------------------------------------------+
| ST_Covers(ST_GeogFromText("POLYGON((0 0,10 0,10 10,0 10,0 0))"),ST_Point(1,2)) |
+-----------------------------------------------------------------------------+
| true                                                                        |
+-----------------------------------------------------------------------------+

nebula> RETURN ST_CoveredBy(ST_Point(1,2),ST_GeogFromText("POLYGON((0 0,10 0,10 10,0 10,0 0))"));
+-----------------------------------------------------------------------------+
| ST_CoveredBy(ST_Point(1,2),ST_GeogFromText("POLYGON((0 0,10 0,10 10,0 10,0 0))")) |
+-----------------------------------------------------------------------------+
| true                                                                        |
+-----------------------------------------------------------------------------+

nebula> RETURN ST_dwithin(ST_GeogFromText("Point(0 0)"),ST_GeogFromText("Point(10 10)"),20000000000.0);
+-----------------------------------------------------------------------------+
| ST_dwithin(ST_GeogFromText("Point(0 0)"),ST_GeogFromText("Point(10 10)"),20000000000) |
+-----------------------------------------------------------------------------+
| true                                                                        |
+-----------------------------------------------------------------------------+

nebula> RETURN ST_Distance(ST_GeogFromText("Point(0 0)"),ST_GeogFromText("Point(10 10)"));
+-----------------------------------------------------------------+
| ST_Distance(ST_GeogFromText("Point(0 0)"),ST_GeogFromText("Point(10 10)")) |
+-----------------------------------------------------------------+
| 1.5685230187677438e+06                                          |
+-----------------------------------------------------------------+

nebula> RETURN S2_CellIdFromPoint(ST_GeogFromText("Point(1 1)"));
+-------------------------------------------------+
| S2_CellIdFromPoint(ST_GeogFromText("Point(1 1)")) |
+-------------------------------------------------+
| 1153277837650709461                             |
+-------------------------------------------------+

nebula> RETURN S2_CoveringCellIds(ST_GeogFromText("POLYGON((0 1, 1 2, 2 3, 0 1))"));
+-----------------------------------------------------------------------------------------------
```

4.5.14 自定义函数

openCypher 兼容性

Nebula Graph 3.1.0 不支持自定义函数（UDF）和存储过程。

最后更新: June 20, 2023

2022 vesoft Inc.

4.6 通用查询语句

4.6.1 MATCH

`MATCH` 语句提供基于模式（pattern）匹配的搜索功能。

一个 `MATCH` 语句定义了一个搜索模式，用该模式匹配存储在 Nebula Graph 中的数据，然后用 `RETURN` 子句检索数据。

本文示例使用测试数据集 basketballplayer 进行演示。

语法

与 `GO` 或 `LOOKUP` 等其他查询语句相比， `MATCH` 的语法更灵活。`MATCH` 语句采用的路径类型是 `trail`，即遍历时只有点可以重复，边不可以重复。

`MATCH` 语法如下：

```
MATCH <pattern> [<clause_1>] RETURN <output> [<clause_2>];
```

- `pattern`： pattern 的详细说明请参见模式。`MATCH` 语句支持匹配一个或多个模式，多个模式之间用英文逗号（,）分隔。例如 `(a)-[]->(b),(c)-[]->(d)`。
- `clause_1`： 支持 `WHERE`、`WITH`、`UNWIND`、`OPTIONAL MATCH` 子句，也可以使用 `MATCH` 作为子句。
- `output`： 定义需要返回的输出。可以使用 `AS` 设置输出的别名。
- `clause_2`： 支持 `ORDER BY`、`LIMIT` 子句。

注意事项

除以下三种情况之外，请确保 `MATCH` 语句有至少一个索引可用。

- `MATCH` 语句中 `WHERE` 子句使用 id() 函数指定了点的 VID，不需要创建索引即可执行。
- 当遍历所有点边时，例如 `MATCH (v) RETURN v LIMIT N`，不需要创建索引，但必须使用 `LIMIT` 限制输出结果数量。
- 当遍历指定 Tag 的点或指定 Edge Type 的边时，例如 `MATCH (v:player) RETURN v LIMIT N`，不需要创建索引，但必须使用 `LIMIT` 限制输出结果数量。

> **Note**
>
> 目前 MATCH 语句无法查询到悬挂边。

> 从3.0.0版本兼容性
>
> 从 3.0.0 版本开始，为了区别不同 Tag 的属性，返回属性时必须额外指定 Tag 名称。即从 `RETURN <变量名>.<属性名>` 改为 `RETURN <变量名>.<Tag名>.<属性名>`。

示例

创建索引

```
# 在 Tag player 的 name 属性和 Edge type follow 上创建索引。
nebula> CREATE TAG INDEX IF NOT EXISTS name ON player(name(20));
nebula> CREATE EDGE INDEX IF NOT EXISTS follow_index on follow();

# 重建索引使其生效。
nebula> REBUILD TAG INDEX name;
+------------+
| New Job Id |
+------------+
| 121        |
```


2022 vesoft Inc.

```
+------------+

nebula> REBUILD EDGE INDEX follow_index;
+------------+
| New Job Id |
+------------+
| 122        |
+------------+

# 确认重建索引成功。
nebula> SHOW JOB 121;
+---------------+---------------------+------------+----------------------------+----------------------------+-------------+
| Job Id(TaskId) | Command(Dest)       | Status     | Start Time                 | Stop Time                  | Error Code  |
+---------------+---------------------+------------+----------------------------+----------------------------+-------------+
| 121           | "REBUILD_TAG_INDEX" | "FINISHED" | 2021-05-27T02:18:02.000000 | 2021-05-27T02:18:02.000000 | "SUCCEEDED" |
| 0             | "storaged1"         | "FINISHED" | 2021-05-27T02:18:02.000000 | 2021-05-27T02:18:02.000000 | "SUCCEEDED" |
| 1             | "storaged0"         | "FINISHED" | 2021-05-27T02:18:02.000000 | 2021-05-27T02:18:02.000000 | "SUCCEEDED" |
| 2             | "storaged2"         | "FINISHED" | 2021-05-27T02:18:02.000000 | 2021-05-27T02:18:02.000000 | "SUCCEEDED" |
+---------------+---------------------+------------+----------------------------+----------------------------+-------------+

nebula> SHOW JOB 122;
+---------------+---------------------+------------+----------------------------+----------------------------+-------------+
| Job Id(TaskId) | Command(Dest)       | Status     | Start Time                 | Stop Time                  | Error Code  |
+---------------+---------------------+------------+----------------------------+----------------------------+-------------+
| 122           | "REBUILD_EDGE_INDEX" | "FINISHED" | 2021-05-27T02:18:11.000000 | 2021-05-27T02:18:11.000000 | "SUCCEEDED" |
| 0             | "storaged1"         | "FINISHED" | 2021-05-27T02:18:11.000000 | 2021-05-27T02:18:21.000000 | "SUCCEEDED" |
| 1             | "storaged0"         | "FINISHED" | 2021-05-27T02:18:11.000000 | 2021-05-27T02:18:21.000000 | "SUCCEEDED" |
| 2             | "storaged2"         | "FINISHED" | 2021-05-27T02:18:11.000000 | 2021-05-27T02:18:21.000000 | "SUCCEEDED" |
+---------------+---------------------+------------+----------------------------+----------------------------+-------------+
```

匹配点

> 历史版本兼容性
>
> 从 Nebula Graph 3.0.0 开始，支持 `MATCH (v) RETURN v LIMIT n`，不需要创建索引；但是必须使用 `LIMIT` 限制输出结果数量。
>
> 不可以直接执行 `MATCH (v) RETURN v`。

用户可以在一对括号中使用自定义变量来表示模式中的点。例如 `(v)`。

```
nebula> MATCH (v) \
        RETURN v \
        LIMIT 3;
+-------------------------------------------------------+
| v                                                     |
+-------------------------------------------------------+
| ("player102" :player{age: 33, name: "LaMarcus Aldridge"}) |
| ("player106" :player{age: 25, name: "Kyle Anderson"})     |
| ("player115" :player{age: 40, name: "Kobe Bryant"})       |
+-------------------------------------------------------+
```

匹配 **TAG**

> 历史版本兼容性
>
> 在 Nebula Graph 3.0.0 之前，匹配 Tag 的前提是 Tag 本身有索引或者 Tag 的某个属性有索引，否则，用户无法基于该 Tag 执行 `MATCH` 语句。从 Nebula Graph 3.0.0 开始，匹配 Tag 可以不创建索引，但需要使用 `LIMIT` 限制输出结果数量。

用户可以在点的右侧用 `:<tag_name>` 表示模式中的 Tag。

```
nebula> MATCH (v:player) \
        RETURN v \
        LIMIT 3;
+-------------------------------------------------------+
| v                                                     |
+-------------------------------------------------------+
| ("player102" :player{age: 33, name: "LaMarcus Aldridge"}) |
| ("player106" :player{age: 25, name: "Kyle Anderson"})     |
| ("player115" :player{age: 40, name: "Kobe Bryant"})       |
+-------------------------------------------------------+
...
```

需要匹配拥有多个 Tag 的点，可以用英文冒号（`:`）。

2022 vesoft Inc.

Note

匹配多个 Tag 的点时，不支持进行属性过滤。

例如，不支持 `match (v1:player:team) where v1.player.name=="Tim Duncan" return v1 limit 10;` 。

```
nebula> CREATE TAG actor (name string, age int);
nebula> INSERT VERTEX actor(name, age) VALUES "player100":("Tim Duncan", 42);
nebula> MATCH (v:player:actor) \
        RETURN v \
        LIMIT 10;
+-------------------------------------------------------------------------------------+
| v                                                                                   |
+-------------------------------------------------------------------------------------+
| ("player100" :actor{age: 42, name: "Tim Duncan"} :player{age: 42, name: "Tim Duncan"}) |
+-------------------------------------------------------------------------------------+
```

匹配点的属性

用户可以在 Tag 的右侧用 `{<prop_name>: <prop_value>}` 表示模式中点的属性。

```
# 使用属性 name 搜索匹配的点。
nebula> MATCH (v:player{name:"Tim Duncan"}) \
        RETURN v;
+-------------------------------------------------+
| v                                               |
+-------------------------------------------------+
| ("player100" :player{age: 42, name: "Tim Duncan"}) |
+-------------------------------------------------+
```

使用 `WHERE` 子句也可以实现相同的操作：

```
nebula> MATCH (v:player) \
        WHERE v.player.name == "Tim Duncan" \
        RETURN v;
+-------------------------------------------------+
| v                                               |
+-------------------------------------------------+
| ("player100" :player{age: 42, name: "Tim Duncan"}) |
+-------------------------------------------------+
```

openCypher 兼容性

在 openCypher 9 中，`=` 是相等运算符，在 nGQL 中，`==` 是相等运算符，`=` 是赋值运算符。

匹配点 ID

用户可以使用点 ID 去匹配点。`id()` 函数可以检索点的 ID。

```
nebula> MATCH (v) \
        WHERE id(v) == 'player101' \
        RETURN v;
+-------------------------------------------------+
| v                                               |
+-------------------------------------------------+
| ("player101" :player{age: 36, name: "Tony Parker"}) |
+-------------------------------------------------+
```

要匹配多个点的 ID，可以用 `WHERE id(v) IN [vid_list]` 。

```
nebula> MATCH (v:player { name: 'Tim Duncan' })--(v2) \
        WHERE id(v2) IN ["player101", "player102"] \
        RETURN v2;
+-------------------------------------------------------+
| v2                                                    |
+-------------------------------------------------------+
| ("player101" :player{age: 36, name: "Tony Parker"})   |
| ("player101" :player{age: 36, name: "Tony Parker"})   |
| ("player102" :player{age: 33, name: "LaMarcus Aldridge"}) |
+-------------------------------------------------------+
```

匹配连接的点

用户可以使用 `--` 符号表示两个方向的边，并匹配这些边连接的点。

2022 vesoft Inc.

版本兼容性

在 nGQL 1.x 中，`--` 符号用于行内注释，从 nGQL 2.x 起，`--` 符号表示出边或入边，不再用于注释。

```
nebula> MATCH (v:player{name:"Tim Duncan"})--(v2:player) \
        RETURN v2.player.name AS Name;
+--------------------+
| Name               |
+--------------------+
| "Manu Ginobili"    |
| "Manu Ginobili"    |
| "Dejounte Murray"  |
...
```

用户可以在 `--` 符号上增加 < 或 > 符号指定边的方向。

```
# -->表示边从 v 开始，指向 v2。对于点 v 来说是出边，对于点 v2 来说是入边。
nebula> MATCH (v:player{name:"Tim Duncan"})-->(v2:player) \
        RETURN v2.player.name AS Name;
+----------------+
| Name           |
+----------------+
| "Tony Parker"  |
| "Manu Ginobili" |
+----------------+
```

如果需要判断目标点，可以使用 `CASE` 表达式。

```
nebula> MATCH (v:player{name:"Tim Duncan"})--(v2) \
        RETURN \
        CASE WHEN v2.team.name IS NOT NULL \
        THEN v2.team.name  \
        WHEN v2.player.name IS NOT NULL \
        THEN v2.player.name END AS Name;

+--------------------+
| Name               |
+--------------------+
| "Manu Ginobili"    |
| "Manu Ginobili"    |
| "Spurs"            |
| "Dejounte Murray"  |
...
```

如果需要扩展模式，可以增加更多点和边。

```
nebula> MATCH (v:player{name:"Tim Duncan"})-->(v2)<--(v3) \
        RETURN v3.player.name AS Name;
+--------------------+
| Name               |
+--------------------+
| "Dejounte Murray"  |
| "LaMarcus Aldridge" |
| "Marco Belinelli"  |
...
```

如果不需要引用点，可以省略括号中表示点的变量。

```
nebula> MATCH (v:player{name:"Tim Duncan"})-->()<--(v3) \
        RETURN v3.player.name AS Name;
+--------------------+
| Name               |
+--------------------+
| "Dejounte Murray"  |
| "LaMarcus Aldridge" |
| "Marco Belinelli"  |
...
```

匹配路径

连接起来的点和边构成了路径。用户可以使用自定义变量命名路径。

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-->(v2) \
        RETURN p;
+-------------------------------------------------------------------------------------------------------------------------------------------+
| p                                                                                                                                         |
+-------------------------------------------------------------------------------------------------------------------------------------------+
| <("player100" :player{age: 42, name: "Tim Duncan"})-[:serve@0 {end_year: 2016, start_year: 1997}]->("team204" :team{name: "Spurs"})>      |
| <("player100" :player{age: 42, name: "Tim Duncan"})-[:follow@0 {degree: 95}]->("player101" :player{age: 36, name: "Tony Parker"})>        |
```


2022 vesoft Inc.

```
| <("player100" :player{age: 42, name: "Tim Duncan"})-[:follow@0 {degree: 95}]->("player125" :player{age: 41, name: "Manu Ginobili"})> |
+-------------------------------------------------------------------------------------------------------------------------------------+
```

openCypher 兼容性

在 nGQL 中，`@`符号表示边的 rank，在 openCypher 中，没有 rank 概念。

匹配边

历史版本兼容性

在 Nebula Graph 3.0.0 之前，匹配边的前提是边本身有对应属性的索引，否则，用户无法基于边执行 `MATCH` 语句。从 Nebula Graph 3.0.0 开始，匹配边可以不创建索引，但需要使用 `LIMIT` 限制输出结果数量，并且必须指定边的方向。

```
nebula> MATCH ()<-[e]-() \
        RETURN e \
        LIMIT 3;
+--------------------------------------------------+
| e                                                |
+--------------------------------------------------+
| [:follow "player101"->"player102" @0 {degree: 90}] |
| [:follow "player103"->"player102" @0 {degree: 70}] |
| [:follow "player135"->"player102" @0 {degree: 80}] |
+--------------------------------------------------+
```

匹配 EDGE TYPE

和点一样，用户可以用 `:<edge_type>` 表示模式中的 Edge type，例如 `-[e:follow]-`。

历史版本兼容性

在 Nebula Graph 3.0.0 之前，匹配 Edge Type 的前提是 Edge type 本身有对应属性的索引，否则，用户无法基于 Edge Type 执行 `MATCH` 语句。从 Nebula Graph 3.0.0 开始，匹配 Edge Type 可以不创建索引，但需要使用 `LIMIT` 限制输出结果数量，并且必须指定边的方向。

```
nebula> MATCH ()-[e:follow]->() \
        RETURN e \
        limit 3;
+--------------------------------------------------+
| e                                                |
+--------------------------------------------------+
| [:follow "player102"->"player100" @0 {degree: 75}] |
| [:follow "player102"->"player101" @0 {degree: 75}] |
| [:follow "player129"->"player116" @0 {degree: 90}] |
+--------------------------------------------------+
```

匹配边的属性

Note

匹配边的属性的前提是 Edge type 本身有对应属性的索引，否则，用户无法执行 `MATCH` 语句匹配该属性。

用户可以用 `{<prop_name>: <prop_value>}` 表示模式中 Edge type 的属性，例如 `[e:follow{likeness:95}]`。

```
nebula> MATCH (v:player{name:"Tim Duncan"})-[e:follow{degree:95}]->(v2) \
        RETURN e;
+-------------------------------------------------------+
| e                                                     |
+-------------------------------------------------------+
| [:follow "player100"->"player101" @0 {degree: 95}]    |
| [:follow "player100"->"player125" @0 {degree: 95}]    |
+-------------------------------------------------------+
```

匹配多个 EDGE TYPE

使用 `|` 可以匹配多个 Edge type，例如 `[e:follow|:serve]`。第一个 Edge type 前的英文冒号（`:`）不可省略，后续 Edge type 前的英文冒号可以省略，例如 `[e:follow|serve]`。

2022 vesoft Inc.

> **Note**

同时匹配多个 Tag 和多个 Edge type 时，不支持进行属性过滤。

例如，不支持 `MATCH (v)-[e:follow|serve]->(v2) where v.player.name=="Tim Duncan" RETURN e limit 10;`，其中 `(v)` 代表匹配点的所有 Tag。

```
nebula> MATCH (v:player{name:"Tim Duncan"})-[e:follow|:serve]->(v2) \
        RETURN e;
+------------------------------------------------------------------------+
| e                                                                      |
+------------------------------------------------------------------------+
| [:follow "player100"->"player101" @0 {degree: 95}]                     |
| [:follow "player100"->"player125" @0 {degree: 95}]                     |
| [:serve "player100"->"team204" @0 {end_year: 2016, start_year: 1997}]  |
+------------------------------------------------------------------------+
```

匹配多条边

用户可以扩展模式，匹配路径中的多条边。

```
nebula> MATCH (v:player{name:"Tim Duncan"})-[]->(v2)<-[e:serve]-(v3) \
        RETURN v2, v3;
+--------------------------------+-----------------------------------------------------+
| v2                             | v3                                                  |
+--------------------------------+-----------------------------------------------------+
| ("team204" :team{name: "Spurs"}) | ("player104" :player{age: 32, name: "Marco Belinelli"})   |
| ("team204" :team{name: "Spurs"}) | ("player101" :player{age: 36, name: "Tony Parker"})       |
| ("team204" :team{name: "Spurs"}) | ("player102" :player{age: 33, name: "LaMarcus Aldridge"}) |
...
```

匹配定长路径

用户可以在模式中使用 `:<edge_type>*<hop>` 匹配定长路径。`hop` 必须是一个非负整数。

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[e:follow*2]->(v2) \
        RETURN DISTINCT v2 AS Friends;
+-------------------------------------------------------+
| Friends                                               |
+-------------------------------------------------------+
| ("player100" :player{age: 42, name: "Tim Duncan"})       |
| ("player125" :player{age: 41, name: "Manu Ginobili"})    |
| ("player102" :player{age: 33, name: "LaMarcus Aldridge"}) |
+-------------------------------------------------------+
```

如果 `hop` 为 0，模式会匹配路径上的起始点。

```
nebula> MATCH (v:player{name:"Tim Duncan"}) -[*0]-> (v2) \
        RETURN v2;
+-------------------------------------------------+
| v2                                              |
+-------------------------------------------------+
| ("player100" :player{age: 42, name: "Tim Duncan"}) |
+-------------------------------------------------+
```

> **Note**

如果想要通过描述多跳的边的过滤条件，比如 `-[e:follow*2]->`，这时候 `e` 不再是单条边时候的数据类型了，而是一列边组成的列表，例如： 以下语句可以运行但是没有返回数据，因为 `e` 是一个列表，没有 `.degree` 的属性。

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[e:follow*2]->(v2) \
        WHERE e.degree > 1 \
        RETURN DISTINCT v2 AS Friends;
```

这是正确的表达：

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[e:follow*2]->(v2) \
        WHERE ALL(e_ in e WHERE e_.degree > 0) \
        RETURN DISTINCT v2 AS Friends;
```

进一步，这是表达对多跳边的第一跳的边属性过滤的表达：

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[e:follow*2]->(v2) \
        WHERE e[0].degree > 98 \
        RETURN DISTINCT v2 AS Friends;
```


2022 vesoft Inc.

匹配变长路径

用户可以在模式中使用 `:<edge_type>*[minHop..maxHop]` 匹配变长路径。

Caution

如果未设置 `maxHop` 可能会导致 graph 服务 OOM，请谨慎执行该命令。

参数	说明
`minHop`	可选项。表示路径的最小长度。`minHop` 必须是一个非负整数，默认值为 1。
`maxHop`	可选项。表示路径的最大长度。`maxHop` 必须是一个非负整数，默认值为无穷大。

如果未指定 `minHop` 和 `maxHop`，仅设置了 `:<edge_type>*`，则二者都应用默认值，即 `minHop` 为 1，`maxHop` 为无穷大。

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[e:follow*]->(v2) \
        RETURN v2 AS Friends;
+-------------------------------------------------------+
| Friends                                               |
+-------------------------------------------------------+
| ("player125" :player{age: 41, name: "Manu Ginobili"})     |
| ("player101" :player{age: 36, name: "Tony Parker"})       |
...

nebula> MATCH p=(v:player{name:"Tim Duncan"})-[e:follow*1..3]->(v2) \
        RETURN v2 AS Friends;
+-------------------------------------------------------+
| Friends                                               |
+-------------------------------------------------------+
| ("player101" :player{age: 36, name: "Tony Parker"})       |
| ("player125" :player{age: 41, name: "Manu Ginobili"})     |
| ("player100" :player{age: 42, name: "Tim Duncan"})        |
...

nebula> MATCH p=(v:player{name:"Tim Duncan"})-[e:follow*1..]->(v2) \
        RETURN v2 AS Friends;
+-------------------------------------------------------+
| Friends                                               |
+-------------------------------------------------------+
| ("player125" :player{age: 41, name: "Manu Ginobili"})     |
| ("player101" :player{age: 36, name: "Tony Parker"})       |
| ("player100" :player{age: 42, name: "Tim Duncan"})        |
...
```

用户可以使用 `DISTINCT` 关键字聚合重复结果。

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[e:follow*1..3]->(v2:player) \
        RETURN DISTINCT v2 AS Friends, count(v2);
+-------------------------------------------------------+----------+
| Friends                                               | count(v2) |
+-------------------------------------------------------+----------+
| ("player102" :player{age: 33, name: "LaMarcus Aldridge"}) | 1         |
| ("player100" :player{age: 42, name: "Tim Duncan"})        | 4         |
| ("player101" :player{age: 36, name: "Tony Parker"})       | 3         |
| ("player125" :player{age: 41, name: "Manu Ginobili"})     | 3         |
+-------------------------------------------------------+----------+
```

如果 `minHop` 为 `0`，模式会匹配路径上的起始点。例如，与上个示例相比，下面的示例设置 `minHop` 为 `0`。此时，因为表示 `"Tim Duncan"` 的点是路径的起始点，所以它在结果集中的计数为 5，比在上个示例的结果中多计一次。

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[e:follow*0..3]->(v2:player) \
        RETURN DISTINCT v2 AS Friends, count(v2);
+-------------------------------------------------------+----------+
| Friends                                               | count(v2) |
+-------------------------------------------------------+----------+
| ("player102" :player{age: 33, name: "LaMarcus Aldridge"}) | 1         |
| ("player100" :player{age: 42, name: "Tim Duncan"})        | 5         |
| ("player125" :player{age: 41, name: "Manu Ginobili"})     | 3         |
| ("player101" :player{age: 36, name: "Tony Parker"})       | 3         |
+-------------------------------------------------------+----------+
```

匹配多个 **EDGE TYPE** 的变长路径

用户可以在变长或定长模式中指定多个 Edge type。`hop`、`minHop` 和 `maxHop` 对所有 Edge type 都生效。

2022 vesoft Inc.

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[e:follow|serve*2]->(v2) \
        RETURN DISTINCT v2;
+-------------------------------------------------------------+
| v2                                                          |
+-------------------------------------------------------------+
| ("team204" :team{name: "Spurs"})                            |
| ("player100" :player{age: 42, name: "Tim Duncan"})          |
| ("team215" :team{name: "Hornets"})                          |
| ("player125" :player{age: 41, name: "Manu Ginobili"})       |
| ("player102" :player{age: 33, name: "LaMarcus Aldridge"}) |
+-------------------------------------------------------------+
```

匹配多个模式

用户可以用英文逗号（,）分隔多个模式。

```
nebula> MATCH (v1:player{name:"Tim Duncan"}), (v2:team{name:"Spurs"}) \
        RETURN v1,v2;
+-----------------------------------------------+----------------------------------+
| v1                                            | v2                               |
+-----------------------------------------------+----------------------------------+
| ("player100" :player{age: 42, name: "Tim Duncan"}) | ("team204" :team{name: "Spurs"}) |
+-----------------------------------------------+----------------------------------+
```

多**MATCH**检索

不同的模式有不同的筛选条件时，可以使用多 `MATCH`，会返回模式完全匹配的行。

```
nebula> MATCH (m)-[]->(n) WHERE id(m)=="player100" \
        MATCH (n)-[]->(l) WHERE id(n)=="player125" \
        RETURN id(m),id(n),id(l);
+-------------+-------------+-------------+
| id(m)       | id(n)       | id(l)       |
+-------------+-------------+-------------+
| "player100" | "player125" | "team204"   |
| "player100" | "player125" | "player100" |
+-------------+-------------+-------------+
```

OPTIONAL MATCH检索

参见OPTIONAL MATCH。

> **Performance**

Nebula Graph 3.1.0 中 `MATCH` 语句的性能和资源占用得到了优化.但对性能要求较高时，仍建议使用 `GO`，`LOOKUP`，`|` 和 `FETCH` 等来替代 `MATCH`。

最后更新: June 20, 2023

2022 vesoft Inc.

4.6.2 OPTIONAL MATCH

`OPTIONAL MATCH` 通常与 `MATCH` 语句一起使用，作为 `MATCH` 语句的可选项去匹配命中的模式，如果没有命中对应的模式，对应的列返回 `NULL`。

openCypher 兼容性

本文操作仅适用于 nGQL 中的 openCypher 方式。

示例

`MATCH` 语句中使用 `OPTIONAL MATCH` 的示例如下：

```
nebula> MATCH (m)-[]->(n) WHERE id(m)=="player100" \
        OPTIONAL MATCH (n)-[]->(l) WHERE id(n)=="player125" \
        RETURN id(m),id(n),id(l);
+-------------+-------------+-------------+
| id(m)       | id(n)       | id(l)       |
+-------------+-------------+-------------+
| "player100" | "team204"   | __NULL__    |
| "player100" | "player101" | __NULL__    |
| "player100" | "player125" | "team204"   |
| "player100" | "player125" | "player100" |
+-------------+-------------+-------------+
```

而使用多 `MATCH`，不使用 `OPTIONAL MATCH` 时，会返回模式完全匹配的行。示例如下：

```
nebula> MATCH (m)-[]->(n) WHERE id(m)=="player100" \
        MATCH (n)-[]->(l) WHERE id(n)=="player125" \
        RETURN id(m),id(n),id(l);
+-------------+-------------+-------------+
| id(m)       | id(n)       | id(l)       |
+-------------+-------------+-------------+
| "player100" | "player125" | "team204"   |
| "player100" | "player125" | "player100" |
+-------------+-------------+-------------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.6.3 LOOKUP

`LOOKUP` 根据索引遍历数据。用户可以使用 `LOOKUP` 实现如下功能:

- 根据 `WHERE` 子句搜索特定数据。
- 通过 Tag 列出点：检索指定 Tag 的所有点 ID。
- 通过 Edge type 列出边：检索指定 Edge type 的所有边的起始点、目的点和 rank。
- 统计包含指定 Tag 的点或属于指定 Edge type 的边的数量。

OpenCypher 兼容性

本文操作仅适用于原生 nGQL。

注意事项

- 索引会导致写性能大幅降低（降低 90%甚至更多）。请不要随意在生产环境中使用索引，除非很清楚使用索引对业务的影响。
- 通过Explain命令查看选择的索引。

> 历史版本兼容性
>
> 在 2.5.0 版本之前，如果用 `LOOKUP` 语句基于指定属性查询时该属性没有索引，系统将报错，而不会使用其它索引。

前提条件

请确保 `LOOKUP` 语句有至少一个索引可用。

如果已经存在相关的点、边或属性，必须在新创建索引后重建索引，才能使其生效。

语法

```
LOOKUP ON {<vertex_tag> | <edge_type>}
[WHERE <expression> [AND <expression> ...]]
YIELD <return_list> [AS <alias>];

<return_list>
    <prop_name> [AS <col_alias>] [, <prop_name> [AS <prop_alias>] ...];
```

- `WHERE <expression>`：指定遍历的过滤条件，还可以结合布尔运算符 AND 和 OR 一起使用。详情请参见 WHERE。
- `YIELD`：定义需要返回的输出。详情请参见 `YIELD`。
- `AS`：设置别名。

WHERE 语句限制

在 `LOOKUP` 语句中使用 `WHERE` 子句，不支持如下操作:

- `$-` 和 `$^`。
- 在关系表达式中，不支持运算符两边都有字段名，例如 `tagName.prop1 > tagName.prop2`。
- 不支持运算表达式和函数表达式中嵌套 AliasProp 表达式。
- 不支持 XOR 运算符。
- 不支持除 `STARTS WITH` 之外的字符串操作。
- 不支持图模式

2022 vesoft Inc.

检索点

返回 Tag 为 `player` 且 `name` 为 `Tony Parker` 的点。

```
nebula> CREATE TAG INDEX IF NOT EXISTS index_player ON player(name(30), age);

nebula> REBUILD TAG INDEX index_player;
+------------+
| New Job Id |
+------------+
| 15         |
+------------+

nebula> LOOKUP ON player \
        WHERE player.name == "Tony Parker" \
        YIELD id(vertex);
+--------------+
| id(VERTEX)   |
+--------------+
| "player101"  |
+--------------+

nebula> LOOKUP ON player \
        WHERE player.name == "Tony Parker" \
        YIELD properties(vertex).name AS name, properties(vertex).age AS age;
+---------------+-----+
| name          | age |
+---------------+-----+
| "Tony Parker" | 36  |
+---------------+-----+

nebula> LOOKUP ON player \
        WHERE player.age  > 45 \
        YIELD id(vertex);
+-------------+
| id(VERTEX)  |
+-------------+
| "player144" |
| "player140" |
+-------------+

nebula> LOOKUP ON player \
        WHERE player.name STARTS WITH "B" \
        AND player.age IN [22,30] \
        YIELD properties(vertex).name, properties(vertex).age;
+-------------------------+------------------------+
| properties(VERTEX).name | properties(VERTEX).age |
+-------------------------+------------------------+
| "Ben Simmons"           | 22                     |
| "Blake Griffin"         | 30                     |
+-------------------------+------------------------+

nebula> LOOKUP ON player \
        WHERE player.name == "Kobe Bryant"\
        YIELD id(vertex) AS VertexID, properties(vertex).name AS name |\
        GO FROM $-.VertexID OVER serve \
        YIELD $-.name, properties(edge).start_year, properties(edge).end_year, properties($$).name;
+---------------+-----------------------------+---------------------------+---------------------+
| $-.name       | properties(EDGE).start_year | properties(EDGE).end_year | properties($$).name |
+---------------+-----------------------------+---------------------------+---------------------+
| "Kobe Bryant" | 1996                        | 2016                      | "Lakers"            |
+---------------+-----------------------------+---------------------------+---------------------+
```

检索边

返回 Edge type 为 `follow` 且 `degree` 为 `90` 的边。

```
nebula> CREATE EDGE INDEX IF NOT EXISTS index_follow ON follow(degree);

nebula> REBUILD EDGE INDEX index_follow;
+------------+
| New Job Id |
+------------+
| 62         |
+------------+

nebula> LOOKUP ON follow \
        WHERE follow.degree == 90 YIELD edge AS e;
+--------------------------------------------------+
| e                                                |
+--------------------------------------------------+
| [:follow "player109"->"player125" @0 {degree: 90}] |
| [:follow "player118"->"player120" @0 {degree: 90}] |
| [:follow "player118"->"player131" @0 {degree: 90}] |
...

nebula> LOOKUP ON follow \
        WHERE follow.degree == 90 \
```


2022 vesoft Inc.

```
        YIELD properties(edge).degree;
+------------------------+
| properties(EDGE).degree |
+------------------------+
| 90                     |
| 90                     |
...

nebula> LOOKUP ON follow \
        WHERE follow.degree == 60 \
        YIELD dst(edge) AS DstVID, properties(edge).degree AS Degree |\
        GO FROM $-.DstVID OVER serve \
        YIELD $-.DstVID, properties(edge).start_year, properties(edge).end_year, properties($$).name;
+-------------+-----------------------------+---------------------------+---------------------+
| $-.DstVID   | properties(EDGE).start_year | properties(EDGE).end_year | properties($$).name |
+-------------+-----------------------------+---------------------------+---------------------+
| "player105" | 2010                        | 2018                      | "Spurs"             |
| "player105" | 2009                        | 2010                      | "Cavaliers"         |
| "player105" | 2018                        | 2019                      | "Raptors"           |
+-------------+-----------------------------+---------------------------+---------------------+
```

通过 **Tag** 列出所有的对应的点/通过 **Edge type** 列出边

如果需要通过 Tag 列出所有的点，或通过 Edge type 列出边，则 Tag、Edge type 或属性上必须有至少一个索引。

例如一个 Tag `player` 有属性 `name` 和 `age`，为了遍历所有包含 Tag `player` 的点 ID，Tag `player` 、属性 `name` 或属性 `age` 中必须有一个已经创建索引。

- 查找所有 Tag 为 `player` 的点 VID。

```
nebula> CREATE TAG IF NOT EXISTS player(name string,age int);

nebula> CREATE TAG INDEX IF NOT EXISTS player_index on player();

nebula> REBUILD TAG INDEX player_index;
+------------+
| New Job Id |
+------------+
| 66         |
+------------+

nebula> INSERT VERTEX player(name,age) \
        VALUES "player100":("Tim Duncan", 42), "player101":("Tony Parker", 36);

# 列出所有的 player。类似于 MATCH (n:player) RETURN id(n) /*, n */。

nebula> LOOKUP ON player YIELD id(vertex);
+-------------+
| id(VERTEX)  |
+-------------+
| "player100" |
| "player101" |
...
```

- 查找 Edge type 为 `follow` 的所有边的信息。

```
nebula> CREATE EDGE IF NOT EXISTS follow(degree int);

nebula> CREATE EDGE INDEX IF NOT EXISTS follow_index on follow();

nebula> REBUILD EDGE INDEX follow_index;
+------------+
| New Job Id |
+------------+
| 88         |
+------------+

nebula> INSERT EDGE follow(degree) \
        VALUES "player100"->"player101":(95);

# 列出所有的 follow 边。类似于 MATCH (s)-[e:follow]->(d) RETURN id(s), rank(e), id(d) /*, type(e) */。

nebula)> LOOKUP ON follow YIELD edge AS e;
+-------------------------------------------------------+
| e                                                     |
+-------------------------------------------------------+
| [:follow "player105"->"player100" @0 {degree: 70}]  |
| [:follow "player105"->"player116" @0 {degree: 80}]  |
| [:follow "player109"->"player100" @0 {degree: 80}]  |
...
```


2022 vesoft Inc.

统计点或边

统计 Tag 为 `player` 的点和 Edge type 为 `follow` 的边。

```
nebula> LOOKUP ON player YIELD id(vertex)|\
        YIELD COUNT(*) AS Player_Number;
+---------------+
| Player_Number |
+---------------+
| 51            |
+---------------+

nebula> LOOKUP ON follow YIELD edge AS e| \
        YIELD COUNT(*) AS Follow_Number;
+---------------+
| Follow_Number |
+---------------+
| 81            |
+---------------+
```

Note

使用 `SHOW STATS` 命令也可以统计点和边。

最后更新: June 20, 2023

2022 vesoft Inc.

4.6.4 GO

`GO` 从给定起始点开始遍历图。`GO` 语句采用的路径类型是`walk`，即遍历时点和边都可以重复。

openCypher 兼容性

本文操作仅适用于原生 nGQL。

语法

```
GO [[<M> TO] <N> STEPS ] FROM <vertex_list>
OVER <edge_type_list> [{REVERSELY | BIDIRECT}]
[ WHERE <conditions> ]
YIELD [DISTINCT] <return_list>
[{ SAMPLE <sample_list> | <limit_by_list_clause> }]
[| GROUP BY {<col_name> | expression> | <position>} YIELD <col_name>]
[| ORDER BY <expression> [{ASC | DESC}]]
[| LIMIT [<offset>,] <number_rows>];

<vertex_list> ::=
    <vid> [, <vid> ...]

<edge_type_list> ::=
   <edge_type> [, <edge_type> ...]
   | *
```


2022 vesoft Inc.

```
<return_list> ::=
    <col_name> [AS <col_alias>] [, <col_name> [AS <col_alias>] ...]
```

- `<N> STEPS`：指定跳数。如果没有指定跳数，默认值 `N` 为 `1`。如果 `N` 为 `0`，Nebula Graph 不会检索任何边。
- `M TO N STEPS`：遍历 `M~N` 跳的边。如果 `M` 为 `0`，输出结果和 `M` 为 `1` 相同，即 `GO 0 TO 2` 和 `GO 1 TO 2` 是相同的。
- `<vertex_list>`：用逗号分隔的点 ID 列表，或特殊的引用符 `$-.id`。详情参见管道符。
- `<edge_type_list>`：遍历的 Edge type 列表。
- `REVERSELY | BIDIRECT`：默认情况下检索的是 `<vertex_list>` 的出边（正向），`REVERSELY` 表示反向，即检索入边；`BIDIRECT` 为双向，即检索正向和反向，通过返回 `<edge_type>._type` 字段判断方向，其正数为正向，负数为反向。
- `WHERE <conditions>`：指定遍历的过滤条件。用户可以在起始点、目的点和边使用 `WHERE` 子句，还可以结合 `AND`、`OR`、`NOT`、`XOR` 一起使用。详情参见 WHERE。

  > **Note**
  >
  > 遍历多个 Edge type 时，`WHERE` 子句有一些限制。例如不支持 `WHERE edge1.prop1 > edge2.prop2`。

- `YIELD [DISTINCT] <return_list>`：定义需要返回的输出。`<return_list>` 建议使用 Schema 函数，当前支持 `src(edge)`、`dst(edge)`、`type(edge)` 等，暂不支持嵌套函数。详情参见 YIELD。
- `SAMPLE <sample_list>`：用于在结果集中取样。详情参见 SAMPLE。
- `<limit_by_list_clause>`：用于在遍历过程中逐步限制输出数量。详情参见 LIMIT。
- `GROUP BY`：根据指定属性的值将输出分组。详情参见 GROUP BY。分组后需要再次使用 `YIELD` 定义需要返回的输出。
- `ORDER BY`：指定输出结果的排序规则。详情参见 ORDER BY。

  > **Note**
  >
  > 没有指定排序规则时，输出结果的顺序不是固定的。

- `LIMIT [<offset>,] <number_rows>]`：限制输出结果的行数。详情参见 LIMIT。

示例

```
# 返回 player102 所属队伍。
nebula> GO FROM "player102" OVER serve YIELD dst(edge);
+-----------+
| dst(EDGE) |
+-----------+
| "team203" |
| "team204" |
+-----------+
```

```
# 返回距离 player102 两跳的朋友。
nebula> GO 2 STEPS FROM "player102" OVER follow YIELD dst(edge);
+-------------+
| dst(EDGE)   |
+-------------+
| "player101" |
| "player125" |
| "player100" |
| "player102" |
| "player125" |
+-------------+
```

```
# 添加过滤条件。
nebula> GO FROM "player100", "player102" OVER serve \
        WHERE properties(edge).start_year > 1995 \
        YIELD DISTINCT properties($$).name AS team_name, properties(edge).start_year AS start_year, properties($^).name AS player_name;

+-----------------+------------+--------------------+
```


2022 vesoft Inc.

```
| team_name       | start_year | player_name         |
+-----------------+------------+---------------------+
| "Spurs"         | 1997       | "Tim Duncan"        |
| "Trail Blazers" | 2006       | "LaMarcus Aldridge" |
| "Spurs"         | 2015       | "LaMarcus Aldridge" |
+-----------------+------------+---------------------+
```

```
# 遍历多个 Edge type。属性没有值时，会显示 UNKNOWN_PROP。
nebula> GO FROM "player100" OVER follow, serve \
        YIELD properties(edge).degree, properties(edge).start_year;
+-------------------------+-----------------------------+
| properties(EDGE).degree | properties(EDGE).start_year |
+-------------------------+-----------------------------+
| 95                      | UNKNOWN_PROP                |
| 95                      | UNKNOWN_PROP                |
| UNKNOWN_PROP            | 1997                        |
+-------------------------+-----------------------------+
```

```
# 返回 player100 入方向的邻居点。
nebula> GO FROM "player100" OVER follow REVERSELY \
        YIELD src(edge) AS destination;
+-------------+
| destination |
+-------------+
| "player101" |
| "player102" |
...

# 该 MATCH 查询与上一个 GO 查询具有相同的语义。
nebula> MATCH (v)<-[e:follow]- (v2) WHERE id(v) == 'player100' \
        RETURN id(v2) AS destination;
+-------------+
| destination |
+-------------+
| "player101" |
| "player102" |
...
```

```
# 查询 player100 的朋友和朋友所属队伍。
nebula> GO FROM "player100" OVER follow REVERSELY \
        YIELD src(edge) AS id | \
        GO FROM $-.id OVER serve \
        WHERE properties($^).age > 20 \
        YIELD properties($^).name AS FriendOf, properties($$).name AS Team;
+--------------------+----------------+
| FriendOf           | Team           |
+--------------------+----------------+
| "Boris Diaw"       | "Spurs"        |
| "Boris Diaw"       | "Jazz"         |
| "Boris Diaw"       | "Suns"         |
...

# 该 MATCH 查询与上一个 GO 查询具有相同的语义。
nebula> MATCH (v)<-[e:follow]- (v2)-[e2:serve]->(v3)  \
        WHERE id(v) == 'player100' \
        RETURN v2.player.name AS FriendOf, v3.team.name AS Team;
+--------------------+----------------+
| FriendOf           | Team           |
+--------------------+----------------+
| "Boris Diaw"       | "Spurs"        |
| "Boris Diaw"       | "Jazz"         |
| "Boris Diaw"       | "Suns"         |
...
```

```
# 查询 player100 1~2 跳内的朋友。
nebula> GO 1 TO 2 STEPS FROM "player100" OVER follow \
        YIELD dst(edge) AS destination;
+-------------+
| destination |
+-------------+
| "player101" |
| "player125" |
...

# 该 MATCH 查询与上一个 GO 查询具有相同的语义。
nebula> MATCH (v) -[e:follow*1..2]->(v2) \
        WHERE id(v) == "player100" \
        RETURN id(v2) AS destination;
+-------------+
| destination |
+-------------+
| "player100" |
| "player102" |
...
```

```
# 根据年龄分组。
nebula> GO 2 STEPS FROM "player100" OVER follow \
        YIELD src(edge) AS src, dst(edge) AS dst, properties($$).age AS age \
        | GROUP BY $-.dst \
        YIELD $-.dst AS dst, collect_set($-.src) AS src, collect($-.age) AS age;
```


2022 vesoft Inc.

```
+-------------+----------------------------+----------+
| dst         | src                        | age      |
+-------------+----------------------------+----------+
| "player125" | ["player101"]              | [41]     |
| "player100" | ["player125", "player101"] | [42, 42] |
| "player102" | ["player101"]              | [33]     |
+-------------+----------------------------+----------+
```

```
# 分组并限制输出结果的行数。
nebula> $a = GO FROM "player100" OVER follow YIELD src(edge) AS src, dst(edge) AS dst; \
        GO 2 STEPS FROM $a.dst OVER follow \
        YIELD $a.src AS src, $a.dst, src(edge), dst(edge) \
        | ORDER BY $-.src | OFFSET 1 LIMIT 2;
+-------------+-------------+-------------+-------------+
| src         | $a.dst      | src(EDGE)   | dst(EDGE)   |
+-------------+-------------+-------------+-------------+
| "player100" | "player125" | "player100" | "player101" |
| "player100" | "player101" | "player100" | "player125" |
+-------------+-------------+-------------+-------------+
```

```
# 在多个边上通过 IS NOT EMPTY 进行判断。
nebula> GO FROM "player100" OVER follow WHERE properties($$).name IS NOT EMPTY YIELD dst(edge);
+-------------+
| dst(EDGE)   |
+-------------+
| "player125" |
| "player101" |
+-------------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.6.5 FETCH

`FETCH` 可以获取指定点或边的属性值。

openCypher 兼容性

本文操作仅适用于原生 nGQL。

获取点的属性值

语法

```
FETCH PROP ON {<tag_name>[, tag_name ...] | *}
<vid> [, vid ...]
YIELD <return_list> [AS <alias>];
```

参数	说明
`tag_name`	Tag 名称。
`*`	表示当前图空间中的所有 Tag。
`vid`	点 ID。
`YIELD`	定义需要返回的输出。详情请参见 `YIELD` 。
`AS`	设置别名。

基于 **TAG** 获取点的属性值

在 `FETCH` 语句中指定 Tag 获取对应点的属性值。

```
nebula> FETCH PROP ON player "player100" YIELD properties(vertex);
+------------------------------+
| properties(VERTEX)           |
+------------------------------+
| {age: 42, name: "Tim Duncan"} |
+------------------------------+
```

获取点的指定属性值

使用 `YIELD` 子句指定返回的属性。

```
nebula> FETCH PROP ON player "player100" \
        YIELD properties(vertex).name AS name;
+-------------+
| name        |
+-------------+
| "Tim Duncan" |
+-------------+
```

获取多个点的属性值

指定多个点 ID 获取多个点的属性值，点之间用英文逗号（,）分隔。

```
nebula> FETCH PROP ON player "player101", "player102", "player103" YIELD properties(vertex);
+--------------------------------------+
| properties(VERTEX)                   |
+--------------------------------------+
| {age: 33, name: "LaMarcus Aldridge"} |
| {age: 40, name: "Tony Parker"}       |
| {age: 32, name: "Rudy Gay"}          |
+--------------------------------------+
```

基于多个 **TAG** 获取点的属性值

在 `FETCH` 语句中指定多个 Tag 获取属性值。Tag 之间用英文逗号（,）分隔。

```
# 创建新 Tag t1。
nebula> CREATE TAG IF NOT EXISTS t1(a string, b int);
```


2022 vesoft Inc.

```
# 为点 player100 添加 Tag t1。
nebula> INSERT VERTEX t1(a, b) VALUES "player100":("Hello", 100);

# 基于 Tag player 和 t1 获取点 player100 上的属性值。
nebula> FETCH PROP ON player, t1 "player100" YIELD vertex AS v;
+------------------------------------------------------------------------+
| v                                                                      |
+------------------------------------------------------------------------+
| ("player100" :player{age: 42, name: "Tim Duncan"} :t1{a: "Hello", b: 100}) |
+------------------------------------------------------------------------+
```

用户可以在 `FETCH` 语句中组合多个 Tag 和多个点。

```
nebula> FETCH PROP ON player, t1 "player100", "player103" YIELD vertex AS v;
+------------------------------------------------------------------------+
| v                                                                      |
+------------------------------------------------------------------------+
| ("player100" :player{age: 42, name: "Tim Duncan"} :t1{a: "Hello", b: 100}) |
| ("player103" :player{age: 32, name: "Rudy Gay"})                       |
+------------------------------------------------------------------------+
```

在所有标签中获取点的属性值

在 `FETCH` 语句中使用 `*` 获取当前图空间所有标签里，点的属性值。

```
nebula> FETCH PROP ON * "player100", "player106", "team200" YIELD vertex AS v;
+------------------------------------------------------------------------+
| v                                                                      |
+------------------------------------------------------------------------+
| ("player100" :player{age: 42, name: "Tim Duncan"} :t1{a: "Hello", b: 100}) |
| ("player106" :player{age: 25, name: "Kyle Anderson"})                  |
| ("team200" :team{name: "Warriors"})                                    |
+------------------------------------------------------------------------+
```

获取边的属性值

语法

```
FETCH PROP ON <edge_type> <src_vid> -> <dst_vid>[@<rank>] [, <src_vid> -> <dst_vid> ...]
YIELD <output>;
```

参数	说明
`edge_type`	Edge type 名称。
`src_vid`	起始点 ID，表示边的起点。
`dst_vid`	目的点 ID，表示边的终点。
`rank`	边的 rank。可选参数，默认值为 `0`。起始点、目的点、Edge type 和 rank 可以唯一确定一条边。
`YIELD`	定义需要返回的输出。详情请参见 `YIELD`。

获取边的所有属性值

```
# 获取连接 player100 和 team204 的边 serve 的所有属性值。
nebula> FETCH PROP ON serve "player100" -> "team204" YIELD properties(edge);
+-----------------------------------+
| properties(EDGE)                  |
+-----------------------------------+
| {end_year: 2016, start_year: 1997} |
+-----------------------------------+
```

获取边的指定属性值

使用 `YIELD` 子句指定返回的属性。

```
nebula> FETCH PROP ON serve "player100" -> "team204"    \
        YIELD properties(edge).start_year;
+----------------------------+
| properties(EDGE).start_year |
+----------------------------+
| 1997                       |
+----------------------------+
```


2022 vesoft Inc.

获取多条边的属性值

指定多个边模式 (`<src_vid> -> <dst_vid>[@<rank>]`) 获取多个边的属性值。模式之间用英文逗号（,）分隔。

```
nebula> FETCH PROP ON serve "player100" -> "team204", "player133" -> "team202" YIELD edge AS e;
+------------------------------------------------------------------------+
| e                                                                      |
+------------------------------------------------------------------------+
| [:serve "player100"->"team204" @0 {end_year: 2016, start_year: 1997}] |
| [:serve "player133"->"team202" @0 {end_year: 2011, start_year: 2002}] |
+------------------------------------------------------------------------+
```

基于 **RANK** 获取属性值

如果有多条边，起始点、目的点和 Edge type 都相同，可以通过指定 rank 获取正确的边属性值。

```
# 插入不同属性值、不同 rank 的边。
nebula> insert edge serve(start_year,end_year) \
        values "player100"->"team204"@1:(1998, 2017);

nebula> insert edge serve(start_year,end_year) \
        values "player100"->"team204"@2:(1990, 2018);

# 默认返回 rank 为 0 的边。
nebula> FETCH PROP ON serve "player100" -> "team204" YIELD edge AS e;
+------------------------------------------------------------------------+
| e                                                                      |
+------------------------------------------------------------------------+
| [:serve "player100"->"team204" @0 {end_year: 2016, start_year: 1997}] |
+------------------------------------------------------------------------+

# 要获取 rank 不为 0 的边，请在 FETCH 语句中设置 rank。
nebula> FETCH PROP ON serve "player100" -> "team204"@1 YIELD edge AS e;
+------------------------------------------------------------------------+
| e                                                                      |
+------------------------------------------------------------------------+
| [:serve "player100"->"team204" @1 {end_year: 2017, start_year: 1998}] |
+------------------------------------------------------------------------+
```

复合语句中使用 **FETCH**

将 `FETCH` 与原生 nGQL 结合使用是一种常见的方式，例如和 `GO` 一起。

```
# 返回从点 player101 开始的 follow 边的 degree 值。
nebula> GO FROM "player101" OVER follow \
        YIELD src(edge) AS s, dst(edge) AS d \
        | FETCH PROP ON follow $-.s -> $-.d \
        YIELD properties(edge).degree;
+-------------------------+
| properties(EDGE).degree |
+-------------------------+
| 95                      |
| 90                      |
| 95                      |
+-------------------------+
```

用户也可以通过自定义变量构建类似的查询。

```
nebula> $var = GO FROM "player101" OVER follow \
        YIELD src(edge) AS s, dst(edge) AS d; \
        FETCH PROP ON follow $var.s -> $var.d \
        YIELD properties(edge).degree;
+-------------------------+
| properties(EDGE).degree |
+-------------------------+
| 95                      |
| 90                      |
| 95                      |
+-------------------------+
```

更多复合语句的详情，请参见复合查询（子句结构）。

最后更新: June 20, 2023

2022 vesoft Inc.

4.6.6 SHOW

SHOW CHARSET

`SHOW CHARSET` 语句显示当前的字符集。

目前可用的字符集为 `utf8` 和 `utf8mb4` 。默认字符集为 `utf8` 。Nebula Graph 扩展 `uft8` 支持四字节字符，因此 `utf8` 和 `utf8mb4` 是等价的。

语法

```
SHOW CHARSET;
```

示例

```
nebula> SHOW CHARSET;
+---------+-----------------+-------------------+--------+
| Charset | Description     | Default collation | Maxlen |
+---------+-----------------+-------------------+--------+
| "utf8"  | "UTF-8 Unicode" | "utf8_bin"        | 4      |
+---------+-----------------+-------------------+--------+
```

参数	说明
`Charset`	字符集名称。
`Description`	字符集说明。
`Default collation`	默认排序规则。
`Maxlen`	存储一个字符所需的最大字节数。

最后更新: June 20, 2023

2022 vesoft Inc.

SHOW COLLATION

`SHOW COLLATION` 语句显示当前的排序规则。

目前可用的排序规则为 `utf8_bin` 、`utf8_general_ci` 、`utf8mb4_bin` 和 `utf8mb4_general_ci` 。

- 当字符集为 `utf8` ，默认排序规则为 `utf8_bin` 。
- 当字符集为 `utf8mb4` ，默认排序规则为 `utf8mb4_bin` 。
- `utf8_general_ci` 和 `utf8mb4_general_ci` 不区分大小写。

语法

```
SHOW COLLATION;
```

示例

```
nebula> SHOW COLLATION;
+------------+---------+
| Collation  | Charset |
+------------+---------+
| "utf8_bin" | "utf8"  |
+------------+---------+
```

参数	说明
`Collation`	排序规则名称。
`Charset`	与排序规则关联的字符集名称。

最后更新: June 20, 2023

2022 vesoft Inc.

SHOW CREATE SPACE

`SHOW CREATE SPACE` 语句显示指定图空间的创建语句。

图空间的更多详细信息，请参见 CREATE SPACE。

语法

```
SHOW CREATE SPACE <space_name>;
```

示例

```
nebula> SHOW CREATE SPACE basketballplayer;
+--------------------
+-----------------------
```

SHOW CREATE TAG/EDGE

`SHOW CREATE TAG` 语句显示指定 Tag 的基本信息。Tag 的更多详细信息，请参见 CREATE TAG。

`SHOW CREATE EDGE` 语句显示指定 Edge type 的基本信息。Edge type 的更多详细信息，请参见 CREATE EDGE。

语法

```
SHOW CREATE {TAG <tag_name> | EDGE <edge_name>};
```

示例

```
nebula> SHOW CREATE TAG player;
+----------+----------------------------------+
| Tag      | Create Tag                       |
+----------+----------------------------------+
| "player" | "CREATE TAG `player` (           |
|          |  `name` string NULL,             |
|          |  `age` int64 NULL                |
|          | ) ttl_duration = 0, ttl_col = """ |
+----------+----------------------------------+

nebula> SHOW CREATE EDGE follow;
+----------+----------------------------------+
| Edge     | Create Edge                      |
+----------+----------------------------------+
| "follow" | "CREATE EDGE `follow` (          |
|          |  `degree` int64 NULL             |
|          | ) ttl_duration = 0, ttl_col = """ |
+----------+----------------------------------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

SHOW HOSTS

`SHOW HOSTS` 语句可以显示集群信息，包括端口、状态、leader、分片、版本等信息，或者指定显示 Graph、Storage、Meta 服务主机信息。

语法

```
SHOW HOSTS [GRAPH | STORAGE | META];
```

> **Note**
>
> 对于使用源码安装的 Nebula Graph，执行添加了服务名的命令后，输出的信息中不显示版本信息。

示例

```
nebula> SHOW HOSTS;
+-------------+------+-----------+----------+--------------+---------------------------------+------------------------------+---------+
| Host        | Port | HTTP port | Status   | Leader count | Leader distribution             | Partition distribution       | Version |
+-------------+------+-----------+----------+--------------+---------------------------------+------------------------------+---------+
| "storaged0" | 9779 | 19669     | "ONLINE" | 8            | "docs:5, basketballplayer:3"    | "docs:5, basketballplayer:3" | "3.1.0" |
| "storaged1" | 9779 | 19669     | "ONLINE" | 9            | "basketballplayer:4, docs:5"    | "docs:5, basketballplayer:4" | "3.1.0" |
| "storaged2" | 9779 | 19669     | "ONLINE" | 8            | "basketballplayer:3, docs:5"    | "docs:5, basketballplayer:3" | "3.1.0" |
+-------------+------+-----------+----------+--------------+---------------------------------+------------------------------+---------+

nebula> SHOW HOSTS GRAPH;
+-----------+------+----------+---------+--------------+---------+
| Host      | Port | Status   | Role    | Git Info Sha | Version |
+-----------+------+----------+---------+--------------+---------+
| "graphd"  | 9669 | "ONLINE" | "GRAPH" | "3ba41bd"    | "3.1.0" |
| "graphd1" | 9669 | "ONLINE" | "GRAPH" | "3ba41bd"    | "3.1.0" |
| "graphd2" | 9669 | "ONLINE" | "GRAPH" | "3ba41bd"    | "3.1.0" |
+-----------+------+----------+---------+--------------+---------+

nebula> SHOW HOSTS STORAGE;
+-------------+------+----------+-----------+--------------+---------+
| Host        | Port | Status   | Role      | Git Info Sha | Version |
+-------------+------+----------+-----------+--------------+---------+
| "storaged0" | 9779 | "ONLINE" | "STORAGE" | "3ba41bd"    | "3.1.0" |
| "storaged1" | 9779 | "ONLINE" | "STORAGE" | "3ba41bd"    | "3.1.0" |
| "storaged2" | 9779 | "ONLINE" | "STORAGE" | "3ba41bd"    | "3.1.0" |
+-------------+------+----------+-----------+--------------+---------+

nebula> SHOW HOSTS META;
+----------+------+----------+--------+--------------+---------+
| Host     | Port | Status   | Role   | Git Info Sha | Version |
+----------+------+----------+--------+--------------+---------+
| "metad2" | 9559 | "ONLINE" | "META" | "3ba41bd"    | "3.1.0" |
| "metad0" | 9559 | "ONLINE" | "META" | "3ba41bd"    | "3.1.0" |
| "metad1" | 9559 | "ONLINE" | "META" | "3ba41bd"    | "3.1.0" |
+----------+------+----------+--------+--------------+---------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

SHOW INDEX STATUS

`SHOW INDEX STATUS` 语句显示重建原生索引的作业状态，以便确定重建索引是否成功。

语法

```
SHOW {TAG | EDGE} INDEX STATUS;
```

示例

```
nebula> SHOW TAG INDEX STATUS;
+-----------------------------------+-------------+
| Name                              | Index Status |
+-----------------------------------+-------------+
| "date1_index"                     | "FINISHED"   |
| "basketballplayer_all_tag_indexes" | "FINISHED"   |
| "any_shape_geo_index"             | "FINISHED"   |
+-----------------------------------+-------------+

nebula> SHOW EDGE INDEX STATUS;
+---------------+-------------+
| Name          | Index Status |
+---------------+-------------+
| "follow_index" | "FINISHED"   |
+---------------+-------------+
```

相关文档

- 管理作业
- REBUILD NATIVE INDEX

最后更新: June 20, 2023

2022 vesoft Inc.

SHOW INDEXES

`SHOW INDEXES` 语句可以列出当前图空间内的所有 Tag 和 Edge type（包括属性）的索引。

语法

```
SHOW {TAG | EDGE} INDEXES;
```

示例

```
nebula> SHOW TAG INDEXES;
+-----------------+-------------+-----------------+
| Index Name      | By Tag      | Columns         |
+-----------------+-------------+-----------------+
| "fix"           | "fix_string" | ["p1"]          |
| "player_index_0" | "player"    | ["name"]        |
| "player_index_1" | "player"    | ["name", "age"] |
| "var"           | "var_string" | ["p1"]          |
+-----------------+-------------+-----------------+

nebula> SHOW EDGE INDEXES;
+---------------+----------+---------+
| Index Name    | By Edge  | Columns |
+---------------+----------+---------+
| "follow_index" | "follow" | []      |
+---------------+----------+---------+
```

> 历史版本兼容性

Nebula Graph 2.0.1 中， `SHOW TAG/EDGE INDEXES` 语句仅返回 `Names`。

最后更新: June 20, 2023

2022 vesoft Inc.

SHOW PARTS

`SHOW PARTS` 语句显示图空间中指定分片或所有分片的信息。

语法

```
SHOW PARTS [<part_id>];
```

示例

```
nebula> SHOW PARTS;
+--------------+--------------------+--------------------+-------+
| Partition ID | Leader             | Peers              | Losts |
+--------------+--------------------+--------------------+-------+
| 1            | "192.168.2.1:9779" | "192.168.2.1:9779" | ""    |
| 2            | "192.168.2.2:9779" | "192.168.2.2:9779" | ""    |
| 3            | "192.168.2.3:9779" | "192.168.2.3:9779" | ""    |
| 4            | "192.168.2.1:9779" | "192.168.2.1:9779" | ""    |
| 5            | "192.168.2.2:9779" | "192.168.2.2:9779" | ""    |
| 6            | "192.168.2.3:9779" | "192.168.2.3:9779" | ""    |
| 7            | "192.168.2.1:9779" | "192.168.2.1:9779" | ""    |
| 8            | "192.168.2.2:9779" | "192.168.2.2:9779" | ""    |
| 9            | "192.168.2.3:9779" | "192.168.2.3:9779" | ""    |
| 10           | "192.168.2.1:9779" | "192.168.2.1:9779" | ""    |
+--------------+--------------------+--------------------+-------+

nebula> SHOW PARTS 1;
+--------------+--------------------+--------------------+-------+
| Partition ID | Leader             | Peers              | Losts |
+--------------+--------------------+--------------------+-------+
| 1            | "192.168.2.1:9779" | "192.168.2.1:9779" | ""    |
+--------------+--------------------+--------------------+-------+
```

返回结果的说明如下：

参数	说明
`Partition ID`	存储分片的 ID
`Leader`	分片对应的 Raft leader 副本的信息，包括 IP 地址与服务端口
`Peers`	分片对应的所有副本（leader 与 follower）的信息，包括 IP 地址与服务端口
`Losts`	分片对应的处于离线状态的副本信息，包括 IP 地址和服务端口

最后更新: June 20, 2023

2022 vesoft Inc.

SHOW ROLES

`SHOW ROLES` 语句显示分配给用户的角色信息。

根据登录的用户角色，返回的结果也有所不同：

- 如果登录的用户角色是 `GOD`，或者有权访问该图空间的 `ADMIN`，则返回该图空间内除 `GOD` 之外的所有用户角色信息。
- 如果登录的用户角色是有权访问该图空间 `DBA` 、 `USER` 或 `GUEST`，则返回自身的角色信息。
- 如果登录的用户角色没有权限访问该图空间，则返回权限错误。

关于角色的详情请参见内置角色权限。

语法

```
SHOW ROLES IN <space_name>;
```

示例

```
nebula> SHOW ROLES in basketballplayer;
+---------+-----------+
| Account | Role Type |
+---------+-----------+
| "user1" | "ADMIN"   |
+---------+-----------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

SHOW SNAPSHOTS

`SHOW SNAPSHOTS` 语句显示所有快照信息。

快照的使用方式请参见管理快照。

角色要求

只有 `GOD` 角色的用户（即 `root`）才能执行 `SHOW SNAPSHOTS` 语句。

语法

```
SHOW SNAPSHOTS;
```

示例

```
nebula> SHOW SNAPSHOTS;
+---------------------------------+---------+-----------------------------------------------------+
| Name                            | Status  | Hosts                                               |
+---------------------------------+---------+-----------------------------------------------------+
| "SNAPSHOT_2020_12_16_11_13_55" | "VALID" | "storaged0:9779, storaged1:9779, storaged2:9779"    |
| "SNAPSHOT_2020_12_16_11_14_10" | "VALID" | "storaged0:9779, storaged1:9779, storaged2:9779"    |
+---------------------------------+---------+-----------------------------------------------------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

SHOW SPACES

`SHOW SPACES` 语句显示现存的图空间。

如何创建图空间，请参见 CREATE SPACE。

语法

```
SHOW SPACES;
```

示例

```
nebula> SHOW SPACES;
+--------------------+
| Name               |
+--------------------+
| "docs"             |
| "basketballplayer" |
+--------------------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

SHOW STATS

`SHOW STATS` 语句显示最近 `STATS` 作业收集的图空间统计信息。

图空间统计信息包含：

- 点的总数
- 边的总数
- 每个 Tag 关联的点的总数
- 每个 Edge type 关联的边的总数

前提条件

在需要查看统计信息的图空间中执行 `SUBMIT JOB STATS` 。详情请参见 SUBMIT JOB STATS。

Caution

`SHOW STATS` 的结果取决于最后一次执行的 `SUBMIT JOB STATS` 。如果发生过新的写入或者更改，必须再次执行 `SUBMIT JOB STATS` ，否则统计数据有错误。

语法

```
SHOW STATS;
```

示例

```
# 选择图空间。
nebula> USE basketballplayer;

# 执行 SUBMIT JOB STATS。
nebula> SUBMIT JOB STATS;
+------------+
| New Job Id |
+------------+
| 98         |
+------------+

# 确认作业执行成功。
nebula> SHOW JOB 98;
+---------------+---------------+------------+----------------------------+----------------------------+-------------+
| Job Id(TaskId) | Command(Dest) | Status     | Start Time                 | Stop Time                  | Error Code  |
+---------------+---------------+------------+----------------------------+----------------------------+-------------+
| 98            | "STATS"       | "FINISHED" | 2021-11-01T09:33:21.000000 | 2021-11-01T09:33:21.000000 | "SUCCEEDED" |
| 0             | "storaged2"   | "FINISHED" | 2021-11-01T09:33:21.000000 | 2021-11-01T09:33:21.000000 | "SUCCEEDED" |
| 1             | "storaged0"   | "FINISHED" | 2021-11-01T09:33:21.000000 | 2021-11-01T09:33:21.000000 | "SUCCEEDED" |
| 2             | "storaged1"   | "FINISHED" | 2021-11-01T09:33:21.000000 | 2021-11-01T09:33:21.000000 | "SUCCEEDED" |
+---------------+---------------+------------+----------------------------+----------------------------+-------------+

# 显示图空间统计信息。
nebula> SHOW STATS;
+---------+-----------+-------+
| Type    | Name      | Count |
+---------+-----------+-------+
| "Tag"   | "player"  | 51    |
| "Tag"   | "team"    | 30    |
| "Edge"  | "follow"  | 81    |
| "Edge"  | "serve"   | 152   |
| "Space" | "vertices" | 81    |
| "Space" | "edges"   | 233   |
+---------+-----------+-------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

SHOW TAGS/EDGES

`SHOW TAGS` 语句显示当前图空间内的所有 Tag。

`SHOW EDGES` 语句显示当前图空间内的所有 Edge type。

语法

```
SHOW {TAGS | EDGES};
```

示例

```
nebula> SHOW TAGS;
+----------+
| Name     |
+----------+
| "player" |
| "star"   |
| "team"   |
+----------+

nebula> SHOW EDGES;
+----------+
| Name     |
+----------+
| "follow" |
| "serve"  |
+----------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

SHOW USERS

`SHOW USERS` 语句显示用户信息。

角色要求

只有 `GOD` 角色的用户（即 `root`）才能执行 `SHOW USERS` 语句。

语法

```
SHOW USERS;
```

示例

```
nebula> SHOW USERS;
+---------+-----------------+
| Account | IP Whitelist    |
+---------+-----------------+
| "root"  | ""              |
| "user1" | ""              |
| "user2" | "192.168.10.10" |
+---------+-----------------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

SHOW SESSIONS

登录 Nebula Graph 数据库时，会创建对应会话，用户可以查询会话信息。

注意事项

- 执行 `exit` 退出登录时，客户端会调用 API `release`，释放会话并清除会话信息。如果没有正常退出，且没有在配置文件 nebula-graphd.conf 设置空闲会话超时时间（`session_idle_timeout_secs`），会话不会自动释放。对于未自动释放的会话，需要手动删除指定会话 (TODO: coding)。
- `SHOW SESSIONS` 查询所有 Graph 服务上的会话信息。
- `SHOW LOCAL SESSIONS` 从当前连接的 Graph 服务获取会话信息，不会查询其他 Graph 服务上的会话信息。
- `SHOW SESSION <Session_Id>` 查询指定 Session ID 的会话信息。

语法

```
SHOW [LOCAL] SESSIONS;
SHOW SESSION <Session_Id>;
```

示例

```
nebula> SHOW SESSIONS;
+------------------+----------+--------------------+----------------------------+----------------------------+-----------------+----------+--------------------+
| SessionId        | UserName | SpaceName          | CreateTime                 | UpdateTime                 | GraphAddr       | Timezone | ClientIp           |
+------------------+----------+--------------------+----------------------------+----------------------------+-----------------+----------+--------------------+
| 1651220858102296 | "root"   | "basketballplayer" | 2022-04-29T08:27:38.102296 | 2022-04-29T08:50:46.282921 | "127.0.0.1:9669" | 0        | "::ffff:127.0.0.1" |
| 1651199330300991 | "root"   | "basketballplayer" | 2022-04-29T02:28:50.300991 | 2022-04-29T08:16:28.339038 | "127.0.0.1:9669" | 0        | "::ffff:127.0.0.1" |
| 1651112899847744 | "root"   | "basketballplayer" | 2022-04-28T02:28:19.847744 | 2022-04-28T08:17:44.470210 | "127.0.0.1:9669" | 0        | "::ffff:127.0.0.1" |
| 1651041092662100 | "root"   | "basketballplayer" | 2022-04-27T06:31:32.662100 | 2022-04-27T07:01:25.200978 | "127.0.0.1:9669" | 0        | "::ffff:127.0.0.1" |
| 1650959429593975 | "root"   | "basketballplayer" | 2022-04-26T07:50:29.593975 | 2022-04-26T07:51:47.184810 | "127.0.0.1:9669" | 0        | "::ffff:127.0.0.1" |
| 1650958897679595 | "root"   | ""                 | 2022-04-26T07:41:37.679595 | 2022-04-26T07:41:37.683802 | "127.0.0.1:9669" | 0        | "::ffff:127.0.0.1" |
+------------------+----------+--------------------+----------------------------+----------------------------+-----------------+----------+--------------------+

nebula> SHOW SESSION 1635254859271703;
+------------------+----------+--------------------+----------------------------+----------------------------+-----------------+----------+--------------------+
| SessionId        | UserName | SpaceName          | CreateTime                 | UpdateTime                 | GraphAddr       | Timezone | ClientIp           |
+------------------+----------+--------------------+----------------------------+----------------------------+-----------------+----------+--------------------+
| 1651220858102296 | "root"   | "basketballplayer" | 2022-04-29T08:27:38.102296 | 2022-04-29T08:50:54.254384 | "127.0.0.1:9669" | 0        | "::ffff:127.0.0.1" |
+------------------+----------+--------------------+----------------------------+----------------------------+-----------------+----------+--------------------+
```

参数	说明
`SessionId`	会话 ID，唯一标识一个会话。
`UserName`	会话的登录用户名称。
`SpaceName`	用户当前所使用的图空间。刚登录时为空（`""`）。
`CreateTime`	会话的创建时间，即用户认证登录的时间。时区为配置文件中 `timezone_name` 指定的时区。
`UpdateTime`	用户有执行操作时，会更新此时间。时区为配置文件中 `timezone_name` 指定的时区。
`GraphAddr`	会话的 Graph 服务地址和端口。
`Timezone`	保留参数，暂无意义。
`ClientIp`	会话的客户端 IP 地址。

最后更新: June 20, 2023

2022 vesoft Inc.

SHOW QUERIES

`SHOW QUERIES` 语句可以查看当前 Session 中正在执行的查询请求信息。

> **Note**
>
> 如果需要终止查询，请参见终止查询。

注意事项

- `SHOW LOCAL QUERIES` 从本地缓存获取当前 Session 中查询的状态，几乎没有延迟。
- `SHOW QUERIES` 从 Meta 服务获取所有 Session 中的查询信息。这些信息会根据参数 `session_reclaim_interval_secs` 定义的周期同步到 Meta 服务，因此在客户端获取到的信息可能属于上个同步周期。

语法

```
SHOW [LOCAL] QUERIES;
```

示例

```
nebula> SHOW LOCAL QUERIES;
+-----------------+-----------------+-------+---------------------+----------------------------+-----------------+----------+----------------------+
| SessionID       | ExecutionPlanID | User  | Host                | StartTime                  | DurationInUSec | Status   | Query                |
+-----------------+-----------------+-------+---------------------+----------------------------+-----------------+----------+----------------------+
| 1625463842921750 | 46             | "root" | ""192.168.x.x":9669" | 2021-07-05T05:44:19.502903 | 0              | "RUNNING" | "SHOW LOCAL QUERIES;" |
+-----------------+-----------------+-------+---------------------+----------------------------+-----------------+----------+----------------------+

nebula> SHOW QUERIES;
+-----------------+-----------------+-------+---------------------+----------------------------+-----------------+----------+-----------------------------------------------------------+
| SessionID       | ExecutionPlanID | User  | Host                | StartTime                  | DurationInUSec | Status   | Query                                                     |
+-----------------+-----------------+-------+---------------------+----------------------------+-----------------+----------+-----------------------------------------------------------+
| 1625456037718757 | 54             | "user1" | ""192.168.x.x":9669" | 2021-07-05T05:51:08.691318 | 1504502        | "RUNNING" | "MATCH p=(v:player)-[*1..4]-(v2) RETURN v2 AS Friends;" |
+-----------------+-----------------+-------+---------------------+----------------------------+-----------------+----------+-----------------------------------------------------------+

# 返回耗时 TOP 10 的查询。
nebula> SHOW QUERIES | ORDER BY $-.DurationInUSec DESC | LIMIT 10;
+-----------------+-----------------+-------+---------------------+----------------------------+-----------------+----------+-----------------------------------------------------------+
| SessionID       | ExecutionPlanID | User  | Host                | StartTime                  | DurationInUSec | Status   | Query                                                     |
+-----------------+-----------------+-------+---------------------+----------------------------+-----------------+----------+-----------------------------------------------------------+
| 1625471375320831 | 98             | "user2" | ""192.168.x.x":9669" | 2021-07-05T07:50:24.461779 | 2608176        | "RUNNING" | "MATCH (v:player)-[*1..4]-(v2) RETURN v2 AS Friends;" |
| 1625456037718757 | 99             | "user1" | ""192.168.x.x":9669" | 2021-07-05T07:50:24.910616 | 2159333        | "RUNNING" | "MATCH (v:player)-[*1..4]-(v2) RETURN v2 AS Friends;" |
+-----------------+-----------------+-------+---------------------+----------------------------+-----------------+----------+-----------------------------------------------------------+
```

参数说明如下。

参数	说明
`SessionID`	会话 ID。
`ExecutionPlanID`	执行计划 ID。
`User`	执行查询的用户名。
`Host`	用户连接的服务器地址和端口。
`StartTime`	执行查询的开始时间。
`DurationInUSec`	执行查询的持续时长。单位：微秒。
`Status`	查询的当前状态。
`Query`	查询语句。

最后更新: June 20, 2023

2022 vesoft Inc.

SHOW META LEADER

`SHOW META LEADER` 语句显示当前 Meta 集群的 leader 信息。

关于 Meta 服务的详细说明请参见 Meta 服务。

语法

```
SHOW META LEADER;
```

示例

```
nebula> SHOW META LEADER;
+-----------------+---------------------------+
| Meta Leader     | secs from last heart beat |
+-----------------+---------------------------+
| "127.0.0.1:9559" | 3                         |
+-----------------+---------------------------+
```

参数	说明
`Meta Leader`	Meta 集群的 leader 信息，包括 leader 所在服务器的 IP 地址和端口。
`secs from last heart beat`	距离上次心跳的时间间隔。单位：秒。

最后更新: June 20, 2023

2022 vesoft Inc.

4.7 子句和选项

4.7.1 GROUP BY

`GROUP BY` 子句可以用于聚合数据。

openCypher 兼容性

本文操作仅适用于原生 nGQL。

用户也可以使用 openCypher 方式的 `count()` 函数聚合数据。

```
nebula>  MATCH (v:player)<-[:follow]-(:player) RETURN v.player.name AS Name, count(*) as cnt ORDER BY cnt DESC;
+----------------------+-----+
| Name                 | cnt |
+----------------------+-----+
| "Tim Duncan"         | 10  |
| "LeBron James"       | 6   |
| "Tony Parker"        | 5   |
| "Chris Paul"         | 4   |
| "Manu Ginobili"      | 4   |
+----------------------+-----+
...
```

语法

`GROUP BY` 子句可以聚合相同值的行，然后进行计数、排序和计算等操作。

`GROUP BY` 子句可以在管道符（|）之后和 `YIELD` 子句之前使用。

```
| GROUP BY <var> YIELD <var>, <aggregation_function(var)>
```

`aggregation_function()` 函数支持 `avg()` 、`sum()` 、`max()` 、`min()` 、`count()` 、`collect()` 、`std()` 。

示例

```
# 查找所有连接到 player100 的点，并根据他们的姓名进行分组，返回姓名的出现次数。
nebula> GO FROM "player100" OVER follow BIDIRECT \
        YIELD properties($$).name as Name \
        | GROUP BY $-.Name \
        YIELD $-.Name as Player, count(*) AS Name_Count;
+--------------------+------------+
| Player             | Name_Count |
+--------------------+------------+
| "Shaquille O'Neal" | 1          |
| "Tiago Splitter"   | 1          |
| "Manu Ginobili"    | 2          |
| "Boris Diaw"       | 1          |
| "LaMarcus Aldridge" | 1          |
| "Tony Parker"      | 2          |
| "Marco Belinelli"  | 1          |
| "Dejounte Murray"  | 1          |
| "Danny Green"      | 1          |
| "Aron Baynes"      | 1          |
+--------------------+------------+
```

用函数进行分组和计算

```
# 查找所有连接到 player100 的点，并根据起始点进行分组，返回 degree 的总和。
nebula> GO FROM "player100" OVER follow \
        YIELD src(edge) AS player, properties(edge).degree AS degree \
        | GROUP BY $-.player \
        YIELD sum($-.degree);
+---------------+
| sum($-.degree) |
+---------------+
| 190           |
+---------------+
```

`sum()` 函数详情请参见内置数学函数。

2022 vesoft Inc.

最后更新: June 20, 2023

2022 vesoft Inc.

4.7.2 LIMIT

`LIMIT` 子句限制输出结果的行数。`LIMIT` 在原生 nGQL 语句和 openCypher 兼容语句中的用法有所不同。

- 在原生 nGQL 语句中，一般需要在 `LIMIT` 子句前使用管道符，可以直接在 LIMIT 语句后设置或者省略偏移量参数。
- 在 openCypher 兼容语句中，不允许在 `LIMIT` 子句前使用管道符，可以使用 `SKIP` 指明偏移量。

> **Note**
>
> 在原生 nGQL 或 openCypher 方式中使用 `LIMIT` 时，使用 `ORDER BY` 子句限制输出顺序非常重要，否则会输出一个不可预知的子集。

原生 nGQL 语句中的 LIMIT

在原生 nGQL 中，`LIMIT` 有通用语法和 `GO` 语句中的专属语法。

原生 NGQL 中的通用 LIMIT 语法

原生 nGQL 中的通用 `LIMIT` 语法与 `SQL` 中的 `LIMIT` 原理相同。`LIMIT` 子句接收一个或两个参数，参数的值必须是非负整数，且必须用在管道符之后。语法和说明如下：

```
... | LIMIT [<offset>,] <number_rows>;
```

参数	说明
`offset`	偏移量，即定义从哪一行开始返回。索引从 `0` 开始。默认值为 `0`，表示从第一行开始返回。
`number_rows`	返回的总行数。

示例：

```
# 从结果中返回最前面的 3 行数据。
nebula> LOOKUP ON player YIELD id(vertex)|\
        LIMIT 3;
+-------------+
| id(VERTEX)  |
+-------------+
| "player100" |
| "player101" |
| "player102" |
+-------------+

# 从排序后结果中返回第 2 行开始的 3 行数据。
nebula> GO FROM "player100" OVER follow REVERSELY \
        YIELD properties($$).name AS Friend, properties($$).age AS Age \
        | ORDER BY $-.Age, $-.Friend \
        | LIMIT 1, 3;
+-------------------+-----+
| Friend            | Age |
+-------------------+-----+
| "Danny Green"     | 31  |
| "Aron Baynes"     | 32  |
| "Marco Belinelli" | 32  |
+-------------------+-----+
```

GO 语句中的 LIMIT

`GO` 语句中的 `LIMIT` 除了支持原生 nGQL 中的通用语法外，还支持根据边限制输出结果数量。

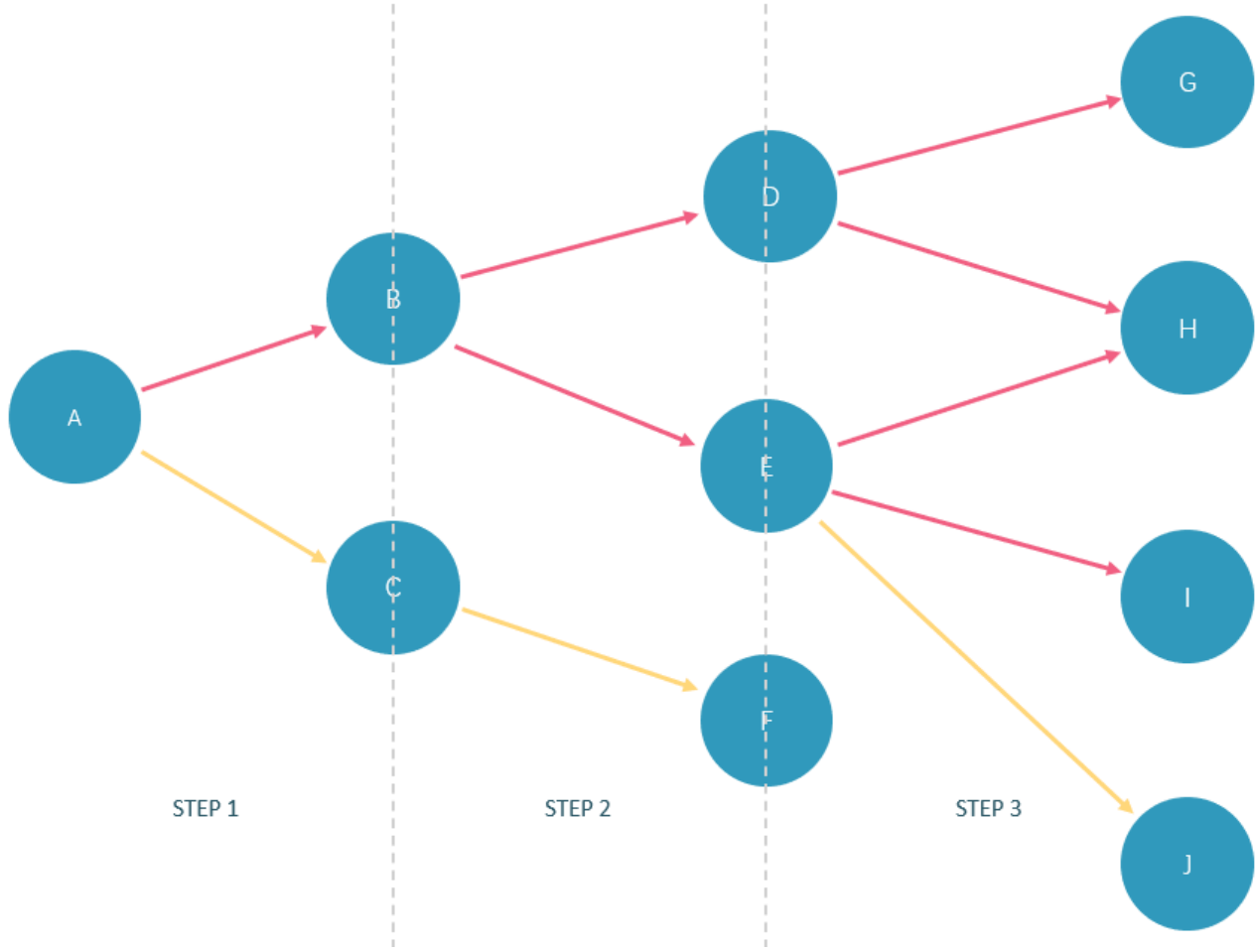
语法：

```
<go_statement> LIMIT <limit_list>;
```


2022 vesoft Inc.

`limit_list` 是一个列表，列表中的元素必须为自然数，且元素数量必须与 `GO` 语句中的 `STEPS` 的最大数相同。下文以 `GO 1 TO 3 STEPS FROM "A" OVER * LIMIT <limit_list>` 为例详细介绍 `LIMIT` 的这种用法。

- 列表 `limit_list` 必须包含 3 个自然数元素，例如 `GO 1 TO 3 STEPS FROM "A" OVER * LIMIT [1,2,4]` 。
- `LIMIT [1,2,4]` 中的 `1` 表示系统在第一步时自动选择 1 条边继续遍历， `2` 表示在第二步时选择 2 条边继续遍历， `4` 表示在第三步时选择 4 条边继续遍历。
- 因为 `GO 1 TO 3 STEPS` 表示返回第一到第三步的所有遍历结果，因此下图中所有红色边和它们的原点与目的点都会被这条 `GO` 语句匹配上，而黄色边表示 `GO` 语句遍历时没有选择的路径。如果不是 `GO 1 TO 3 STEPS` 而是 `GO 3 STEPS` ，则只会匹配上第三步的红色边和它们两端的点。



在 basketballplayer 数据集中的执行示例如下：

```
nebula> GO 3 STEPS FROM "player100" \
        OVER * \
        YIELD properties($$).name AS NAME, properties($$).age AS Age \
        LIMIT [3,3,3];
+-----------------+-------------+
| NAME            | Age         |
+-----------------+-------------+
| "Spurs"         | UNKNOWN_PROP |
| "Tony Parker"   | 36          |
| "Manu Ginobili" | 41          |
+-----------------+-------------+

nebula> GO 3 STEPS FROM "player102" OVER * BIDIRECT\
        YIELD dst(edge) \
        LIMIT [rand32(5),rand32(5),rand32(5)];
+-------------+
| dst(EDGE)   |
+-------------+
| "player100" |
| "player100" |
+-------------+
```


2022 vesoft Inc.

openCypher 兼容语句中的 LIMIT

在 MATCH 等 openCypher 兼容语句中使用 LIMIT 不需要加管道符。语法和说明如下：

```
... [SKIP <offset>] [LIMIT <number_rows>];
```

参数	说明
offset	偏移量，即定义从哪一行开始返回。索引从 0 开始。默认值为 0，表示从第一行开始返回。
number_rows	返回的总行数量。

offset 和 number_rows 可以使用表达式，但是表达式的结果必须是非负整数。

> **Note**
>
> 两个整数组成的分数表达式会自动向下取整。例如 8/6 向下取整为 1。

单独使用 LIMIT

LIMIT 可以单独使用，返回指定数量的结果。

```
nebula> MATCH (v:player) RETURN v.player.name AS Name, v.player.age AS Age \
        ORDER BY Age LIMIT 5;
+-------------------------+-----+
| Name                    | Age |
+-------------------------+-----+
| "Luka Doncic"           | 20  |
| "Ben Simmons"           | 22  |
| "Kristaps Porzingis"    | 23  |
| "Giannis Antetokounmpo" | 24  |
| "Kyle Anderson"         | 25  |
+-------------------------+-----+

nebula> MATCH (v:player) RETURN v.player.name AS Name, v.player.age AS Age \
        ORDER BY Age LIMIT rand32(5);
+-------------------------+-----+
| Name                    | Age |
+-------------------------+-----+
| "Luka Doncic"           | 20  |
| "Ben Simmons"           | 22  |
| "Kristaps Porzingis"    | 23  |
| "Giannis Antetokounmpo" | 24  |
+-------------------------+-----+
```

单独使用 SKIP

SKIP 可以单独使用，用于设置偏移量，返回指定位置之后的数据。

```
nebula> MATCH (v:player{name:"Tim Duncan"}) --> (v2) \
        RETURN v2.player.name AS Name, v2.player.age AS Age \
        ORDER BY Age DESC SKIP 1;
+-----------------+-----+
| Name            | Age |
+-----------------+-----+
| "Manu Ginobili" | 41  |
| "Tony Parker"   | 36  |
+-----------------+-----+

nebula> MATCH (v:player{name:"Tim Duncan"}) --> (v2) \
        RETURN v2.player.name AS Name, v2.player.age AS Age \
        ORDER BY Age DESC SKIP 1+1;
+---------------+-----+
| Name          | Age |
+---------------+-----+
| "Tony Parker" | 36  |
+---------------+-----+
```

同时使用 SKIP 与 LIMIT

同时使用 SKIP 与 LIMIT 可以返回从指定位置开始的指定数量的数据。

```
nebula> MATCH (v:player{name:"Tim Duncan"}) --> (v2) \
        RETURN v2.player.name AS Name, v2.player.age AS Age \
        ORDER BY Age DESC SKIP 1 LIMIT 1;
+-----------------+-----+
| Name            | Age |
```


2022 vesoft Inc.

```
+-----------------+-----+
| "Manu Ginobili" | 41  |
+-----------------+-----+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.7.3 SAMPLE

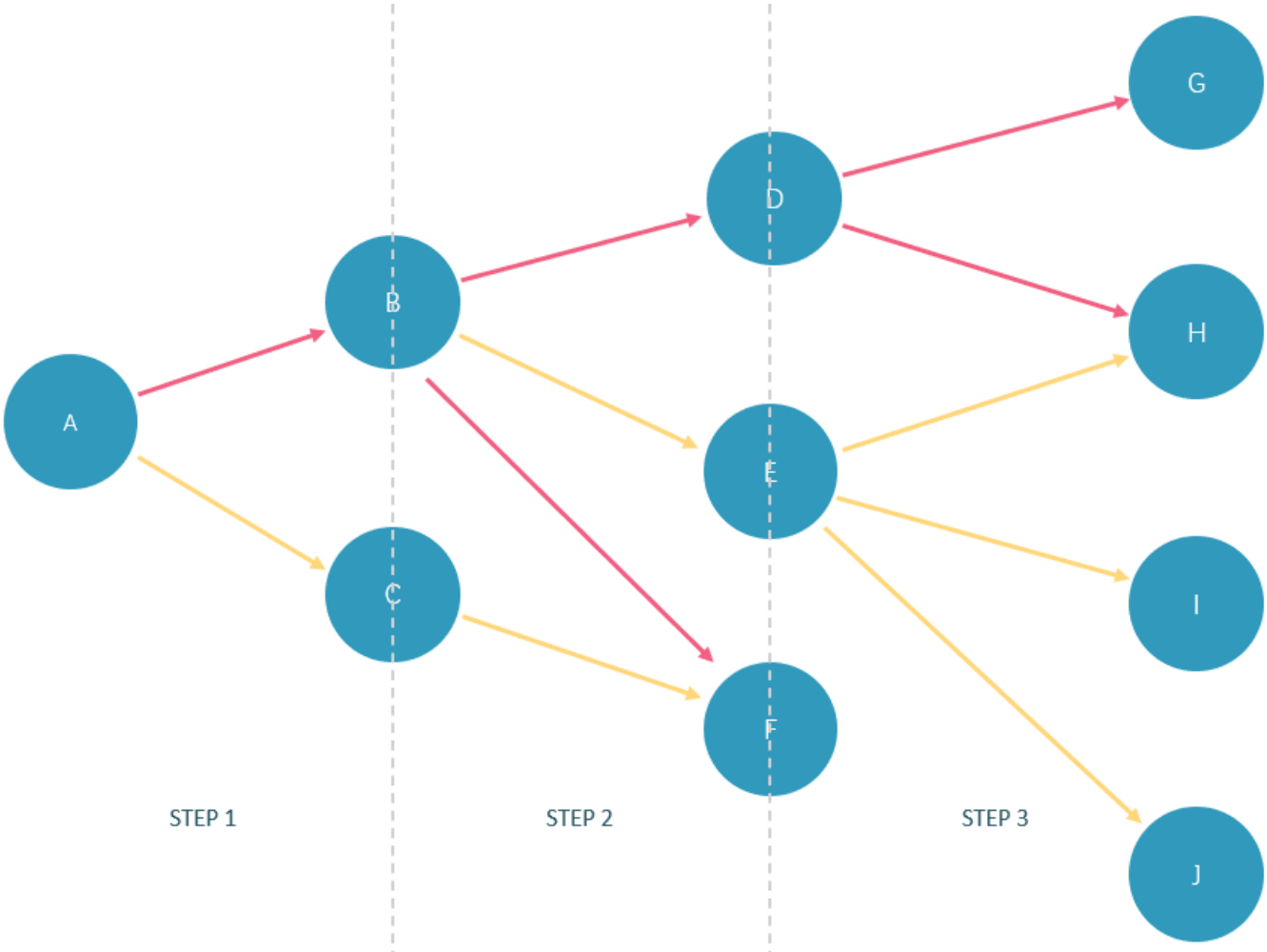
`SAMPLE` 子句用于在结果集中均匀取样并返回指定数量的数据。

`SAMPLE` 仅能在 `GO` 语句中使用，语法如下：

```
<go_statement> SAMPLE <sample_list>;
```

`sample_list` 是一个列表，列表中的元素必须为自然数，且元素数量必须与 `GO` 语句中的 `STEPS` 的最大数相同。下文以 `GO 1 TO 3 STEPS FROM "A" OVER * SAMPLE <sample_list>` 为例详细介绍 `SAMPLE` 的用法。

- 列表 `sample_list` 必须包含 3 个自然数元素，例如 `GO 1 TO 3 STEPS FROM "A" OVER * SAMPLE [1,2,4]`。
- `SAMPLE [1,2,4]` 中的 `1` 表示系统在第一步时自动选择 1 条边继续遍历，`2` 表示在第二步时选择 2 条边继续遍历，`4` 表示在第三步时选择 4 条边继续遍历。如果某一步没有匹配的边或者匹配到的边数量小于指定数量，则按实际数量返回。
- 因为 `GO 1 TO 3 STEPS` 表示返回第一到第三步的所有遍历结果，因此下图中所有红色边和它们的原点与目的点都会被这条 `GO` 语句匹配上，而黄色边表示 `GO` 语句遍历时没有选择的路径。如果不是 `GO 1 TO 3 STEPS` 而是 `GO 3 STEPS`，则只会匹配上第三步的红色边和它们两端的点。



在 basketballplayer 数据集中的执行示例如下：

```
nebula> GO 3 STEPS FROM "player100" \
        OVER * \
        YIELD properties($$).name AS NAME, properties($$).age AS Age \
        SAMPLE [1,2,3];
+-----------------+-------------+
| NAME            | Age         |
+-----------------+-------------+
| "Spurs"         | UNKNOWN_PROP |
| "Tony Parker"   | 36          |
| "Manu Ginobili" | 41          |
```


2022 vesoft Inc.

```
+----------------+-------------+

nebula> GO 1 TO 3 STEPS FROM "player100" \
        OVER * \
        YIELD properties($$).name AS NAME, properties($$).age AS Age \
        SAMPLE [2,2,2];
+--------------------+-----+
| NAME               | Age |
+--------------------+-----+
| "Manu Ginobili"    | 41  |
| "Tony Parker"      | 36  |
| "Tim Duncan"       | 42  |
| "LaMarcus Aldridge" | 33  |
| "Tony Parker"      | 36  |
| "Tim Duncan"       | 42  |
+--------------------+-----+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.7.4 ORDER BY

`ORDER BY` 子句指定输出结果的排序规则。

- 在原生 nGQL 中，必须在 `YIELD` 子句之后使用管道符（|）和 `ORDER BY` 子句。
- 在 openCypher 方式中，不允许使用管道符。在 `RETURN` 子句之后使用 `ORDER BY` 子句。

排序规则分为如下两种:

- `ASC`（默认）：升序。
- `DESC`：降序。

原生 **nGQL** 语法

```
<YIELD clause>
ORDER BY <expression> [ASC | DESC] [, <expression> [ASC | DESC] ...];
```

> **mpatibility**

原生 nGQL 语法中，`ORDER BY` 命令后必须使用引用符 `$-.`。但在 2.5.0 之前的版本中不需要。

示例

```
nebula> FETCH PROP ON player "player100", "player101", "player102", "player103" \
        YIELD properties(vertex).age AS age, properties(vertex).name AS name \
        | ORDER BY $-.age ASC, $-.name DESC;
+-----+---------------------+
| age | name                |
+-----+---------------------+
| 32  | "Rudy Gay"          |
| 33  | "LaMarcus Aldridge" |
| 36  | "Tony Parker"       |
| 42  | "Tim Duncan"        |
+-----+---------------------+

nebula> $var = GO FROM "player100" OVER follow \
        YIELD dst(edge) AS dst; \
        ORDER BY $var.dst DESC;
+-------------+
| dst         |
+-------------+
| "player125" |
| "player101" |
+-------------+
```

OpenCypher 方式语法

```
<RETURN clause>
ORDER BY <expression> [ASC | DESC] [, <expression> [ASC | DESC] ...];
```

示例

```
nebula> MATCH (v:player) RETURN v.player.name AS Name, v.player.age AS Age  \
        ORDER BY Name DESC;
+-----------------+-----+
| Name            | Age |
+-----------------+-----+
| "Yao Ming"      | 38  |
| "Vince Carter"  | 42  |
| "Tracy McGrady" | 39  |
| "Tony Parker"   | 36  |
| "Tim Duncan"    | 42  |
+-----------------+-----+
...

# 首先以年龄排序，如果年龄相同，再以姓名排序。
nebula> MATCH (v:player) RETURN v.player.age AS Age, v.player.name AS Name  \
        ORDER BY Age DESC, Name ASC;
+-----+------------------+
| Age | Name             |
+-----+------------------+
| 47  | "Shaquille O'Neal" |
```


2022 vesoft Inc.

```
| 46  | "Grant Hill"      |
| 45  | "Jason Kidd"      |
| 45  | "Steve Nash"      |
+-----+-------------------+
...
```

NULL 值的排序

升序排列时，会在输出的最后列出 NULL 值，降序排列时，会在输出的开头列出 NULL 值。

```
nebula> MATCH (v:player{name:"Tim Duncan"}) --> (v2) \
        RETURN v2.player.name AS Name, v2.player.age AS Age  \
        ORDER BY Age;
+-----------------+----------+
| Name            | Age      |
+-----------------+----------+
| "Tony Parker"   | 36       |
| "Manu Ginobili" | 41       |
| __NULL__        | __NULL__ |
+-----------------+----------+

nebula> MATCH (v:player{name:"Tim Duncan"}) --> (v2) \
        RETURN v2.player.name AS Name, v2.player.age AS Age  \
        ORDER BY Age DESC;
+-----------------+----------+
| Name            | Age      |
+-----------------+----------+
| __NULL__        | __NULL__ |
| "Manu Ginobili" | 41       |
| "Tony Parker"   | 36       |
+-----------------+----------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.7.5 RETURN

`RETURN` 子句定义了 nGQL 查询的输出结果。如果需要返回多个字段，用英文逗号（,）分隔。

`RETURN` 可以引导子句或语句：

- `RETURN` 子句可以用于 nGQL 中的 openCypher 方式语句中，例如 `MATCH` 或 `UNWIND`。
- `RETURN` 可以单独使用，输出表达式的结果。

openCypher 兼容性

本文操作仅适用于 nGQL 中的 openCypher 方式。关于原生 nGQL 如何定义输出结果，请参见 `YIELD`。

`RETURN` 不支持如下 openCypher 功能：

- 使用不在英文字母表中的字符作为变量名。例如：

```
MATCH (`点 1`:player) \
RETURN `点 1`;
```

- 设置一个模式，并返回该模式匹配的所有元素。例如：

```
MATCH (v:player) \
RETURN (v)-[e]->(v2);
```

历史版本兼容性

- 在 nGQL 1.x 中，`RETURN` 适用于原生 nGQL，语法为 `RETURN <var_ref> IF <var_ref> IS NOT NULL`。
- 从 nGQL 2.0 开始，`RETURN` 不适用于原生 nGQL。

Map 顺序说明

`RETURN` 返回 Map 时，Key 的顺序是未定义的。

```
nebula> RETURN {age: 32, name: "Marco Belinelli"};
+-----------------------------------+
| {age:32,name:"Marco Belinelli"}   |
+-----------------------------------+
| {age: 32, name: "Marco Belinelli"} |
+-----------------------------------+

nebula> RETURN {zage: 32, name: "Marco Belinelli"};
+-----------------------------------+
| {zage:32,name:"Marco Belinelli"}   |
+-----------------------------------+
| {name: "Marco Belinelli", zage: 32} |
+-----------------------------------+
```

返回点或边

使用 `RETURN {<vertex_name> | <edge_name>}` 返回点或边的所有信息。

```
// 返回点
nebula> MATCH (v:player) \
        RETURN v;
+------------------------------------------------------------+
| v                                                          |
+------------------------------------------------------------+
| ("player104" :player{age: 32, name: "Marco Belinelli"})    |
| ("player107" :player{age: 32, name: "Aron Baynes"})        |
| ("player116" :player{age: 34, name: "LeBron James"})       |
| ("player120" :player{age: 29, name: "James Harden"})       |
| ("player125" :player{age: 41, name: "Manu Ginobili"})      |
+------------------------------------------------------------+
...
```


2022 vesoft Inc.

```
// 返回边
nebula> MATCH (v:player)-[e]->() \
        RETURN e;
+------------------------------------------------------------------------+
| e                                                                      |
+------------------------------------------------------------------------+
| [:follow "player104"->"player100" @0 {degree: 55}]                     |
| [:follow "player104"->"player101" @0 {degree: 50}]                     |
| [:follow "player104"->"player105" @0 {degree: 60}]                     |
| [:serve "player104"->"team200" @0 {end_year: 2009, start_year: 2007}]  |
| [:serve "player104"->"team208" @0 {end_year: 2016, start_year: 2015}]  |
+------------------------------------------------------------------------+
...
```

返回点 **ID**

使用 `id()` 函数返回点 ID。

```
nebula> MATCH (v:player{name:"Tim Duncan"}) \
        RETURN id(v);
+------------+
| id(v)      |
+------------+
| "player100" |
+------------+
```

返回 **Tag**

使用 `labels()` 函数返回点上的 Tag 列表。

```
nebula> MATCH (v:player{name:"Tim Duncan"}) \
        RETURN labels(v);
+------------+
| labels(v)  |
+------------+
| ["player"] |
+------------+
```

返回列表 `labels(v)` 中的第 N 个元素，可以使用 `labels(v)[n-1]` 。例如下面示例使用 `labels(v)[0]` 检索第一个元素。

```
nebula> MATCH (v:player{name:"Tim Duncan"}) \
        RETURN labels(v)[0];
+-------------+
| labels(v)[0] |
+-------------+
| "player"    |
+-------------+
```

返回属性

使用语法 `{<vertex_name>|<edge_name>}.<property>` 返回点或边的属性。

```
nebula> MATCH (v:player) \
        RETURN v.player.name, v.player.age \
        LIMIT 3;
+-----------------+-------------+
| v.player.name   | v.player.age |
+-----------------+-------------+
| "Danny Green"   | 31          |
| "Tiago Splitter" | 34          |
| "David West"    | 38          |
+-----------------+-------------+
```

使用 `properties()` 函数返回点或边的所有属性。

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[]->(v2) \
        RETURN properties(v2);
+----------------------------------+
| properties(v2)                   |
+----------------------------------+
| {name: "Spurs"}                  |
| {age: 36, name: "Tony Parker"}   |
| {age: 41, name: "Manu Ginobili"} |
+----------------------------------+
```


2022 vesoft Inc.

返回 **Edge type**

使用 `type()` 函数返回匹配的 Edge type。

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[e]->() \
        RETURN DISTINCT type(e);
+----------+
| type(e)  |
+----------+
| "serve"  |
| "follow" |
+----------+
```

返回路径

使用 `RETURN <path_name>` 返回匹配路径的所有信息。

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[*3]->() \
        RETURN p;
+-----------------------
```

```
| <("player100" :player{age: 42, name: "Tim Duncan"})-[:follow@0 {degree: 95}]->("player125" :player{age: 41, name: "Manu
Ginobili"})>                                                                                                  | 1      |
| <("player100" :player{age: 42, name: "Tim Duncan"})-[:follow@0 {degree: 95}]->("player101" :player{age: 36, name: "Tony Parker"})-[:serve@0 {end_year: 2018, start_year: 1999}]-
>("team204" :team{name: "Spurs"})>     | 2      |
| <("player100" :player{age: 42, name: "Tim Duncan"})-[:follow@0 {degree: 95}]->("player101" :player{age: 36, name: "Tony Parker"})-[:serve@0 {end_year: 2019, start_year: 2018}]-
>("team215" :team{name: "Hornets"})>   | 2      |
| <("player100" :player{age: 42, name: "Tim Duncan"})-[:follow@0 {degree: 95}]->("player101" :player{age: 36, name: "Tony Parker"})-[:follow@0 {degree: 95}]->("player100" :player{age: 42,
name: "Tim Duncan"})>        | 2      |
| <("player100" :player{age: 42, name: "Tim Duncan"})-[:follow@0 {degree: 95}]->("player101" :player{age: 36, name: "Tony Parker"})-[:follow@0 {degree: 90}]->("player102" :player{age: 33,
name: "LaMarcus Aldridge"})> | 2      |
| <("player100" :player{age: 42, name: "Tim Duncan"})-[:follow@0 {degree: 95}]->("player101" :player{age: 36, name: "Tony Parker"})-[:follow@0 {degree: 95}]->("player125" :player{age: 41,
name: "Manu Ginobili"})>     | 2      |
| <("player100" :player{age: 42, name: "Tim Duncan"})-[:follow@0 {degree: 95}]->("player125" :player{age: 41, name: "Manu Ginobili"})-[:serve@0 {end_year: 2018, start_year: 2002}]-
>("team204" :team{name: "Spurs"})>   | 2      |
| <("player100" :player{age: 42, name: "Tim Duncan"})-[:follow@0 {degree: 95}]->("player125" :player{age: 41, name: "Manu Ginobili"})-[:follow@0 {degree: 90}]->("player100" :player{age: 42,
name: "Tim Duncan"})>      | 2      |
+-----------------------
```

```
nebula> MATCH (v:player{name:"Tony Parker"})-->(v2:player) \
        RETURN DISTINCT v2.player.name, "Hello"+" graphs!", v2.player.age > 35;
+--------------------+--------------------+-------------------+
| v2.player.name     | ("Hello"+" graphs!") | (v2.player.age>35) |
+--------------------+--------------------+-------------------+
| "LaMarcus Aldridge" | "Hello graphs!"     | false             |
| "Tim Duncan"       | "Hello graphs!"     | true              |
| "Manu Ginobili"    | "Hello graphs!"     | true              |
+--------------------+--------------------+-------------------+

nebula> RETURN 1+1;
+-------+
| (1+1) |
+-------+
| 2     |
+-------+

nebula> RETURN 3 > 1;
+-------+
| (3>1) |
+-------+
| true  |
+-------+

nebula> RETURN 1+1, rand32(1, 5);
+-------+------------+
| (1+1) | rand32(1,5) |
+-------+------------+
| 2     | 1          |
+-------+------------+
```

返回唯一字段

使用 `DISTINCT` 可以删除结果集中的重复字段。

```
# 未使用 DISTINCT。
nebula> MATCH (v:player{name:"Tony Parker"})--(v2:player) \
        RETURN v2.player.name, v2.player.age;
+--------------------+--------+
| v2.name            | v2.age |
+--------------------+--------+
| "Tim Duncan"       | 42     |
| "LaMarcus Aldridge" | 33     |
| "Marco Belinelli"  | 32     |
| "Boris Diaw"       | 36     |
| "Dejounte Murray"  | 29     |
| "Tim Duncan"       | 42     |
| "LaMarcus Aldridge" | 33     |
| "Manu Ginobili"    | 41     |
+--------------------+--------+

# 使用 DISTINCT。
nebula> MATCH (v:player{name:"Tony Parker"})--(v2:player) \
        RETURN DISTINCT v2.player.name, v2.player.age;
+--------------------+--------+
| v2.name            | v2.age |
+--------------------+--------+
| "Tim Duncan"       | 42     |
| "LaMarcus Aldridge" | 33     |
| "Marco Belinelli"  | 32     |
| "Boris Diaw"       | 36     |
| "Dejounte Murray"  | 29     |
| "Manu Ginobili"    | 41     |
+--------------------+--------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.7.6 TTL

TTL（Time To Live）指定属性的存活时间，超时后，该属性就会过期。

openCypher 兼容性

本文操作仅适用于原生 nGQL。

注意事项

- 不能修改带有 TTL 选项的属性的 Schema。
- TTL 和 INDEX 共存问题：
- 如果一个 Tag/Edge type 的其中一属性已有 INDEX，则不能为其设置 TTL，也不能为该 Tag 的其他属性设置 TTL。
- 如果已有 TTL，可以再添加 INDEX。

属性过期

点属性过期

点属性过期有如下影响：

- 如果一个点仅有一个 Tag，点上的一个属性过期，点也会过期。
- 如果一个点有多个 Tag，点上的一个属性过期，和该属性相同 Tag 的其他属性也会过期，但是点不会过期，点上其他 Tag 的属性保持不变。

边属性过期

因为一条边仅有一个 Edge type，边上的一个属性过期，边也会过期。

过期处理

属性过期后，对应的过期数据仍然存储在硬盘上，但是查询时会过滤过期数据。

Nebula Graph 自动删除过期数据后，会在下一次 Compaction 过程中回收硬盘空间。

> **Note**
>
> 如果关闭 TTL 选项，上一次 Compaction 之后的过期数据将可以被查询到。

TTL 选项

nGQL 支持的 TTL 选项如下。

选项	说明
`ttl_col`	指定要设置存活时间的属性。属性的数据类型必须是 `int` 或者 `timestamp` 。
`ttl_duration`	指定时间戳差值，单位：秒。时间戳差值必须为 64 位非负整数。属性值和时间戳差值之和如果小于当前时间戳，属性就会过期。如果 `ttl_duration` 为 `0` ，属性永不过期。

使用 TTL 选项

TAG 或 EDGE TYPE 已存在

如果 Tag 和 Edge type 已经创建，请使用 `ALTER` 语句更新 Tag 或 Edge type。

2022 vesoft Inc.

```
# 创建 Tag。
nebula> CREATE TAG IF NOT EXISTS t1 (a timestamp);

# ALTER 修改 Tag，添加 TTL 选项。
nebula> ALTER TAG t1 TTL_COL = "a", TTL_DURATION = 5;

# 插入点，插入后 5 秒过期。
nebula> INSERT VERTEX t1(a) VALUES "101":(now());
```

TAG 或 **EDGE TYPE** 不存在

创建 Tag 或 Edge type 时可以同时设置 TTL 选项。详情请参见 CREATE TAG 和 CREATE EDGE。

```
# 创建 Tag 并设置 TTL 选项。
nebula> CREATE TAG IF NOT EXISTS t2(a int, b int, c string) TTL_DURATION= 100, TTL_COL = "a";

# 插入点。过期时间戳为 1648197238（1648197138 + 100）。
nebula> INSERT VERTEX t2(a, b, c) VALUES "102":(1648197138, 30, "Hello");
```

删除存活时间

删除存活时间可以使用如下几种方法:

- 删除设置存活时间的属性。

  ```
  nebula> ALTER TAG t1 DROP (a);
  ```

- 设置 `ttl_col` 为空字符串。

  ```
  nebula> ALTER TAG t1 TTL_COL = "";
  ```

- 设置 `ttl_duration` 为 `0`。本操作可以保留 TTL 选项，属性永不过期，且属性的 Schema 无法修改。

  ```
  nebula> ALTER TAG t1 TTL_DURATION = 0;
  ```

最后更新: June 20, 2023

2022 vesoft Inc.

4.7.7 WHERE

`WHERE` 子句可以根据条件过滤输出结果。

`WHERE` 子句通常用于如下查询：

- 原生 nGQL，例如 `GO` 和 `LOOKUP` 语句。
- openCypher 方式，例如 `MATCH` 和 `WITH` 语句。

openCypher 兼容性

过滤 Rank 是原生 nGQL 功能。如需在 openCypher 兼容语句中直接获取 Rank 值，可以使用 rank() 函数，例如 `MATCH (:player)-[e:follow]->() RETURN rank(e);` 。

基础用法

> **Note**
>
> 下文示例中的 `$$` 、`$^` 等是引用符号，详情请参见引用符。

用布尔运算符定义条件

在 `WHERE` 子句中使用布尔运算符 `NOT` 、`AND` 、`OR` 和 `XOR` 定义条件。关于运算符的优先级，请参见运算符优先级。

```
nebula> MATCH (v:player) \
        WHERE v.player.name == "Tim Duncan" \
        XOR (v.player.age < 30 AND v.player.name == "Yao Ming") \
        OR NOT (v.player.name == "Yao Ming" OR v.player.name == "Tim Duncan") \
        RETURN v.player.name, v.player.age;
+-------------------------+--------------+
| v.player.name           | v.player.age |
+-------------------------+--------------+
| "Danny Green"           | 31           |
| "Tiago Splitter"        | 34           |
| "David West"            | 38           |
...
```

```
nebula> GO FROM "player100" \
        OVER follow \
        WHERE properties(edge).degree > 90 \
        OR properties($$).age != 33 \
        AND properties($$).name != "Tony Parker" \
        YIELD properties($$);
+----------------------------------+
| properties($$)                   |
+----------------------------------+
| {age: 41, name: "Manu Ginobili"} |
+----------------------------------+
```


2022 vesoft Inc.

过滤属性

在 `WHERE` 子句中使用点或边的属性定义条件。

- 过滤点属性：

```
nebula> MATCH (v:player)-[e]->(v2) \
        WHERE v2.player.age < 25 \
        RETURN v2.player.name, v2.player.age;
+----------------------+--------------+
| v2.player.name       | v2.player.age |
+----------------------+--------------+
| "Ben Simmons"        | 22           |
| "Luka Doncic"        | 20           |
| "Kristaps Porzingis" | 23           |
+----------------------+--------------+
```

```
nebula> GO FROM "player100" OVER follow \
        WHERE $^.player.age >= 42 \
        YIELD dst(edge);
+-------------+
| dst(EDGE)   |
+-------------+
| "player101" |
| "player125" |
+-------------+
```

- 过滤边属性：

```
nebula> MATCH (v:player)-[e]->() \
        WHERE e.start_year < 2000 \
        RETURN DISTINCT v.player.name, v.player.age;
+--------------------+--------------+
| v.player.name      | v.player.age |
+--------------------+--------------+
| "Tony Parker"      | 36           |
| "Tim Duncan"       | 42           |
| "Grant Hill"       | 46           |
...
```

```
nebula> GO FROM "player100" OVER follow \
        WHERE follow.degree > 90 \
        YIELD dst(edge);
+-------------+
| dst(EDGE)   |
+-------------+
| "player101" |
| "player125" |
+-------------+
```

过滤动态计算属性

```
nebula> MATCH (v:player) \
        WHERE v[toLower("AGE")] < 21 \
        RETURN v.player.name, v.player.age;
+---------------+-------+
| v.name        | v.age |
+---------------+-------+
| "Luka Doncic" | 20    |
+---------------+-------+
```

过滤现存属性

```
nebula> MATCH (v:player) \
        WHERE exists(v.player.age) \
        RETURN v.player.name, v.player.age;
+-------------------------+--------------+
| v.player.name           | v.player.age |
+-------------------------+--------------+
| "Danny Green"           | 31           |
| "Tiago Splitter"        | 34           |
| "David West"            | 38           |
...
```

过滤 **RANK**

在 nGQL 中，如果多个边拥有相同的起始点、目的点和属性，则它们的唯一区别是 rank 值。在 `WHERE` 子句中可以使用 rank 过滤边。

```
# 创建测试数据。
nebula> CREATE SPACE IF NOT EXISTS test (vid_type=FIXED_STRING(30));
nebula> USE test;
```


2022 vesoft Inc.

```
nebula> CREATE EDGE IF NOT EXISTS e1(p1 int);
nebula> CREATE TAG IF NOT EXISTS person(p1 int);
nebula> INSERT VERTEX person(p1) VALUES "1":(1);
nebula> INSERT VERTEX person(p1) VALUES "2":(2);
nebula> INSERT EDGE e1(p1) VALUES "1"->"2"@0:(10);
nebula> INSERT EDGE e1(p1) VALUES "1"->"2"@1:(11);
nebula> INSERT EDGE e1(p1) VALUES "1"->"2"@2:(12);
nebula> INSERT EDGE e1(p1) VALUES "1"->"2"@3:(13);
nebula> INSERT EDGE e1(p1) VALUES "1"->"2"@4:(14);
nebula> INSERT EDGE e1(p1) VALUES "1"->"2"@5:(15);
nebula> INSERT EDGE e1(p1) VALUES "1"->"2"@6:(16);

# 通过 rank 过滤边，查找 rank 大于 2 的边。
nebula> GO FROM "1" \
        OVER e1 \
        WHERE rank(edge) > 2 \
        YIELD src(edge), dst(edge), rank(edge) AS Rank, properties(edge).p1 | \
        ORDER BY $-.Rank DESC;
+-----------+-----------+------+--------------------+
| src(EDGE) | dst(EDGE) | Rank | properties(EDGE).p1 |
+-----------+-----------+------+--------------------+
| "1"       | "2"       | 6    | 16                 |
| "1"       | "2"       | 5    | 15                 |
| "1"       | "2"       | 4    | 14                 |
| "1"       | "2"       | 3    | 13                 |
+-----------+-----------+------+--------------------+
```

过滤字符串

在 `WHERE` 子句中使用 `STARTS WITH` 、 `ENDS WITH` 或 `CONTAINS` 可以匹配字符串的特定部分。匹配时区分大小写。

`STARTS WITH`

`STARTS WITH` 会从字符串的起始位置开始匹配。

```
# 查询姓名以 T 开头的 player 信息。
nebula> MATCH (v:player) \
        WHERE v.player.name STARTS WITH "T" \
        RETURN v.player.name, v.player.age;
+------------------+--------------+
| v.player.name    | v.player.age |
+------------------+--------------+
| "Tony Parker"    | 36           |
| "Tiago Splitter" | 34           |
| "Tim Duncan"     | 42           |
| "Tracy McGrady"  | 39           |
+------------------+--------------+
```

如果使用小写 `t` （ `STARTS WITH "t"` ），会返回空集，因为数据库中没有以小写 `t` 开头的姓名。

```
nebula> MATCH (v:player) \
        WHERE v.player.name STARTS WITH "t" \
        RETURN v.player.name, v.player.age;
+---------------+--------------+
| v.player.name | v.player.age |
+---------------+--------------+
+---------------+--------------+
Empty set (time spent 5080/6474 us)
```

`ENDS WITH`

`ENDS WITH` 会从字符串的结束位置开始匹配。

```
nebula> MATCH (v:player) \
        WHERE v.player.name ENDS WITH "r" \
        RETURN v.player.name, v.player.age;
+------------------+--------------+
| v.player.name    | v.player.age |
+------------------+--------------+
| "Tony Parker"    | 36           |
| "Tiago Splitter" | 34           |
| "Vince Carter"   | 42           |
+------------------+--------------+
```

`CONTAINS`

`CONTAINS` 会检查关键字是否匹配字符串的某一部分。

```
nebula> MATCH (v:player) \
        WHERE v.player.name CONTAINS "Pa" \
        RETURN v.player.name, v.player.age;
+---------------+--------------+
| v.player.name | v.player.age |
```


2022 vesoft Inc.

```
+--------------+-------------+
| "Paul George" | 28          |
| "Tony Parker" | 36          |
| "Paul Gasol"  | 38          |
| "Chris Paul"  | 33          |
+--------------+-------------+
```

结合 **NOT** 使用

用户可以结合布尔运算符 `NOT` 一起使用，否定字符串匹配条件。

```
nebula> MATCH (v:player) \
        WHERE NOT v.player.name ENDS WITH "R" \
        RETURN v.player.name, v.player.age;
+------------------------+-------------+
| v.player.name          | v.player.age |
+------------------------+-------------+
| "Danny Green"          | 31          |
| "Tiago Splitter"       | 34          |
| "David West"           | 38          |
| "Russell Westbrook"    | 30          |
...
```

过滤列表

匹配列表中的值

使用 `IN` 运算符检查某个值是否在指定列表中。

```
nebula> MATCH (v:player) \
        WHERE v.player.age IN range(20,25) \
        RETURN v.player.name, v.player.age;
+------------------------+-------------+
| v.player.name          | v.player.age |
+------------------------+-------------+
| "Ben Simmons"          | 22          |
| "Giannis Antetokounmpo" | 24          |
| "Kyle Anderson"        | 25          |
| "Joel Embiid"          | 25          |
| "Kristaps Porzingis"   | 23          |
| "Luka Doncic"          | 20          |
+------------------------+-------------+

nebula> LOOKUP ON player \
        WHERE player.age IN [25,28]  \
        YIELD properties(vertex).name, properties(vertex).age;
+------------------------+-----------------------+
| properties(VERTEX).name | properties(VERTEX).age |
+------------------------+-----------------------+
| "Kyle Anderson"        | 25                    |
| "Damian Lillard"       | 28                    |
| "Joel Embiid"          | 25                    |
| "Paul George"          | 28                    |
| "Ricky Rubio"          | 28                    |
+------------------------+-----------------------+
```

结合 **NOT** 使用

```
nebula> MATCH (v:player) \
        WHERE v.player.age NOT IN range(20,25) \
        RETURN v.player.name AS Name, v.player.age AS Age \
        ORDER BY Age;
+--------------------+-----+
| Name               | Age |
+--------------------+-----+
| "Kyrie Irving"     | 26  |
| "Cory Joseph"      | 27  |
| "Damian Lillard"   | 28  |
| "Paul George"      | 28  |
| "Ricky Rubio"      | 28  |
...
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.7.8 YIELD

`YIELD` 定义 nGQL 查询的输出结果。

`YIELD` 可以引导子句或语句：

- `YIELD` 子句用于原生 nGQL 语句中，例如 `GO` 、 `FETCH` 或 `LOOKUP` ，必须通过 `YIELD` 子句定义返回结果。
- `YIELD` 语句可以在独立查询或复合查询中使用。

openCypher 兼容性

本文操作仅适用于原生 nGQL。关于 openCypher 方式如何定义输出结果，请参见 `RETURN` 。

`YIELD` 在 nGQL 和 openCypher 中有不同的函数：

- 在 openCypher 中， `YIELD` 用于在 `CALL[...YIELD]` 子句中指定过程调用的输出。

  Note

  nGQL 不支持 `CALL[...YIELD]` 。

- 在 nGQL 中， `YIELD` 和 openCypher 中的 `RETURN` 类似。

  Note

  下文示例中的 `$$` 、 `$-` 等是引用符号，详情请参见引用符。

YIELD 子句

语法

```
YIELD [DISTINCT] <col> [AS <alias>] [, <col> [AS <alias>] ...];
```

参数	说明
`DISTINCT`	聚合输出结果，返回去重后的结果集。
`col`	要返回的字段。如果没有为字段设置别名，返回结果中的列名为 `col` 。
`alias`	`col` 的别名。使用关键字 `AS` 进行设置，设置后返回结果中的列名为该别名。

使用 YIELD 子句

- `GO` 语句中使用 `YIELD` ：

```
nebula> GO FROM "player100" OVER follow \
        YIELD properties($$).name AS Friend, properties($$).age AS Age;
+-----------------+-----+
| Friend          | Age |
+-----------------+-----+
| "Tony Parker"   | 36  |
```


2022 vesoft Inc.

```
| "Manu Ginobili" | 41  |
+-----------------+-----+
```

- `FETCH` 语句中使用 `YIELD`：

```
nebula> FETCH PROP ON player "player100" \
        YIELD properties(vertex).name;
+-------------------------+
| properties(VERTEX).name |
+-------------------------+
| "Tim Duncan"            |
+-------------------------+
```

- `LOOKUP` 语句中使用 `YIELD`：

```
nebula> LOOKUP ON player WHERE player.name == "Tony Parker" \
        YIELD properties(vertex).name, properties(vertex).age;
+-------------------------+------------------------+
| properties(VERTEX).name | properties(VERTEX).age |
+-------------------------+------------------------+
| "Tony Parker"           | 36                     |
+-------------------------+------------------------+
```

YIELD 语句

语法

```
YIELD [DISTINCT] <col> [AS <alias>] [, <col> [AS <alias>] ...]
[WHERE <conditions>];
```

参数	说明
`DISTINCT`	聚合输出结果，返回去重后的结果集。
`col`	要按返回的字段。如果没有为字段设置别名，返回结果中的列名为 `col`。
`alias`	`col` 的别名。使用关键字 `AS` 进行设置，设置后返回结果中的列名为该别名。
`conditions`	在 `WHERE` 子句中设置的过滤条件。详情请参见 `WHERE`。

复合查询中使用 **YIELD** 语句

在复合查询中，`YIELD` 语句可以接收、过滤、修改之前语句的结果集，然后输出。

```
# 查找 player100 关注的 player，并计算他们的平均年龄。
nebula> GO FROM "player100" OVER follow \
        YIELD dst(edge) AS ID \
        | FETCH PROP ON player $-.ID \
        YIELD properties(vertex).age AS Age \
        | YIELD AVG($-.Age) as Avg_age, count(*)as Num_friends;
+---------+-------------+
| Avg_age | Num_friends |
+---------+-------------+
| 38.5    | 2           |
+---------+-------------+
```

```
# 查找 player101 关注的 player，返回 degree 大于 90 的 player。
nebula> $var1 = GO FROM "player101" OVER follow \
        YIELD properties(edge).degree AS Degree, dst(edge) as ID; \
        YIELD $var1.ID AS ID WHERE $var1.Degree > 90;
+-------------+
| ID          |
+-------------+
| "player100" |
| "player125" |
+-------------+
```

独立使用 **YIELD** 语句

`YIELD` 可以计算表达式并返回结果。

```
nebula> YIELD rand32(1, 6);
+-------------+
| rand32(1,6) |
+-------------+
```


2022 vesoft Inc.

```
| 3           |
+-------------+

nebula> YIELD "Hel" + "\tlo" AS string1, ", World!" AS string2;
+-------------+------------+
| string1     | string2    |
+-------------+------------+
| "Hel    lo" | ", World!" |
+-------------+------------+

nebula> YIELD hash("Tim") % 100;
+-------------------+
| (hash("Tim")%100) |
+-------------------+
| 42                |
+-------------------+

nebula> YIELD \
      CASE 2+3 \
      WHEN 4 THEN 0 \
      WHEN 5 THEN 1 \
      ELSE -1 \
      END \
      AS result;
+--------+
| result |
+--------+
| 1      |
+--------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.7.9 WITH

`WITH` 子句可以获取并处理查询前半部分的结果，并将处理结果作为输入传递给查询的后半部分。

openCypher 兼容性

本文操作仅适用于 openCypher 方式。

> **Note**
>
> 在原生 nGQL 中，有与 `WITH` 类似的管道符，但它们的工作方式不同。不要在 openCypher 方式中使用管道符，也不要在原生 nGQL 中使用 `WITH` 子句。

组成复合查询

使用 `WITH` 子句可以组合语句，将一条语句的输出转换为另一条语句的输入。

示例 1

1. 匹配一个路径。
2. 通过 `nodes()` 函数将路径上的所有点输出到一个列表。
3. 将列表拆分为行。
4. 去重后返回点的信息。

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})--() \
        WITH nodes(p) AS n \
        UNWIND n AS n1 \
        RETURN DISTINCT n1;
+-------------------------------------------------------+
| n1                                                    |
+-------------------------------------------------------+
| ("player100" :player{age: 42, name: "Tim Duncan"})    |
| ("player101" :player{age: 36, name: "Tony Parker"})   |
| ("team204" :team{name: "Spurs"})                      |
| ("player102" :player{age: 33, name: "LaMarcus Aldridge"}) |
| ("player125" :player{age: 41, name: "Manu Ginobili"}) |
| ("player104" :player{age: 32, name: "Marco Belinelli"}) |
| ("player144" :player{age: 47, name: "Shaquille O'Neal"}) |
| ("player105" :player{age: 31, name: "Danny Green"})   |
| ("player113" :player{age: 29, name: "Dejounte Murray"}) |
| ("player107" :player{age: 32, name: "Aron Baynes"})   |
| ("player109" :player{age: 34, name: "Tiago Splitter"}) |
| ("player108" :player{age: 36, name: "Boris Diaw"})    |
+-------------------------------------------------------+
```

示例 2

1. 匹配点 ID 为 `player100` 的点。
2. 通过 `labels()` 函数将点的所有 Tag 输出到一个列表。
3. 将列表拆分为行。
4. 返回结果。

```
nebula> MATCH (v) \
        WHERE id(v)=="player100" \
        WITH labels(v) AS tags_unf \
        UNWIND tags_unf AS tags_f \
        RETURN tags_f;
+----------+
| tags_f   |
+----------+
| "player" |
+----------+
```

过滤聚合查询

`WITH` 可以在聚合查询中作为过滤器使用。

2022 vesoft Inc.

```
nebula> MATCH (v:player)-->(v2:player) \
        WITH DISTINCT v2 AS v2, v2.player.age AS Age \
        ORDER BY Age \
        WHERE Age<25 \
        RETURN v2.player.name AS Name, Age;
+----------------------+-----+
| Name                 | Age |
+----------------------+-----+
| "Luka Doncic"        | 20  |
| "Ben Simmons"        | 22  |
| "Kristaps Porzingis" | 23  |
+----------------------+-----+
```

collect() 之前处理输出

在 `collect()` 函数将输出结果转换为列表之前，可以使用 `WITH` 子句排序和限制输出结果。

```
nebula> MATCH (v:player) \
        WITH v.player.name AS Name \
        ORDER BY Name DESC \
        LIMIT 3 \
        RETURN collect(Name);
+-------------------------------------------------+
| collect(Name)                                   |
+-------------------------------------------------+
| ["Yao Ming", "Vince Carter", "Tracy McGrady"] |
+-------------------------------------------------+
```

结合 **RETURN** 语句使用

在 `WITH` 子句中设置别名，并通过 `RETURN` 子句输出结果。

```
nebula> WITH [1, 2, 3] AS `list` RETURN 3 IN `list` AS r;
+------+
| r    |
+------+
| true |
+------+

nebula> WITH 4 AS one, 3 AS two RETURN one > two AS result;
+--------+
| result |
+--------+
| true   |
+--------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.7.10 UNWIND

`UNWIND` 语句可以将列表拆分为单独的行，列表中的每个元素为一行。

`UNWIND` 可以作为单独语句或语句中的子句使用。

语法

```
UNWIND <list> AS <alias> <RETURN clause>;
```

拆分列表

```
nebula> UNWIND [1,2,3] AS n RETURN n;
+---+
| n |
+---+
| 1 |
| 2 |
| 3 |
+---+
```

返回去重列表

在 `UNWIND` 语句中使用 `WITH DISTINCT` 可以将列表中的重复项忽略，返回去重后的结果。

示例 **1**

1. 拆分列表 `[1,1,2,2,3,3]` 。
2. 删除重复行。
3. 排序行。
4. 将行转换为列表。

```
nebula> WITH [1,1,2,2,3,3] AS n \
        UNWIND n AS r \
        WITH DISTINCT r AS r \
        ORDER BY r \
        RETURN collect(r);
+------------+
| collect(r) |
+------------+
| [1, 2, 3]  |
+------------+
```

示例 **2**

1. 将匹配路径上的顶点输出到列表中。
2. 拆分列表。
3. 删除重复行。
4. 将行转换为列表。

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})--(v2) \
        WITH nodes(p) AS n \
        UNWIND n AS r \
        WITH DISTINCT r AS r \
        RETURN collect(r);
+-------------------------------------------------------------------------------------------------------------------+
| collect(r)                                                                                                        |
+-------------------------------------------------------------------------------------------------------------------+
| [("player100" :player{age: 42, name: "Tim Duncan"}), ("player101" :player{age: 36, name: "Tony Parker"}),         |
|("team204" :team{name: "Spurs"}), ("player102" :player{age: 33, name: "LaMarcus Aldridge"}),                       |
|("player125" :player{age: 41, name: "Manu Ginobili"}), ("player104" :player{age: 32, name: "Marco Belinelli"}),    |
|("player144" :player{age: 47, name: "Shaquile O'Neal"}), ("player105" :player{age: 31, name: "Danny Green"}),      |
|("player113" :player{age: 29, name: "Dejounte Murray"}), ("player107" :player{age: 32, name: "Aron Baynes"}),      |
|("player109" :player{age: 34, name: "Tiago Splitter"}), ("player108" :player{age: 36, name: "Boris Diaw"})]        |
+-------------------------------------------------------------------------------------------------------------------+
```


2022 vesoft Inc.

最后更新: June 20, 2023

2022 vesoft Inc.

4.8 图空间语句

4.8.1 CREATE SPACE

图空间是 Nebula Graph 中彼此隔离的图数据集合，与 MySQL 中的 database 概念类似。`CREATE SPACE` 语句可以创建一个新的图空间，或者克隆现有图空间的 Schema。

前提条件

只有 God 角色的用户可以执行 `CREATE SPACE` 语句。详情请参见身份验证。

语法

创建图空间

```
CREATE SPACE [IF NOT EXISTS] <graph_space_name> (
    [partition_num = <partition_number>,]
    [replica_factor = <replica_number>,]
    vid_type = {FIXED_STRING(<N>) | INT[64]}
    )
    [COMMENT = '<comment>'];
```

参数	说明
`IF NOT EXISTS`	检测待创建的图空间是否存在，只有不存在时，才会创建图空间。仅检测图空间的名称，不会检测具体属性。
`<graph_space_name>`	在 Nebula Graph 实例中唯一标识一个图空间。图空间名称以英文字母开头，支持 1~4 字节的 UTF-8 编码字符，包括英文字母（区分大小写）、数字、中文等，但是不包括除下划线外的特殊字符。使用特殊字符或保留关键字时，需要用反引号（`）包围，详情参见关键字和保留字。
`partition_num`	指定图空间的分片数量。建议设置为集群中硬盘数量的 20 倍（HDD 硬盘建议为 2 倍）。例如集群中有 3 个硬盘，建议设置 60 个分片。默认值为 100。
`replica_factor`	指定每个分片的副本数量。建议在生产环境中设置为 3，在测试环境中设置为 1。由于需要基于多数表决，副本数量必须是奇数。默认值为 1。
`vid_type`	必选参数。指定点 ID 的数据类型。可选值为 `FIXED_STRING(<N>)` 和 `INT64`。`INT` 等同于 `INT64`。`FIXED_STRING(<N>)` 表示数据类型为字符串，最大长度为 `N`，超出长度会报错；`INT64` 表示数据类型为整数。
`COMMENT`	图空间的描述。最大为 256 字节。默认无描述。

Caution

- 如果将副本数设置为 1，用户将无法使用 BALANCE 命令为 Nebula Graph 的存储服务平衡负载或扩容。
- VID 类型变更与长度限制：
- 在 Nebula Graph 1.x 中，VID 的类型只能为 `INT64`，不支持字符型；在 Nebula Graph 2.x 中，VID 的类型支持 `INT64` 和 `FIXED_STRING(<N>)`。请在创建图空间时指定 VID 类型，使用 `INSERT` 语句时也需要保持一致，否则会报错 VID 类型不匹配 `Wrong vertex id type: 1001`。
- VID 最大长度必须为 `N`，不可任意长度；超过该长度也会报错 `The VID must be a 64-bit integer or a string fitting space vertex id length limit.`。

版本兼容性

2.5.0 之前的 2.x 版本中，`vid_type` 不是必选参数，默认为 `FIXED_STRING(8)`。

2022 vesoft Inc.

Note

`graph_space_name`，`partition_num`，`replica_factor`，`vid_type`，`comment` 设置后就无法改变。除非 `DROP SPACE`，并重新 `CREATE SPACE`。

克隆图空间

```
CREATE SPACE <new_graph_space_name> AS <old_graph_space_name>;
```

参数	说明
`<new_graph_space_name>`	目标图空间名称。该图空间必须未创建。图空间名称以英文字母开头，支持 1~4 字节的 UTF-8 编码字符，包括英文字母（区分大小写）、数字、中文等，但是不包括除下划线外的特殊字符。使用保留关键字时，需要用反引号（`）包围，详情参见关键字和保留字。创建时会克隆 `<old_graph_space_name>` 图空间的 Schema，包括图空间本身参数（分片数量、副本数量等）、Tag、Edge type 和 原生索引。
`<old_graph_space_name>`	原始图空间名称。该图空间必须已存在。

示例

```
# 仅指定 VID 类型，其他选项使用默认值。
nebula> CREATE SPACE IF NOT EXISTS my_space_1 (vid_type=FIXED_STRING(30));

# 指定分片数量、副本数量和 VID 类型。
nebula> CREATE SPACE IF NOT EXISTS my_space_2 (partition_num=15, replica_factor=1, vid_type=FIXED_STRING(30));

# 指定分片数量、副本数量和 VID 类型，并添加描述。
nebula> CREATE SPACE IF NOT EXISTS my_space_3 (partition_num=15, replica_factor=1, vid_type=FIXED_STRING(30)) comment="测试图空间";

# 克隆图空间。
nebula> CREATE SPACE IF NOT EXISTS my_space_4 as my_space_3;
nebula> SHOW CREATE SPACE my_space_4;
+-------------+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| Space       | Create Space                                                                                                                                                                                              |
+-------------+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| "my_space_4" | "CREATE SPACE `my_space_4` (partition_num = 15, replica_factor = 1, charset = utf8, collate = utf8_bin, vid_type = FIXED_STRING(30)) ON default comment = '测试图空间'"  |
+-------------+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
```

Caution

立刻尝试使用刚创建的图空间可能会失败。因为创建是异步实现的。为确保数据同步，后续操作能顺利进行，请等待 2 个心跳周期（20 秒）。如果需要修改心跳间隔，请为所有配置文件修改参数 `heartbeat_interval_secs`。但过短的心跳周期（<5 秒）可能会导致分布式系统中的机器误判对端失联。

检查分片分布情况

在大型集群中，由于启动时间不同，分片的分布可能不均衡。用户可以执行如下命令检查分片的分布情况：

```
nebula> SHOW HOSTS;
+-------------+------+----------+----------+--------------+------------------------------+------------------------------+---------+
| Host        | Port | HTTP port | Status   | Leader count | Leader distribution          | Partition distribution       | Version |
+-------------+------+----------+----------+--------------+------------------------------+------------------------------+---------+
| "storaged0" | 9779 | 19669     | "ONLINE" | 8            | "basketballplayer:3, test:5"   | "basketballplayer:10, test:10" | "3.1.0" |
| "storaged1" | 9779 | 19669     | "ONLINE" | 9            | "basketballplayer:4, test:5"   | "basketballplayer:10, test:10" | "3.1.0" |
| "storaged2" | 9779 | 19669     | "ONLINE" | 3            | "basketballplayer:3"           | "basketballplayer:10, test:10" | "3.1.0" |
+-------------+------+----------+----------+--------------+------------------------------+------------------------------+---------+
```

如果需要均衡负载，请执行如下命令：

```
nebula> BALANCE LEADER;
nebula> SHOW HOSTS;
+-------------+------+----------+----------+--------------+------------------------------+------------------------------+---------+
| Host        | Port | HTTP port | Status   | Leader count | Leader distribution          | Partition distribution       | Version |
+-------------+------+----------+----------+--------------+------------------------------+------------------------------+---------+
| "storaged0" | 9779 | 19669     | "ONLINE" | 7            | "basketballplayer:3, test:4"   | "basketballplayer:10, test:10" | "3.1.0" |
| "storaged1" | 9779 | 19669     | "ONLINE" | 7            | "basketballplayer:4, test:3"   | "basketballplayer:10, test:10" | "3.1.0" |
| "storaged2" | 9779 | 19669     | "ONLINE" | 6            | "basketballplayer:3, test:3"   | "basketballplayer:10, test:10" | "3.1.0" |
+-------------+------+----------+----------+--------------+------------------------------+------------------------------+---------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.8.2 USE

`USE` 语句可以指定一个图空间，或切换到另一个图空间，将其作为后续查询的工作空间。

前提条件

执行 `USE` 语句指定图空间时，需要当前登录的用户拥有指定图空间的权限，否则会报错。

语法

```
USE <graph_space_name>;
```

示例

```
# 创建示例空间。
nebula> CREATE SPACE IF NOT EXISTS space1 (vid_type=FIXED_STRING(30));
nebula> CREATE SPACE IF NOT EXISTS space2 (vid_type=FIXED_STRING(30));

# 指定图空间 space1 作为工作空间。
nebula> USE space1;

# 切换到图空间 space2。检索 space2 时，无法从 space1 读取任何数据，检索的点和边与 space1 无关。
nebula> USE space2;
```



不能在一条语句中同时操作两个图空间。

与 Fabric Cypher 不同，Nebula Graph 的图空间彼此之间是完全隔离的，将一个图空间作为工作空间后，用户无法访问其他空间。使用新图空间的唯一方法是通过 `USE` 语句切换。而在 Fabric Cypher 中可以在一条语句中（`USE + CALL` 语法）使用两个图空间。

最后更新: June 20, 2023

2022 vesoft Inc.

4.8.3 SHOW SPACES

`SHOW SPACES` 语句可以列出 Nebula Graph 示例中的所有图空间。

语法

```
SHOW SPACES;
```

示例

```
nebula> SHOW SPACES;
+--------------------+
| Name               |
+--------------------+
| "cba"              |
| "basketballplayer" |
+--------------------+
```

创建图空间请参见 CREATE SPACE。

最后更新: June 20, 2023

2022 vesoft Inc.

4.8.4 DESCRIBE SPACE

`DESCRIBE SPACE` 语句可以显示指定图空间的信息。

语法

你可以用 `DESC` 作为 `DESCRIBE` 的缩写。

```
DESC[RIBE] SPACE <graph_space_name>;
```

示例

```
nebula> DESCRIBE SPACE basketballplayer;
+----+--------------------+------------------+----------------+---------+-----------+--------------------+---------+
| ID | Name               | Partition Number | Replica Factor | Charset | Collate   | Vid Type           | Comment |
+----+--------------------+------------------+----------------+---------+-----------+--------------------+---------+
| 1  | "basketballplayer" | 10               | 1              | "utf8"  | "utf8_bin" | "FIXED_STRING(32)" |         |
+----+--------------------+------------------+----------------+---------+-----------+--------------------+---------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.8.5 CLEAR SPACE

`CLEAR SPACE` 语句用于清空图空间中的点和边，但不会删除图空间本身以及其中的 Schema 信息。

权限要求

只有 God 角色的用户可以执行 `CLEAR SPACE` 语句。

注意事项

- 数据清除后，如无备份，无法恢复。使用该功能务必谨慎。
- `CLEAR SPACE` 不是原子性操作。如果执行出错，请重新执行，避免残留数据。
- 图空间中的数据量越大，`CLEAR SPACE` 消耗的时间越长。如果 `CLEAR SPACE` 的执行因客户端连接超时而失败，可以增大 Graph 服务配置中 `storage_client_timeout_ms` 参数的值。
- 在 `CLEAR SPACE` 的执行过程中，向该图空间写入数据的行为不会被自动禁止。这样的写入行为可能导致 `CLEAR SPACE` 清除数据不完全，残留的数据也可能受到损坏。

  Enterpriseonly

  - 社区版 Nebula Graph 不支持在运行 `CLEAR SPACE` 的同时禁止写入。
  - 企业版 Nebula Graph 支持先运行 `SET VARIABLE read_only=true` 阻止向 Nebula Graph 写入数据，再运行 `CLEAR SPACE` 。数据清除成功后运行 `SET VARIABLE read_only=false` 即可重新允许写入。

语法

```
CLEAR SPACE [IF EXISTS] <space_name>;
```

参数/选项	说明
`IF EXISTS`	检查待清空的图空间是否存在，如果图空间存在，则继续执行清空操作；如果图空间不存在，则完成执行，并且提示执行成功，不会提示图空间不存在。若不设置该选项，当图空间不存在时，`CLEAR SPACE` 语句会执行失败，系统会报错。
`space_name`	要清除数据的图空间名称。

示例：

```
CLEAR SPACE basketballplayer;
```

保留的数据

图空间中，`CLEAR SPACE` 不会删除的数据包括：

- Tag 信息。
- Edge type 信息。
- 原生索引和全文索引的元数据。

下面的执行示例明确展示了 `CLEAR SPACE` 会删除与保留的数据。

```
# 进入图空间 basketballplayer。
nebula [(none)]> use basketballplayer;
Execution succeeded

# 查看 Tag 和 Edge type。
nebula[basketballplayer]> SHOW TAGS;
+----------+
| Name     |
+----------+
```


2022 vesoft Inc.

```
| "player" |
| "team"   |
+----------+
Got 2 rows

nebula[basketballplayer]> SHOW EDGES;
+----------+
| Name     |
+----------+
| "follow" |
| "serve"  |
+----------+
Got 2 rows

# 统计图空间中的数据。
nebula[basketballplayer]> SUBMIT JOB STATS;
+------------+
| New Job Id |
+------------+
| 4          |
+------------+
Got 1 rows

# 查看统计结果。
nebula[basketballplayer]> SHOW STATS;
+---------+------------+-------+
| Type    | Name       | Count |
+---------+------------+-------+
| "Tag"   | "player"   | 51    |
| "Tag"   | "team"     | 30    |
| "Edge"  | "follow"   | 81    |
| "Edge"  | "serve"    | 152   |
| "Space" | "vertices" | 81    |
| "Space" | "edges"    | 233   |
+---------+------------+-------+
Got 6 rows

# 查看 Tag 索引。
nebula[basketballplayer]> SHOW TAG INDEXES;
+------------------+----------+----------+
| Index Name       | By Tag   | Columns  |
+------------------+----------+----------+
| "player_index_0" | "player" | []       |
| "player_index_1" | "player" | ["name"] |
+------------------+----------+----------+
Got 2 rows

# ----------------------- CLEAR SPACE 分割线 -----------------------
# 执行 CLEAR SPACE 清空图空间 basketballplayer。
nebula[basketballplayer]> CLEAR SPACE basketballplayer;
Execution succeeded

# 更新统计信息。
nebula[basketballplayer]> SUBMIT JOB STATS;
+------------+
| New Job Id |
+------------+
| 5          |
+------------+
Got 1 rows

# 查看统计信息。Tag 和 Edge type 还存在，但所有的点和边都没了。
nebula[basketballplayer]> SHOW STATS;
+---------+------------+-------+
| Type    | Name       | Count |
+---------+------------+-------+
| "Tag"   | "player"   | 0     |
| "Tag"   | "team"     | 0     |
| "Edge"  | "follow"   | 0     |
| "Edge"  | "serve"    | 0     |
| "Space" | "vertices" | 0     |
| "Space" | "edges"    | 0     |
+---------+------------+-------+
Got 6 rows

# 查看 Tag 索引，它们依然存在。
nebula[basketballplayer]> SHOW TAG INDEXES;
+------------------+----------+----------+
| Index Name       | By Tag   | Columns  |
+------------------+----------+----------+
| "player_index_0" | "player" | []       |
| "player_index_1" | "player" | ["name"] |
+------------------+----------+----------+
Got 2 rows (time spent 523/978 us)
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.8.6 DROP SPACE

`DROP SPACE` 语句用于删除指定图空间以及其中的所有信息。

前提条件

只有 God 角色的用户可以执行 `DROP SPACE` 语句。详情请参见身份验证。

语法

```
DROP SPACE [IF EXISTS] <graph_space_name>;
```

`IF EXISTS` 关键字可以检测待删除的图空间是否存在，只有存在时，才会删除图空间。

> 旧版本兼容性
>
> 在 Nebula Graph 3.1.0 版本前， `DROP SPACE` 语句不会删除硬盘上对应图空间的目录和文件。

> **Caution**
>
> 请谨慎执行删除图空间操作。

最后更新: June 20, 2023

2022 vesoft Inc.

4.9 Tag 语句

4.9.1 CREATE TAG

`CREATE TAG` 语句可以通过指定名称创建一个 Tag。

OpenCypher 兼容性

nGQL 中的 Tag 和 openCypher 中的 Label 相似，但又有所不同，例如它们的创建方式。

- openCypher 中的 Label 需要在 `CREATE` 语句中与点一起创建。
- nGQL 中的 Tag 需要使用 `CREATE TAG` 语句独立创建。Tag 更像是 MySQL 中的表。

前提条件

执行 `CREATE TAG` 语句需要当前登录的用户拥有指定图空间的创建 Tag 权限，否则会报错。

语法

创建 Tag 前，需要先用 `USE` 语句指定工作空间。

```
CREATE TAG [IF NOT EXISTS] <tag_name>
    (
      <prop_name> <data_type> [NULL | NOT NULL] [DEFAULT <default_value>] [COMMENT '<comment>']
      [{, <prop_name> <data_type> [NULL | NOT NULL] [DEFAULT <default_value>] [COMMENT '<comment>']} ...]
    )
    [TTL_DURATION = <ttl_duration>]
    [TTL_COL = <prop_name>]
    [COMMENT = '<comment>'];
```

参数	说明
`IF NOT EXISTS`	检测待创建的 Tag 是否存在，只有不存在时，才会创建 Tag。仅检测 Tag 的名称，不会检测具体属性。
`<tag_name>`	每个图空间内的 Tag 必须是唯一的。Tag 名称设置后无法修改。Tag 名称以英文字母开头，支持 1~4 字节的 UTF-8 编码字符，包括英文字母（区分大小写）、数字、中文等，但是不包括除下划线外的特殊字符。使用特殊字符或保留关键字时，需要用反引号（`）包围，详情参见关键字和保留字。
`<prop_name>`	属性名称。每个 Tag 中的属性名称必须唯一。属性的命名规则与 Tag 相同。
`<data_type>`	属性的数据类型，目前支持数值、布尔、字符串以及日期与时间。
`NULL | NOT NULL`	指定属性值是否支持为 `NULL`。默认值为 `NULL`。
`DEFAULT`	指定属性的默认值。默认值可以是一个文字值或 Nebula Graph 支持的表达式。如果插入点时没有指定某个属性的值，则使用默认值。
`COMMENT`	对单个属性或 Tag 的描述。最大为 256 字节。默认无描述。
`TTL_DURATION`	指定时间戳差值，单位：秒。时间戳差值必须为 64 位非负整数。属性值和时间戳差值之和如果小于当前时间戳，属性就会过期。默认值为 `0`，表示属性永不过期。
`TTL_COL`	指定要设置存活时间的属性。属性的数据类型必须是 `int` 或者 `timestamp`。一个 Tag 只能指定一个字段为 `TTL_COL`。更多 TTL 的信息请参见 TTL。

示例

```
nebula> CREATE TAG IF NOT EXISTS player(name string, age int);

# 创建没有属性的 Tag。
nebula> CREATE TAG IF NOT EXISTS no_property();

# 创建包含默认值的 Tag。
```


2022 vesoft Inc.

```
nebula> CREATE TAG IF NOT EXISTS player_with_default(name string, age int DEFAULT 20);

# 对字段 create_time 设置 TTL 为 100 秒。
nebula> CREATE TAG IF NOT EXISTS woman(name string, age int, \
        married bool, salary double, create_time timestamp) \
        TTL_DURATION = 100, TTL_COL = "create_time";
```

创建 **Tag** 说明

尝试使用新创建的 Tag 可能会失败，因为创建是异步实现的。为确保数据同步，后续操作能顺利进行，请等待 2 个心跳周期（20 秒）。

如果需要修改心跳间隔，请为所有配置文件修改参数 `heartbeat_interval_secs` 。

最后更新: June 20, 2023

2022 vesoft Inc.

4.9.2 DROP TAG

`DROP TAG` 语句可以删除当前工作空间内所有点上的指定 Tag。

点可以有一个或多个 Tag。

- 如果某个点只有一个 Tag，删除这个 Tag 后，该点变为无 Tag 的点。
- 如果某个点有多个 Tag，删除其中一个 Tag，仍然可以访问这个点，但是无法访问已删除 Tag 所定义的所有属性。

删除 Tag 操作仅删除 Schema 数据，硬盘上的文件或目录不会立刻删除，而是在下一次 Compaction 操作时删除。

前提条件

- 登录的用户必须拥有对应权限才能执行 `DROP TAG` 语句。详情请参见内置角色权限。
- 确保 Tag 不包含任何索引，否则 `DROP TAG` 时会报冲突错误 `[ERROR (-1005)]: Conflict!` 。删除索引请参见 drop index。

语法

```
DROP TAG [IF EXISTS] <tag_name>;
```

- `IF EXISTS`：检测待删除的 Tag 是否存在，只有存在时，才会删除 Tag。
- `tag_name`：指定要删除的 Tag 名称。一次只能删除一个 Tag。

示例

```
nebula> CREATE TAG IF NOT EXISTS test(p1 string, p2 int);
nebula> DROP TAG test;
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.9.3 ALTER TAG

`ALTER TAG` 语句可以修改 Tag 的结构。例如增删属性、修改数据类型，也可以为属性设置、修改 TTL（Time-To-Live）。

前提条件

- 登录的用户必须拥有对应权限才能执行 `ALTER TAG` 语句。详情请参见内置角色权限。
- 确保要修改的属性不包含索引，否则 `ALTER TAG` 时会报冲突错误 `[ERROR (-1005)]: Conflict!` 。删除索引请参见 drop index。

语法

```
ALTER TAG <tag_name>
    <alter_definition> [[, alter_definition] ...]
    [ttl_definition [, ttl_definition] ... ]
    [COMMENT = '<comment>'];

alter_definition:
| ADD    (prop_name data_type [NULL | NOT NULL] [DEFAULT <default_value>] [COMMENT '<comment>'])
| DROP   (prop_name)
| CHANGE (prop_name data_type [NULL | NOT NULL] [DEFAULT <default_value>] [COMMENT '<comment>'])

ttl_definition:
    TTL_DURATION = ttl_duration, TTL_COL = prop_name
```

- `tag_name`： 指定要修改的 Tag 名称。一次只能修改一个 Tag。请确保要修改的 Tag 在当前工作空间中存在，否则会报错。
- 可以在一个 `ALTER TAG` 语句中使用多个 `ADD` 、 `DROP` 和 `CHANGE` 子句，子句之间用英文逗号（,）分隔。

示例

```
nebula> CREATE TAG IF NOT EXISTS t1 (p1 string, p2 int);
nebula> ALTER TAG t1 ADD (p3 int, p4 string);
nebula> ALTER TAG t1 TTL_DURATION = 2, TTL_COL = "p2";
nebula> ALTER TAG t1 COMMENT = 'test1';
nebula> ALTER TAG t1 ADD (p5 double NOT NULL DEFAULT 0.4 COMMENT 'p5') COMMENT='test2';
```

修改 **Tag** 说明

尝试使用刚修改的 Tag 可能会失败，因为修改是异步实现的。为确保数据同步，后续操作能顺利进行，请等待 2 个心跳周期（20 秒）。

如果需要修改心跳间隔，请为所有配置文件修改参数 `heartbeat_interval_secs` 。

最后更新: June 20, 2023

2022 vesoft Inc.

4.9.4 SHOW TAGS

`SHOW TAGS` 语句显示当前图空间内的所有 Tag 名称。

执行 `SHOW TAGS` 语句不需要任何权限，但是返回结果由登录的用户权限决定。

语法

```
SHOW TAGS;
```

示例

```
nebula> SHOW TAGS;
+----------+
| Name     |
+----------+
| "player" |
| "team"   |
+----------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.9.5 DESCRIBE TAG

`DESCRIBE TAG` 显示指定 Tag 的详细信息，例如字段名称、数据类型等。

前提条件

登录的用户必须拥有对应权限才能执行 `DESCRIBE TAG` 语句。详情请参见内置角色权限。

语法

```
DESC[RIBE] TAG <tag_name>;
```

`DESCRIBE` 可以缩写为 `DESC` 。

示例

```
nebula> DESCRIBE TAG player;
+--------+----------+-------+---------+---------+
| Field  | Type     | Null  | Default | Comment |
+--------+----------+-------+---------+---------+
| "name" | "string" | "YES" |         |         |
| "age"  | "int64"  | "YES" |         |         |
+--------+----------+-------+---------+---------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.9.6 DELETE TAG

`DELETE TAG` 语句可以删除指定点上的指定 Tag。

前提条件

登录的用户必须拥有对应权限才能执行 `DELETE TAG` 语句。详情请参见内置角色权限。

语法

```
DELETE TAG <tag_name_list> FROM <VID>;
```

- `tag_name_list`：指定 Tag 名称。多个 Tag 用英文逗号（,）分隔，也可以用 `*` 表示所有 Tag。
- `VID`：指定要删除 Tag 的点 ID。

示例

```
nebula> CREATE TAG IF NOT EXISTS test1(p1 string, p2 int);
nebula> CREATE TAG IF NOT EXISTS test2(p3 string, p4 int);
nebula> INSERT VERTEX test1(p1, p2),test2(p3, p4) VALUES "test":("123", 1, "456", 2);
nebula> FETCH PROP ON * "test" YIELD vertex AS v;
+-------------------------------------------------------+
| v                                                     |
+-------------------------------------------------------+
| ("test" :test1{p1: "123", p2: 1} :test2{p3: "456", p4: 2}) |
+-------------------------------------------------------+

nebula> DELETE TAG test1 FROM "test";
nebula> FETCH PROP ON * "test" YIELD vertex AS v;
+----------------------------------+
| v                                |
+----------------------------------+
| ("test" :test2{p3: "456", p4: 2}) |
+----------------------------------+

nebula> DELETE TAG * FROM "test";
nebula> FETCH PROP ON * "test" YIELD vertex AS v;
+---+
| v |
+---+
+---+
```

Compatibility

- 在 openCypher 中，可以使用 `REMOVE v:LABEL` 语句来移除该点 `v` 的 `LABEL`。
- 相同语意，但不同语法。在 nGQL 中使用 `DELETE TAG`。

最后更新: June 20, 2023

2022 vesoft Inc.

4.9.7 增加和删除标签

在 openCypher 中，有增加标签（`SET label`）和移除标签（`REMOVE label`）的功能，可以用于加速查询或者标记过程。

在 Nebula Graph 中，可以通过 Tag 变相实现相同操作，创建 Tag 并将 Tag 插入到已有的点上，就可以根据 Tag 名称快速查找点，也可以通过 `DELETE TAG` 删除某些点上不再需要的 Tag。

示例

例如在 basketballplayer 数据集中，部分篮球运动员同时也是球队股东，可以为股东 Tag `shareholder` 创建索引，方便快速查找。如果不再是股东，可以通过 `DELETE TAG` 语句删除相应运动员的股东 Tag。

```
//创建股东 Tag 和索引
nebula> CREATE TAG IF NOT EXISTS shareholder();
nebula> CREATE TAG INDEX IF NOT EXISTS shareholder_tag on shareholder();

//为点添加 Tag
nebula> INSERT VERTEX shareholder() VALUES "player100":();
nebula> INSERT VERTEX shareholder() VALUES "player101":();

//快速查询所有股东
nebula> MATCH (v:shareholder) RETURN v;
+------------------------------------------------------------------+
| v                                                                |
+------------------------------------------------------------------+
| ("player100" :player{age: 42, name: "Tim Duncan"} :shareholder{})  |
| ("player101" :player{age: 36, name: "Tony Parker"} :shareholder{}) |
+------------------------------------------------------------------+
nebula> LOOKUP ON shareholder YIELD id(vertex);
+-------------+
| id(VERTEX)  |
+-------------+
| "player100" |
| "player101" |
+-------------+

//如果 player100 不再是股东
nebula> DELETE TAG shareholder FROM "player100";
nebula> LOOKUP ON shareholder YIELD id(vertex);
+-------------+
| id(VERTEX)  |
+-------------+
| "player101" |
+-------------+
```

Note

如果插入测试数据后才创建索引，请用 `REBUILD TAG INDEX <index_name_list>;` 语句重建索引。

最后更新: June 20, 2023

2022 vesoft Inc.

4.10 Edge type 语句

4.10.1 CREATE EDGE

`CREATE EDGE` 语句可以通过指定名称创建一个 Edge type。

OpenCypher 兼容性

nGQL 中的 Edge type 和 openCypher 中的关系类型相似，但又有所不同，例如它们的创建方式。

- openCypher 中的关系类型需要在 `CREATE` 语句中与点一起创建。
- nGQL 中的 Edge type 需要使用 `CREATE EDGE` 语句独立创建。Edge type 更像是 MySQL 中的表。

前提条件

执行 `CREATE EDGE` 语句需要当前登录的用户拥有指定图空间的创建 Edge type 权限，否则会报错。

语法

创建 Edge type 前，需要先用 `USE` 语句指定工作空间。

```
CREATE EDGE [IF NOT EXISTS] <edge_type_name>
    (
      <prop_name> <data_type> [NULL | NOT NULL] [DEFAULT <default_value>] [COMMENT '<comment>']
      [{, <prop_name> <data_type> [NULL | NOT NULL] [DEFAULT <default_value>] [COMMENT '<comment>']} ...]
    )
    [TTL_DURATION = <ttl_duration>]
    [TTL_COL = <prop_name>]
    [COMMENT = '<comment>'];
```

参数	说明
`IF NOT EXISTS`	检测待创建的 Edge type 是否存在，只有不存在时，才会创建 Edge type。仅检测 Edge type 的名称，不会检测具体属性。
`<edge_type_name>`	每个图空间内的 Edge type 必须是唯一的。Edge type 名称设置后无法修改。Edge type 名称以英文字母开头，支持 1~4 字节的 UTF-8 编码字符，包括英文字母（区分大小写）、数字、中文等，但是不包括除下划线外的特殊字符。使用特殊字符或保留关键字时，需要用反引号（`）包围，详情参见关键字和保留字。
`<prop_name>`	属性名称。每个 Edge type 中的属性名称必须唯一。属性的命名规则与 Edge type 相同。
`<data_type>`	属性的数据类型，目前支持数值、布尔、字符串以及日期与时间。
`NULL | NOT NULL`	指定属性值是否支持为 `NULL`。默认值为 `NULL`。
`DEFAULT`	指定属性的默认值。默认值可以是一个文字值或 Nebula Graph 支持的表达式。如果插入边时没有指定某个属性的值，则使用默认值。
`COMMENT`	对单个属性或 Edge type 的描述。最大为 256 字节。默认无描述。
`TTL_DURATION`	指定时间戳差值，单位：秒。时间戳差值必须为 64 位非负整数。属性值和时间戳差值之和如果小于当前时间戳，属性就会过期。默认值为 `0`，表示属性永不过期。
`TTL_COL`	指定要设置存活时间的属性。属性的数据类型必须是 `int` 或者 `timestamp`。一个 Edge type 只能指定一个字段为 `TTL_COL`。更多 TTL 的信息请参见 TTL。

示例

```
nebula> CREATE EDGE IF NOT EXISTS follow(degree int);

# 创建没有属性的 Edge type。
nebula> CREATE EDGE IF NOT EXISTS no_property();

# 创建包含默认值的 Edge type。
nebula> CREATE EDGE IF NOT EXISTS follow_with_default(degree int DEFAULT 20);
```


2022 vesoft Inc.

```
# 对字段 p2 设置 TTL 为 100 秒。
nebula> CREATE EDGE IF NOT EXISTS e1(p1 string, p2 int, p3 timestamp) \
        TTL_DURATION = 100, TTL_COL = "p2";
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.10.2 DROP EDGE

`DROP EDGE` 语句可以删除当前工作空间内的指定 Edge type。

一个边只能有一个 Edge type，删除这个 Edge type 后，用户就无法访问这个边，下次 Compaction 操作时会删除该边。

删除 Edge type 操作仅删除 Schema 数据，硬盘上的文件或目录不会立刻删除，而是在下一次 Compaction 操作时删除。

前提条件

- 登录的用户必须拥有对应权限才能执行 `DROP EDGE` 语句。详情请参见内置角色权限。
- 确保 Edge type 不包含任何索引，否则 `DROP EDGE` 时会报冲突错误 `[ERROR (-1005)]: Conflict!`。删除索引请参见 drop index。

语法

```
DROP EDGE [IF EXISTS] <edge_type_name>
```

- `IF EXISTS`：检测待删除的 Edge type 是否存在，只有存在时，才会删除 Edge type。
- `edge_type_name`：指定要删除的 Edge type 名称。一次只能删除一个 Edge type。

示例

```
nebula> CREATE EDGE IF NOT EXISTS e1(p1 string, p2 int);
nebula> DROP EDGE e1;
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.10.3 ALTER EDGE

`ALTER EDGE` 语句可以修改 Edge type 的结构。例如增删属性、修改数据类型，也可以为属性设置、修改 TTL（Time-To-Live）。

前提条件

- 登录的用户必须拥有对应权限才能执行 `ALTER EDGE` 语句。详情请参见内置角色权限。
- 确保要修改的属性不包含索引，否则 `ALTER EDGE` 时会报冲突错误 `[ERROR (-1005)]: Conflict!`。删除索引请参见 drop index。

语法

```
ALTER EDGE <edge_type_name>
    <alter_definition> [, alter_definition] ...]
    [ttl_definition [, ttl_definition] ... ]
    [COMMENT = '<comment>'];

alter_definition:
| ADD    (prop_name data_type)
| DROP   (prop_name)
| CHANGE (prop_name data_type)

ttl_definition:
    TTL_DURATION = ttl_duration, TTL_COL = prop_name
```

- `edge_type_name`：指定要修改的 Edge type 名称。一次只能修改一个 Edge type。请确保要修改的 Edge type 在当前工作空间中存在，否则会报错。
- 可以在一个 `ALTER EDGE` 语句中使用多个 `ADD`、`DROP` 和 `CHANGE` 子句，子句之间用英文逗号（,）分隔。

示例

```
nebula> CREATE EDGE IF NOT EXISTS e1(p1 string, p2 int);
nebula> ALTER EDGE e1 ADD (p3 int, p4 string);
nebula> ALTER EDGE e1 TTL_DURATION = 2, TTL_COL = "p2";
nebula> ALTER EDGE e1 COMMENT = 'edge1';
```

修改 **Edge type** 说明

尝试使用刚修改的 Edge type 可能会失败，因为修改是异步实现的。为确保数据同步，后续操作能顺利进行，请等待 2 个心跳周期（20 秒）。

如果需要修改心跳间隔，请为所有配置文件修改参数 `heartbeat_interval_secs`。

最后更新: June 20, 2023

2022 vesoft Inc.

4.10.4 SHOW EDGES

`SHOW EDGES` 语句显示当前图空间内的所有 Edge type 名称。

执行 `SHOW EDGES` 语句不需要任何权限，但是返回结果由登录的用户权限 决定。

语法

```
SHOW EDGES;
```

示例

```
nebula> SHOW EDGES;
+----------+
| Name     |
+----------+
| "follow" |
| "serve"  |
+----------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.10.5 DESCRIBE EDGE

`DESCRIBE EDGE` 显示指定 Edge type 的详细信息，例如字段名称、数据类型等。

前提条件

登录的用户必须拥有对应权限才能执行 `DESCRIBE EDGE` 语句。详情请参见内置角色权限。

语法

```
DESC[RIBE] EDGE <edge_type_name>
```

`DESCRIBE` 可以缩写为 `DESC` 。

示例

```
nebula> DESCRIBE EDGE follow;
+----------+---------+-------+---------+---------+
| Field    | Type    | Null  | Default | Comment |
+----------+---------+-------+---------+---------+
| "degree" | "int64" | "YES" |         |         |
+----------+---------+-------+---------+---------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.11 点语句

4.11.1 INSERT VERTEX

`INSERT VERTEX` 语句可以在 Nebula Graph 实例的指定图空间中插入一个或多个点。

前提条件

执行 `INSERT VERTEX` 语句需要当前登录的用户拥有指定图空间的插入点权限，否则会报错。

语法

```
INSERT VERTEX [IF NOT EXISTS] [tag_props, [tag_props] ...]
VALUES VID: ([prop_value_list])

tag_props:
  tag_name ([prop_name_list])

prop_name_list:
   [prop_name [, prop_name] ...]

prop_value_list:
   [prop_value [, prop_value] ...]
```

- `IF NOT EXISTS`：检测待插入的 VID 是否存在，只有不存在时，才会插入，如果已经存在，不会进行修改。

  Note

  - `IF NOT EXISTS` 仅检测 VID + Tag 的值是否相同，不会检测属性值。
  - `IF NOT EXISTS` 会先读取一次数据是否存在，因此对性能会有明显影响。

- `tag_name`：点关联的 Tag（点类型）。Tag 的创建，详情请参见 CREATE TAG。

  Caution

  Nebula Graph 3.1.0 中支持插入无 Tag 的点。

- `property_name`：需要设置的属性名称。
- `vid`：点 ID。在 Nebula Graph 3.1.0 中支持字符串和整数，需要在创建图空间时设置，详情请参见 CREATE SPACE。
- `property_value`：根据 `prop_name_list` 填写属性值。如果没有填写属性值，而 Tag 中对应的属性设置为 `NOT NULL`，会返回错误。详情请参见 CREATE TAG。

Caution

`INSERT VERTEX` 与 openCypher 中 `CREATE` 的语意不同：

- `INSERT VERTEX` 语意更接近于 NoSQL(key-value) 方式的 INSERT 语意，或者 SQL 中的 `UPSERT`（`UPDATE` or `INSERT`）。
- 相同 `VID` 和 `TAG` 的情况下，如果没有使用 `IF NOT EXISTS`，新写入的数据会覆盖旧数据，不存在时会新写入。
- 相同 `VID` 但不同 `TAG` 的情况下，不同 TAG 对应的记录不会相互覆盖，不存在会新写入。

参考以下示例。

2022 vesoft Inc.

示例

```
# 插入不包含 Tag 的点。
nebula> INSERT VERTEX VALUES "1":();

# 插入不包含属性的点。
nebula> CREATE TAG IF NOT EXISTS t1();
nebula> INSERT VERTEX t1() VALUES "10":();
```

```
nebula> CREATE TAG IF NOT EXISTS t2 (name string, age int);
nebula> INSERT VERTEX t2 (name, age) VALUES "11":("n1", 12);

# 创建失败，因为"a13"不是 int 类型。
nebula> INSERT VERTEX t2 (name, age) VALUES "12":("n1", "a13");

# 一次插入 2 个点。
nebula> INSERT VERTEX t2 (name, age) VALUES "13":("n3", 12), "14":("n4", 8);
```

```
nebula> CREATE TAG IF NOT EXISTS t3(p1 int);
nebula> CREATE TAG IF NOT EXISTS t4(p2 string);

# 一次插入两个 Tag 的属性到同一个点。
nebula> INSERT VERTEX  t3 (p1), t4(p2) VALUES "21": (321, "hello");
```

一个点可以多次插入属性值，以最后一次为准。

```
# 多次插入属性值。
nebula> INSERT VERTEX t2 (name, age) VALUES "11":("n2", 13);
nebula> INSERT VERTEX t2 (name, age) VALUES "11":("n3", 14);
nebula> INSERT VERTEX t2 (name, age) VALUES "11":("n4", 15);
nebula> FETCH PROP ON t2 "11" YIELD properties(vertex);
+----------------------+
| properties(VERTEX)   |
+----------------------+
| {age: 15, name: "n4"} |
+----------------------+
```

```
nebula> CREATE TAG IF NOT EXISTS t5(p1 fixed_string(5) NOT NULL, p2 int, p3 int DEFAULT NULL);
nebula> INSERT VERTEX t5(p1, p2, p3) VALUES "001":("Abe", 2, 3);

# 插入失败，因为属性 p1 不能为 NULL。
nebula> INSERT VERTEX t5(p1, p2, p3) VALUES "002":(NULL, 4, 5);
[ERROR (-1009)]: SemanticError: No schema found for `t5'

# 属性 p3 为默认值 NULL。
nebula> INSERT VERTEX t5(p1, p2) VALUES "003":("cd", 5);
nebula> FETCH PROP ON t5 "003" YIELD properties(vertex);
+---------------------------------+
| properties(VERTEX)              |
+---------------------------------+
| {p1: "cd", p2: 5, p3: __NULL__} |
+---------------------------------+

# 属性 p1 最大长度为 5，因此会被截断。
nebula> INSERT VERTEX t5(p1, p2) VALUES "004":("shalalalala", 4);
nebula> FETCH PROP on t5 "004" YIELD properties(vertex);
+-----------------------------------+
| properties(VERTEX)                |
+-----------------------------------+
| {p1: "shala", p2: 4, p3: __NULL__} |
+-----------------------------------+
```

使用 `IF NOT EXISTS` 插入已存在的点时，不会进行修改。

```
# 插入点 1。
nebula> INSERT VERTEX t2 (name, age) VALUES "1":("n2", 13);

# 使用 IF NOT EXISTS 修改点 1，因为点 1 已存在，不会进行修改。
nebula> INSERT VERTEX IF NOT EXISTS t2 (name, age) VALUES "1":("n3", 14);
nebula> FETCH PROP ON t2 "1" YIELD properties(vertex);
+----------------------+
| properties(VERTEX)   |
+----------------------+
| {age: 13, name: "n2"} |
+----------------------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.11.2 DELETE VERTEX

`DELETE VERTEX` 语句可以删除点，但是默认不删除该点关联的出边和入边。

> **Compatibility**
>
> Nebula Graph 2.x 默认删除点及关联该点的出边和入，Nebula Graph 3.1.0 默认只删除点，不删除该点关联的出边和入边，此时将默认存在悬挂边。

`DELETE VERTEX` 语句一次可以删除一个或多个点。用户可以结合管道符一起使用，详情请参见管道符。

> **Note**

- `DELETE VERTEX` 是直接删除点，不删除关联的边。
- `DELETE TAG` 是删除指定点上的指定 Tag。

语法

```
DELETE VERTEX <vid> [ , <vid> ... ] [WITH EDGE];
```

- `WITH EDGE`：删除该点关联的出边和入边。

示例

```
# 删除 VID 为 `team1` 的点，不删除该点关联的出边和入边。
nebula> DELETE VERTEX "team1";

# 删除 VID 为 `team1` 的点，并删除该点关联的出边和入边。
nebula> DELETE VERTEX "team1" WITH EDGE;
```

```
# 结合管道符，删除符合条件的点。
nebula> GO FROM "player100" OVER serve WHERE properties(edge).start_year == "2021" YIELD dst(edge) AS id | DELETE VERTEX $-.id;
```

删除过程

Nebula Graph 找到目标点并删除，该目标点的所有邻边（出边和入边）将成为悬挂边。

> **Caution**

- 不支持原子性删除，如果发生错误请重试，避免出现部分删除的情况。否则会导致悬挂边。
- 删除超级节点耗时较多，为避免删除完成前连接超时，可以调整 `nebula-graphd.conf` 中的参数 `--storage_client_timeout_ms` 延长超时时间。

悬挂边视频

- Nebula Graph 的悬挂边小科普（2 分 28 秒）

最后更新: June 20, 2023

2022 vesoft Inc.

4.11.3 UPDATE VERTEX

`UPDATE VERTEX` 语句可以修改点上 Tag 的属性值。

Nebula Graph 支持 CAS（compare and set）操作。

> **Note**
>
> 一次只能修改一个 Tag。

语法

```
UPDATE VERTEX ON <tag_name> <vid>
SET <update_prop>
[WHEN <condition>]
[YIELD <output>]
```

参数	是否必须	说明	示例
`ON <tag_name>`	是	指定点的 Tag。要修改的属性必须在这个 Tag 内。	`ON player`
`<vid>`	是	指定要修改的点 ID。	`"player100"`
`SET <update_prop>`	是	指定如何修改属性值。	`SET age = age +1`
`WHEN <condition>`	否	指定过滤条件。如果 `<condition>` 结果为 `false`，`SET` 子句不会生效。	`WHEN name == "Tim"`
`YIELD <output>`	否	指定语句的输出格式。	`YIELD name AS Name`

示例

```
// 查看点”player101“的属性。
nebula> FETCH PROP ON player "player101" YIELD properties(vertex);
+--------------------------------+
| properties(VERTEX)             |
+--------------------------------+
| {age: 36, name: "Tony Parker"} |
+--------------------------------+

// 修改属性 age 的值，并返回 name 和新的 age。
nebula> UPDATE VERTEX ON player "player101" \
        SET age = age + 2 \
        WHEN name == "Tony Parker" \
        YIELD name AS Name, age AS Age;
+---------------+-----+
| Name          | Age |
+---------------+-----+
| "Tony Parker" | 38  |
+---------------+-----+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.11.4 UPSERT VERTEX

`UPSERT VERTEX` 语句结合 `UPDATE` 和 `INSERT`，如果点存在，会修改点的属性值；如果点不存在，会插入新的点。

> **Note**
>
> `UPSERT VERTEX` 一次只能修改一个 Tag。

`UPSERT VERTEX` 性能远低于 `INSERT`，因为 `UPSERT` 是一组分片级别的读取、修改、写入操作。

> **Danger**
>
> 禁止在高并发写操作的情况下使用 `UPSERT` 语句，请使用 `UPDATE` 或 `INSERT` 代替。

语法

```
UPSERT VERTEX ON <tag> <vid>
SET <update_prop>
[WHEN <condition>]
[YIELD <output>]
```

参数	是否必须	说明	示例
`ON <tag>`	是	指定点的 Tag。要修改的属性必须在这个 Tag 内。	`ON player`
`<vid>`	是	指定要修改或插入的点 ID。	`"player100"`
`SET <update_prop>`	是	指定如何修改属性值。	`SET age = age +1`
`WHEN <condition>`	否	指定过滤条件。	`WHEN name == "Tim"`
`YIELD <output>`	否	指定语句的输出格式。	`YIELD name AS Name`

插入不存在的点

如果点不存在，无论 `WHEN` 子句的条件是否满足，都会插入点，同时执行 `SET` 子句，因此新插入的点的属性值取决于：

- `SET` 子句。
- 属性是否有默认值。

例如：

- 要插入的点包含基于 Tag `player` 的属性 `name` 和 `age`。
- `SET` 子句指定 `age=30`。

不同情况下的属性值如下表。

是否满足 `WHEN` 子句条件	属性是否有默认值	`name` 属性值	`age` 属性值
是	是	默认值	`30`
是	否	`NULL`	`30`
否	是	默认值	`30`
否	否	`NULL`	`30`

示例如下：

2022 vesoft Inc.

```
// 查看三个点是否存在，结果 “Empty set” 表示顶点不存在。
nebula> FETCH PROP ON * "player666", "player667", "player668" YIELD properties(vertex);
+--------------------+
| properties(VERTEX) |
+--------------------+
+--------------------+
Empty set

nebula> UPSERT VERTEX ON player "player666" \
        SET age = 30 \
        WHEN name == "Joe" \
        YIELD name AS Name, age AS Age;
+----------+----------+
| Name     | Age      |
+----------+----------+
| __NULL__ | 30       |
+----------+----------+

nebula> UPSERT VERTEX ON player "player666" \
        SET age = 31 \
        WHEN name == "Joe" \
        YIELD name AS Name, age AS Age;
+----------+-----+
| Name     | Age |
+----------+-----+
| __NULL__ | 30  |
+----------+-----+

nebula> UPSERT VERTEX ON player "player667" \
        SET age = 31 \
        YIELD name AS Name, age AS Age;
+----------+-----+
| Name     | Age |
+----------+-----+
| __NULL__ | 31  |
+----------+-----+

nebula> UPSERT VERTEX ON player "player668" \
        SET name = "Amber", age = age + 1 \
        YIELD name AS Name, age AS Age;
+---------+----------+
| Name    | Age      |
+---------+----------+
| "Amber" | __NULL__ |
+---------+----------+
```

上面最后一个示例中，因为 age 没有默认值，插入点时， age 默认值为 NULL ，执行 age = age + 1 后仍为 NULL 。如果 age 有默认值，则 age = age + 1 可以正常执行，例如：

```
nebula> CREATE TAG IF NOT EXISTS player_with_default(name string, age int DEFAULT 20);
Execution succeeded

nebula> UPSERT VERTEX ON player_with_default "player101" \
        SET age = age + 1 \
        YIELD name AS Name, age AS Age;

+----------+-----+
| Name     | Age |
+----------+-----+
| __NULL__ | 21  |
+----------+-----+
```

修改存在的点

如果点存在，且满足 WHEN 子句的条件，就会修改点的属性值。

```
nebula> FETCH PROP ON player "player101" YIELD properties(vertex);
+----------------------------------+
| properties(VERTEX)               |
+----------------------------------+
| {age: 36, name: "Tony Parker"} |
+----------------------------------+

nebula> UPSERT VERTEX ON player "player101" \
        SET age = age + 2 \
        WHEN name == "Tony Parker" \
        YIELD name AS Name, age AS Age;
+---------------+-----+
| Name          | Age |
+---------------+-----+
| "Tony Parker" | 38  |
+---------------+-----+
```

如果点存在，但是不满足 WHEN 子句的条件，修改不会生效。

2022 vesoft Inc.

```
nebula> FETCH PROP ON player "player101" YIELD properties(vertex);
+---------------------------------+
| properties(VERTEX)              |
+---------------------------------+
| {age: 38, name: "Tony Parker"} |
+---------------------------------+

nebula> UPSERT VERTEX ON player "player101" \
        SET age = age + 2 \
        WHEN name == "Someone else" \
        YIELD name AS Name, age AS Age;
+---------------+-----+
| Name          | Age |
+---------------+-----+
| "Tony Parker" | 38  |
+---------------+-----+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.12 边语句

4.12.1 INSERT EDGE

`INSERT EDGE` 语句可以在 Nebula Graph 实例的指定图空间中插入一条或多条边。边是有方向的，从起始点（src_vid）到目的点（dst_vid）。

`INSERT EDGE` 的执行方式为覆盖式插入。如果已有 Edge type、起点、终点、rank 都相同的边，则覆盖原边。

语法

```
INSERT EDGE [IF NOT EXISTS] <edge_type> ( <prop_name_list> ) VALUES
<src_vid> -> <dst_vid>[@<rank>] : ( <prop_value_list> )
[, <src_vid> -> <dst_vid>[@<rank>] : ( <prop_value_list> ), ...];

<prop_name_list> ::=
  [ <prop_name> [, <prop_name> ] ...]

<prop_value_list> ::=
  [ <prop_value> [, <prop_value> ] ...]
```

- `IF NOT EXISTS`：用户可以使用 `IF NOT EXISTS` 关键字检测待插入的边是否存在，只有不存在时，才会插入。

  > **Note**
  >
  > - `IF NOT EXISTS` 仅检测<边的类型、起始点、目的点和 rank>是否存在，不会检测属性值是否重合。
  > - `IF NOT EXISTS` 会先读取一次数据是否存在，因此对性能会有明显影响。

- `<edge_type>`：边关联的 Edge type，只能指定一个 Edge type。Edge type 必须提前创建，详情请参见 CREATE EDGE。
- `<prop_name_list>`：需要设置的属性名称列表。
- `src_vid`：起始点 ID，表示边的起点。
- `dst_vid`：目的点 ID，表示边的终点。
- `rank`：可选项。边的 rank 值。默认值为 `0`。

  > **openCypher 兼容性**
  >
  > openCypher 中没有 rank 的概念。

- `<prop_value_list>`：根据 `prop_name_list` 填写属性值。如果属性值和 Edge type 中的数据类型不匹配，会返回错误。如果没有填写属性值，而 Edge type 中对应的属性设置为 `NOT NULL`，也会返回错误。详情请参见 CREATE EDGE。

示例

```
# 插入不包含属性的边。
nebula> CREATE EDGE IF NOT EXISTS e1();
nebula> INSERT EDGE e1 () VALUES "10"->"11":();

# 插入 rank 为 1 的边。
nebula> INSERT EDGE e1 () VALUES "10"->"11"@1:();
```

```
nebula> CREATE EDGE IF NOT EXISTS e2 (name string, age int);
nebula> INSERT EDGE e2 (name, age) VALUES "11"->"13":("n1", 1);

# 一次插入 2 条边。
nebula> INSERT EDGE e2 (name, age) VALUES \
        "12"->"13":("n1", 1), "13"->"14":("n2", 2);
```


2022 vesoft Inc.

```
# 创建失败，因为"a13"不是 int 类型。
nebula> INSERT EDGE e2 (name, age) VALUES "11"->"13":("n1", "a13");
```

一条边可以多次插入属性值，以最后一次为准。

```
# 多次插入属性值。
nebula> INSERT EDGE e2 (name, age) VALUES "11"->"13":("n1", 12);
nebula> INSERT EDGE e2 (name, age) VALUES "11"->"13":("n1", 13);
nebula> INSERT EDGE e2 (name, age) VALUES "11"->"13":("n1", 14);
nebula> FETCH PROP ON e2 "11"->"13" YIELD edge AS e;
+------------------------------------------+
| e                                        |
+------------------------------------------+
| [:e2 "11"->"13" @0 {age: 14, name: "n1"}] |
+------------------------------------------+
```

使用 `IF NOT EXISTS` 插入已存在的边时，不会进行修改。

```
# 插入边。
nebula> INSERT EDGE e2 (name, age) VALUES "14"->"15"@1:("n1", 12);
# 使用 IF NOT EXISTS 修改边，因为边已存在，不会进行修改。
nebula> INSERT EDGE IF NOT EXISTS e2 (name, age) VALUES "14"->"15"@1:("n2", 13);
nebula> FETCH PROP ON e2 "14"->"15"@1 YIELD edge AS e;
+------------------------------------------+
| e                                        |
+------------------------------------------+
| [:e2 "14"->"15" @1 {age: 12, name: "n1"}] |
+------------------------------------------+
```

Note

- Nebula Graph 3.1.0 允许存在悬挂边（Dangling edge）。因此可以在起点或者终点存在前，先写边；此时就可以通过 `<edgetype>._src` 或 `<edgetype>._dst` 获取到（尚未写入的）点 VID（不建议这样使用）。
- 目前还不能保证操作的原子性，如果失败请重试，否则会发生部分写入。此时读取该数据的行为是未定义的。
- 并发写入同一条边会报 `edge conflict` 错误，可稍后重试。
- `边的 INSERT 速度` 大约是点的 INSERT 速度一半。原因是 INSERT 边会对应 storaged 的两个 INSERT，`INSERT 点` 对应 storaged 的一个 INSERT。

最后更新: June 20, 2023

2022 vesoft Inc.

4.12.2 DELETE EDGE

`DELETE EDGE` 语句可以删除边。一次可以删除一条或多条边。用户可以结合管道符一起使用，详情请参见管道符。

如果需要删除一个点的所有出边，请删除这个点。详情请参见 DELETE VERTEX。

语法

```
DELETE EDGE <edge_type> <src_vid> -> <dst_vid>[@<rank>] [, <src_vid> -> <dst_vid>[@<rank>] ...]
```



如果不指定 rank，则仅仅删除 rank 为 0 的边。需要删除所有的 rank，见如下示例。

示例

```
nebula> DELETE EDGE serve "player100" -> "team204"@0;
```

```
# 结合管道符，删除两点之间同类型的所有rank的边。
nebula> GO FROM "player100" OVER follow \
        WHERE dst(edge) == "team204" \
        YIELD src(edge) AS src, dst(edge) AS dst, rank(edge) AS rank \
        | DELETE EDGE follow $-.src -> $-.dst @ $-.rank;
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.12.3 UPDATE EDGE

`UPDATE EDGE` 语句可以修改边上 Edge type 的属性。

Nebula Graph 支持 CAS（compare and swap）操作。

语法

```
UPDATE EDGE ON <edge_type>
<src_vid> -> <dst_vid> [@<rank>]
SET <update_prop>
[WHEN <condition>]
[YIELD <output>]
```

参数	是否必须	说明	示例
`ON <edge_type>`	是	指定 Edge type。要修改的属性必须在这个 Edge type 内。	`ON serve`
`<src_vid>`	是	指定边的起始点 ID。	`"player100"`
`<dst_vid>`	是	指定边的目的点 ID。	`"team204"`
`<rank>`	否	指定边的 rank 值。	`10`
`SET <update_prop>`	是	指定如何修改属性值。	`SET start_year = start_year +1`
`WHEN <condition>`	否	指定过滤条件。如果 `<condition>` 结果为 `false`，`SET` 子句不会生效。	`WHEN end_year < 2010`
`YIELD <output>`	否	指定语句的输出格式。	`YIELD start_year AS Start_Year`

示例

```
// 用 GO 语句查看边的属性值。
nebula> GO FROM "player100" \
        OVER serve \
        YIELD properties(edge).start_year, properties(edge).end_year;
+------------------+----------------+
| serve.start_year | serve.end_year |
+------------------+----------------+
| 1997             | 2016           |
+------------------+----------------+

// 修改属性 start_year 的值，并返回 end_year 和新的 start_year。

nebula> UPDATE EDGE ON serve "player100" -> "team204"@0 \
        SET start_year = start_year + 1 \
        WHEN end_year > 2010 \
        YIELD start_year, end_year;
+------------+----------+
| start_year | end_year |
+------------+----------+
| 1998       | 2016     |
+------------+----------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.12.4 UPSERT EDGE

`UPSERT EDGE` 语句结合 `UPDATE` 和 `INSERT`，如果边存在，会更新边的属性；如果边不存在，会插入新的边。

`UPSERT EDGE` 性能远低于 `INSERT`，因为 `UPSERT` 是一组分片级别的读取、修改、写入操作。

Danger

禁止在高并发写操作的情况下使用 `UPSERT` 语句，请使用 `UPDATE` 或 `INSERT` 代替。

语法

```
UPSERT EDGE ON <edge_type>
<src_vid> -> <dst_vid> [@rank]
SET <update_prop>
[WHEN <condition>]
[YIELD <properties>]
```

参数	是否必须	说明	示例
`ON <edge_type>`	是	指定 Edge type。要修改的属性必须在这个 Edge type 内。	`ON serve`
`<src_vid>`	是	指定边的起始点 ID。	`"player100"`
`<dst_vid>`	是	指定边的目的点 ID。	`"team204"`
`<rank>`	否	指定边的 rank 值。	`10`
`SET <update_prop>`	是	指定如何修改属性值。	`SET start_year = start_year +1`
`WHEN <condition>`	否	指定过滤条件。	`WHEN end_year < 2010`
`YIELD <output>`	否	指定语句的输出格式。	`YIELD start_year AS Start_Year`

插入不存在的边

如果边不存在，无论 `WHEN` 子句的条件是否满足，都会插入边，同时执行 `SET` 子句，因此新插入的边的属性值取决于：

- `SET` 子句。
- 属性是否有默认值。

例如：

- 要插入的边包含基于 Edge type `serve` 的属性 `start_year` 和 `end_year`。
- `SET` 子句指定 `end_year = 2021`。

不同情况下的属性值如下表。

是否满足 `WHEN` 子句条件	属性是否有默认值	`start_year` 属性值	`end_year` 属性值
是	是	默认值	`2021`
是	否	`NULL`	`2021`
否	是	默认值	`2021`
否	否	`NULL`	`2021`

2022 vesoft Inc.

示例如下：

```
// 查看如下三个点是否有 serve 类型的出边，结果 “Empty set” 表示没有 serve 类型的出边。
nebula> GO FROM "player666", "player667", "player668" \
        OVER serve \
        YIELD properties(edge).start_year, properties(edge).end_year;
+-----------------------------+---------------------------+
| properties(EDGE).start_year | properties(EDGE).end_year |
+-----------------------------+---------------------------+
+-----------------------------+---------------------------+
Empty set

nebula> UPSERT EDGE on serve \
        "player666" -> "team200"@0 \
        SET end_year = 2021 \
        WHEN end_year == 2010 \
        YIELD start_year, end_year;
+------------+----------+
| start_year | end_year |
+------------+----------+
| __NULL__   | 2021     |
+------------+----------+

nebula> UPSERT EDGE on serve \
        "player666" -> "team200"@0 \
        SET end_year = 2022 \
        WHEN end_year == 2010 \
        YIELD start_year, end_year;
+------------+----------+
| start_year | end_year |
+------------+----------+
| __NULL__   | 2021     |
+------------+----------+

nebula> UPSERT EDGE on serve \
        "player667" -> "team200"@0 \
        SET end_year = 2022 \
        YIELD start_year, end_year;
+------------+----------+
| start_year | end_year |
+------------+----------+
| __NULL__   | 2022     |
+------------+----------+

nebula> UPSERT EDGE on serve \
        "player668" -> "team200"@0 \
        SET start_year = 2000, end_year = end_year + 1 \
        YIELD start_year, end_year;
+------------+----------+
| start_year | end_year |
+------------+----------+
| 2000       | __NULL__ |
+------------+----------+
```

上面最后一个示例中，因为 `end_year` 没有默认值，插入边时， `end_year` 默认值为 `NULL` ，执行 `end_year = end_year + 1` 后仍为 `NULL` 。如果 `end_year` 有默认值，则 `end_year = end_year + 1` 可以正常执行，例如：

```
nebula> CREATE EDGE IF NOT EXISTS serve_with_default(start_year int, end_year int DEFAULT 2010);
Execution succeeded

nebula> UPSERT EDGE on serve_with_default \
        "player668" -> "team200" \
        SET end_year = end_year + 1 \
        YIELD start_year, end_year;
+------------+----------+
| start_year | end_year |
+------------+----------+
| __NULL__   | 2011     |
+------------+----------+
```

修改存在的边

如果边存在，且满足 `WHEN` 子句的条件，就会修改边的属性值。

```
nebula> MATCH (v:player{name:"Ben Simmons"})-[e:serve]-(v2) \
        RETURN e;
+------------------------------------------------------------------------+
| e                                                                      |
+------------------------------------------------------------------------+
| [:serve "player149"->"team219" @0 {end_year: 2019, start_year: 2016}] |
+------------------------------------------------------------------------+

nebula> UPSERT EDGE on serve \
        "player149" -> "team219" \
        SET end_year = end_year + 1 \
        WHEN start_year == 2016 \
        YIELD start_year, end_year;
```


2022 vesoft Inc.

```
+------------+----------+
| start_year | end_year |
+------------+----------+
| 2016       | 2020     |
+------------+----------+
```

如果边存在，但是不满足 `WHEN` 子句的条件，修改不会生效。

```
nebula> MATCH (v:player{name:"Ben Simmons"})-[e:serve]-(v2) \
        RETURN e;
+------------------------------------------------------------------------+
| e                                                                      |
+------------------------------------------------------------------------+
| [:serve "player149"->"team219" @0 {end_year: 2020, start_year: 2016}] |
+------------------------------------------------------------------------+

nebula> UPSERT EDGE on serve \
        "player149" -> "team219" \
        SET end_year = end_year + 1 \
        WHEN start_year != 2016 \
        YIELD start_year, end_year;
+------------+----------+
| start_year | end_year |
+------------+----------+
| 2016       | 2020     |
+------------+----------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.13 原生索引

4.13.1 索引介绍

Nebula Graph 支持两种类型索引：原生索引和全文索引。

和一般数据库意义上的索引概念不同，Nebula Graph中的索引没有加速查询的功能，是用于定位到数据的必要前置条件。

原生索引

原生索引可以基于指定的属性查询数据，有如下特点：

- 包括 Tag 索引和 Edge type 索引。
- 必须手动重建索引（`REBUILD INDEX`）。
- 支持创建同一个 Tag 或 Edge type 的多个属性的索引（复合索引），但是不能跨 Tag 或 Edge type。

原生索引操作

- CREATE INDEX
- SHOW CREATE INDEX
- SHOW INDEXES
- DESCRIBE INDEX
- REBUILD INDEX
- SHOW INDEX STATUS
- DROP INDEX
- LOOKUP
- MATCH

全文索引

全文索引是基于Elastic Search来实现的，用于对字符串属性进行前缀搜索、通配符搜索、正则表达式搜索和模糊搜索，有如下特点：

- 只允许创建一个属性的索引。
- 只能创建指定长度（不超过 256 字节）字符串的索引。
- 不支持逻辑操作，例如`AND`、`OR`、`NOT`。

> **Note**
>
> 如果需要进行整个字符串的匹配，请使用原生索引。

全文索引操作

在对全文索引执行任何操作之前，请确保已经部署全文索引。详情请参见部署全文索引 和部署 listener。

部署完成后，Elasticsearch 集群上会自动创建全文索引。不支持重建或修改全文索引。如果需要删除全文索引，请在 Elasticsearch 集群上手动删除。

使用全文索引请参见使用全文索引查询。

2022 vesoft Inc.

没有 **NULL** 值索引

不支持对值为 NULL 的属性创建索引。

没有唯一索引

在 Cypher 中，可以通过 Constrains 实现属性值的唯一性限制。在 MySQL 中，可以建立唯一索引来限制某字段只有唯一值。在 nGQL 中没有属性的唯一索引（用户自行保证属性值的唯一性）。

数字、日期和时间类型的范围查询

原生索引还支持对数字、日期和时间类型的属性进行范围查询，不支持其他属性类型的范围查询。

最后更新: June 20, 2023

2022 vesoft Inc.

4.13.2 CREATE INDEX

前提条件

创建索引之前，请确保相关的 Tag 或 Edge type 已经创建。如何创建 Tag 和 Edge type，请参见 CREATE TAG 和 CREATE EDGE。

如何创建全文索引，请参见部署全文索引。

使用索引必读

索引的概念和使用限制都较为复杂。索引配合 `LOOKUP` 和 `MATCH` 语句使用。

`CREATE INDEX` 语句用于对 Tag、EdgeType 或其属性创建原生索引。通常分别称为“Tag 索引”、“Edge type 索引”和“属性索引”。

- Tag 索引和 Edge type 索引应用于和 Tag、Edge type 自身相关的查询，例如用 `LOOKUP` 查找有 Tag `player` 的所有点。
- “属性索引”应用于基于属性的查询，例如基于属性 `age` 找到 `age == 19` 的所有的点。

如果已经为 Tag `T` 的属性 `A` 建立过属性索引 `i_TA`，索引之间的可替代关系如下（Edge type 索引同理）：

- 查询引擎可以使用 `i_TA` 来替代 `i_T`。
- 在 `MATCH` 语句中 `i_T` 不能替代 `i_TA` 用于属性查找。
- 在 `LOOKUP` 语句中 `i_T` 可以替代 `i_TA` 用于属性查找。

> 历史版本兼容性
>
> 在此前的版本中，`LOOKUP` 语句中的 Tag 或 Edge type 索引不可替代属性索引用于属性查找。

使用替代索引进行查询虽然能获得相同的结果，但查询性能会根据选择的索引有所区别。

> **Caution**
>
> 不要任意在生产环境中使用索引，除非很清楚使用索引对业务的影响。索引会导致写性能下降 90%甚至更多。
>
> 索引并不用于查询加速。只用于：根据属性定位到点或边，或者统计点边数量。
>
> 长索引会降低 Storage 服务的扫描性能，以及占用更多内存。建议将索引长度设置为和要被索引的最长字符串相同。索引长度最长为 256。

如果必须使用索引，通常按照如下步骤：

1. 初次导入数据至 Nebula Graph。
2. 创建索引。
3. 重建索引。
4. 使用 LOOKUP 或 MATCH 语句查询数据。不需要（也无法）指定使用哪个索引，Nebula Graph 会自动计算。

> **Note**
>
> 如果先创建索引再导入数据，会因为写性能的下降导致导入速度极慢。
>
> 日常增量写入时保持 `--disable_auto_compaction = false`。
>
> 新创建的索引并不会立刻生效。创建新的索引并尝试立刻使用（例如 `LOOKUP` 或者 `REBUILD INDEX`）通常会失败（报错 `can't find xxx in the space`）。因为创建步骤是异步实现的，Nebula Graph 要在下一个心跳周期才能完成索引的创建。为确保数据同步，后续操作能顺利进行，请等待 2 个心跳周期（20 秒）。如果需要修改心跳间隔，请为所有配置文件修改参数 `heartbeat_interval_secs`。

2022 vesoft Inc.

Danger

创建索引，或者删除并再次创建同名索引后，必须 `REBUILD INDEX` 。否则无法在 `MATCH` 和 `LOOKUP` 语句中返回这些数据。

语法

```
CREATE {TAG | EDGE} INDEX [IF NOT EXISTS] <index_name> ON {<tag_name> | <edge_name>} ([<prop_name_list>]) [COMMENT '<comment>'];
```

参数	说明
`TAG | EDGE`	指定要创建的索引类型。
`IF NOT EXISTS`	检测待创建的索引是否存在，只有不存在时，才会创建索引。
`<index_name>`	索引名。索引名在一个图空间中必须是唯一的。推荐的命名方式为 `i_tagName_propName` 。索引名称以英文字母开头，支持 1~4 字节的 UTF-8 编码字符，包括英文字母（区分大小写）、数字、中文等，但是不包括除下划线外的特殊字符。使用特殊字符或保留关键字时，需要用反引号（`）包围，详情参见关键字和保留字。
`<tag_name> | <edge_name>`	指定索引关联的 Tag 或 Edge 名称。
`<prop_name_list>`	为变长字符串属性创建索引时，必须用 `prop_name(length)` 指定索引长度；为 Tag 或 Edge type 本身创建索引时，忽略 `<prop_name_list>` 。
`COMMENT`	索引的描述。最大为 256 字节。默认无描述。

创建 **Tag/Edge type** 索引

```
nebula> CREATE TAG INDEX player_index on player();
```

```
nebula> CREATE EDGE INDEX follow_index on follow();
```

为 Tag 或 Edge type 创建索引后，用户可以使用 `LOOKUP` 语句查找 `带有该 Tag 的` 所有点的 VID，或者 `所有该类型的边` 的 `对应起始点 VID、目的点 VID、以及 rank` 。详情请参见 LOOKUP。

创建单属性索引

```
nebula> CREATE TAG INDEX IF NOT EXISTS player_index_0 on player(name(10));
```

上述示例是为所有包含 Tag `player` 的点创建属性 `name` 的索引，索引长度为 10。即只使用属性 `name` 的前 10 个字符来创建索引。

```
# 变长字符串需要指定索引长度。
nebula> CREATE TAG IF NOT EXISTS var_string(p1 string);
nebula> CREATE TAG INDEX IF NOT EXISTS var ON var_string(p1(10));

# 定长字符串不需要指定索引长度。
nebula> CREATE TAG IF NOT EXISTS fix_string(p1 FIXED_STRING(10));
nebula> CREATE TAG INDEX IF NOT EXISTS fix ON fix_string(p1);
```

```
nebula> CREATE EDGE INDEX IF NOT EXISTS follow_index_0 on follow(degree);
```

创建复合属性索引

`复合属性索引` 用于查找一个 Tag（或者 Edge type) 中的多个属性（的组合）。

```
nebula> CREATE TAG INDEX IF NOT EXISTS player_index_1 on player(name(10), age);
```

Caution

不支持跨 Tag 或 Edge type 创建复合索引。

2022 vesoft Inc.

Note

使用复合属性索引时，遵循"最左匹配原则"，必须从复合属性索引的最左侧开始匹配。

最后更新: June 20, 2023

2022 vesoft Inc.

4.13.3 SHOW INDEXES

`SHOW INDEXES` 语句可以列出当前图空间内的所有 Tag 和 Edge type（包括属性）的索引。

语法

```
SHOW {TAG | EDGE} INDEXES;
```

示例

```
nebula> SHOW TAG INDEXES;
+-----------------+-------------+-----------------+
| Index Name      | By Tag      | Columns         |
+-----------------+-------------+-----------------+
| "fix"           | "fix_string" | ["p1"]          |
| "player_index_0" | "player"    | ["name"]        |
| "player_index_1" | "player"    | ["name", "age"] |
| "var"           | "var_string" | ["p1"]          |
+-----------------+-------------+-----------------+

nebula> SHOW EDGE INDEXES;
+---------------+----------+---------+
| Index Name    | By Edge  | Columns |
+---------------+----------+---------+
| "follow_index" | "follow" | []      |
+---------------+----------+---------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.13.4 SHOW CREATE INDEX

`SHOW CREATE INDEX` 展示创建 Tag 或者 Edge type 时使用的 nGQL 语句，其中包含索引的详细信息，例如其关联的属性。

语法

```
SHOW CREATE {TAG | EDGE} INDEX <index_name>;
```

示例

用户可以先运行 `SHOW TAG INDEXES` 查看有哪些 Tag 索引，然后用 `SHOW CREATE TAG INDEX` 查看指定索引的创建信息。

```
nebula> SHOW TAG INDEXES;
+------------------+----------+----------+
| Index Name       | By Tag   | Columns  |
+------------------+----------+----------+
| "player_index_0" | "player" | []       |
| "player_index_1" | "player" | ["name"] |
+------------------+----------+----------+

nebula> SHOW CREATE TAG INDEX player_index_1;
+------------------+-------------------------------------------------+
| Tag Index Name   | Create Tag Index                                |
+------------------+-------------------------------------------------+
| "player_index_1" | "CREATE TAG INDEX `player_index_1` ON `player` ( |
|                  |  `name`(20)                                     |
|                  | )"                                              |
+------------------+-------------------------------------------------+
```

Edge type 索引可以用类似的方法查询:

```
nebula> SHOW EDGE INDEXES;
+----------------+----------+---------+
| Index Name     | By Edge  | Columns |
+----------------+----------+---------+
| "follow_index" | "follow" | []      |
+----------------+----------+---------+

nebula> SHOW CREATE EDGE INDEX follow_index;
+-----------------+--------------------------------------------------+
| Edge Index Name | Create Edge Index                                |
+-----------------+--------------------------------------------------+
| "follow_index"  | "CREATE EDGE INDEX `follow_index` ON `follow` ( |
|                 | )"                                               |
+-----------------+--------------------------------------------------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.13.5 DESCRIBE INDEX

`DESCRIBE INDEX` 语句可以查看指定索引的信息，包括索引的属性名称（Field）和数据类型（Type）。

语法

```
DESCRIBE {TAG | EDGE} INDEX <index_name>;
```

示例

```
nebula> DESCRIBE TAG INDEX player_index_0;
+--------+--------------------+
| Field  | Type               |
+--------+--------------------+
| "name" | "fixed_string(30)" |
+--------+--------------------+

nebula> DESCRIBE TAG INDEX player_index_1;
+--------+--------------------+
| Field  | Type               |
+--------+--------------------+
| "name" | "fixed_string(10)" |
| "age"  | "int64"            |
+--------+--------------------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.13.6 REBUILD INDEX

Danger

- 索引功能不会自动对其创建之前已存在的存量数据生效————在索引重建完成之前，无法基于该索引使用 `LOOKUP` 和 `MATCH` 语句查询到存量数据。
- 索引的重建未完成时，依赖索引的查询仅能使用部分索引，因此不能获得准确结果。

请在创建索引后，选择合适的时间为存量数据重建索引。使用索引的详情请参见 CREATE INDEX。

语法

```
REBUILD {TAG | EDGE} INDEX [<index_name_list>];

<index_name_list>::=
    [index_name [, index_name] ...]
```

- 可以一次重建多个索引，索引名称之间用英文逗号（,）分隔。如果没有指定索引名称，将会重建所有索引。
- 重建完成后，用户可以使用命令 `SHOW {TAG | EDGE} INDEX STATUS` 检查索引是否重建完成。详情请参见 SHOW INDEX STATUS。

示例

```
nebula> CREATE TAG IF NOT EXISTS person(name string, age int, gender string, email string);
nebula> CREATE TAG INDEX IF NOT EXISTS single_person_index ON person(name(10));

# 重建索引，返回任务 ID。
nebula> REBUILD TAG INDEX single_person_index;
+------------+
| New Job Id |
+------------+
| 31         |
+------------+

# 查看索引状态。
nebula> SHOW TAG INDEX STATUS;
+------------------------+--------------+
| Name                   | Index Status |
+------------------------+--------------+
| "single_person_index" | "FINISHED"   |
+------------------------+--------------+

# 也可以使用 SHOW JOB <job_id>查看重建索引的任务状态。
nebula> SHOW JOB 31;
+---------------+--------------------+------------+-------------------------+-------------------------+-------------+
| Job Id(TaskId) | Command(Dest)       | Status     | Start Time              | Stop Time               | Error Code  |
+---------------+--------------------+------------+-------------------------+-------------------------+-------------+
| 31             | "REBUILD_TAG_INDEX" | "FINISHED" | 2021-07-07T09:04:24.000 | 2021-07-07T09:04:24.000 | "SUCCEEDED" |
| 0              | "storaged1"         | "FINISHED" | 2021-07-07T09:04:24.000 | 2021-07-07T09:04:28.000 | "SUCCEEDED" |
| 1              | "storaged2"         | "FINISHED" | 2021-07-07T09:04:24.000 | 2021-07-07T09:04:28.000 | "SUCCEEDED" |
| 2              | "storaged0"         | "FINISHED" | 2021-07-07T09:04:24.000 | 2021-07-07T09:04:28.000 | "SUCCEEDED" |
+---------------+--------------------+------------+-------------------------+-------------------------+-------------+
```

Nebula Graph 创建一个任务去重建索引，因此可以根据返回的任务 ID，通过 `SHOW JOB <job_id>` 语句查看任务状态。详情请参见 SHOW JOB。

最后更新: June 20, 2023

2022 vesoft Inc.

4.13.7 SHOW INDEX STATUS

`SHOW INDEX STATUS` 语句可以查看索引名称和对应作业的状态。

索引状态包括:

- `QUEUE`：队列中
- `RUNNING`：执行中
- `FINISHED`：已完成
- `FAILED`：失败
- `STOPPED`：停止
- `INVALID`：失效

> **Note**
>
> 如何创建索引请参见 CREATE INDEX。

语法

```
SHOW {TAG | EDGE} INDEX STATUS;
```

示例

```
nebula> SHOW TAG INDEX STATUS;
+----------------------+--------------+
| Name                 | Index Status |
+----------------------+--------------+
| "player_index_0"     | "FINISHED"   |
| "player_index_1"     | "FINISHED"   |
+----------------------+--------------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.13.8 DROP INDEX

`DROP INDEX` 语句可以删除当前图空间中已存在的索引。

前提条件

执行 `DROP INDEX` 语句需要当前登录的用户拥有指定图空间的 `DROP TAG INDEX` 和 `DROP EDGE INDEX` 权限，否则会报错。

语法

```
DROP {TAG | EDGE} INDEX [IF EXISTS] <index_name>;
```

`IF NOT EXISTS`：检测待删除的索引是否存在，只有存在时，才会删除索引。

示例

```
nebula> DROP TAG INDEX player_index_0;
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.14 全文索引

4.14.1 全文索引限制

Caution

本文介绍全文索引的限制，请在使用全文索引前仔细阅读。

全文索引有如下 16 条限制：

- 全文索引当前仅支持 `LOOKUP` 语句。
- 定长字符串长度超过 256 字节，将无法创建全文索引。
- 如果 Tag/Edge type 上存在全文索引，无法删除或修改 Tag/Edge type。
- 一个 Tag/Edge type 只能有一个全文索引。
- 属性的类型必须为 `String` 。
- 全文索引不支持多个 Tag/Edge type 的搜索。
- 不支持排序全文搜索的返回结果，而是按照数据插入的顺序返回。
- 全文索引不支持搜索属性值为 `NULL` 的属性。
- 不支持修改 Elastic Search 中的索引，只能删除重建。
- 不支持管道符。
- `WHERE` 子句只能用单个条件进行全文搜索。
- 全文索引不会与图空间一起删除。
- 确保同时启动了 Elasticsearch 集群和 Nebula Graph，否则可能导致 Elasticsearch 集群写入的数据不完整。
- 在点或边的属性值中不要包含 `'` 或 `\`，否则会导致 Elasticsearch 集群存储时报错。
- 从写入 Nebula Graph，到写入 listener，再到写入 Elasticsearch 并创建索引可能需要一段时间。如果访问全文索引时返回未找到索引，可等待索引生效（但是，该等待时间未知，也无返回码检查）。
- 使用 K8s 方式部署的 Nebula Graph 集群不支持全文索引。

最后更新: June 20, 2023

2022 vesoft Inc.

4.14.2 部署全文索引

Nebula Graph 的全文索引是基于 Elasticsearch 实现，这意味着用户可以使用 Elasticsearch 全文查询语言来检索想要的内容。全文索引由内置的进程管理，当 listener 集群和 Elasticsearch 集群部署后，内置的进程只能为数据类型为定长字符串或变长字符串的属性创建全文索引。

注意事项

使用全文索引前，请确认已经了解全文索引的使用限制。

部署 **Elasticsearch** 集群

部署 Elasticsearch 集群请参见 Kubernetes 安装 Elasticsearch 或单机安装 Elasticsearch。

当 Elasticsearch 集群启动时，请添加 Nebula Graph 全文索引的模板文件。关于索引模板的说明请参见 Elasticsearch 官方文档。

以下面的模板为例（针对ES 7.0及以上版本）：

```
{
 "template": "nebula*",
  "settings": {
    "index": {
      "number_of_shards": 3,
      "number_of_replicas": 1
    }
  },
  "mappings": {
    "properties" : {
            "tag_id" : { "type" : "long" },
            "column_id" : { "type" : "text" },
            "value" :{ "type" : "keyword"}
        }
  }
}
```

请确保指定的以下字段严格符合上述模板格式:

```
"template": "nebula*"
"tag_id" : { "type" : "long" },
"column_id" : { "type" : "text" },
"value" :{ "type" : "keyword"}
```

Caution

创建全文索引时，索引名称需要以 `nebula` 开头。

示例命令:

```
curl -H "Content-Type: application/json; charset=utf-8" -XPUT http://127.0.0.1:9200/_template/nebula_index_template -d '
{
 "template": "nebula*",
  "settings": {
    "index": {
      "number_of_shards": 3,
      "number_of_replicas": 1
    }
  },
  "mappings": {
    "properties" : {
            "tag_id" : { "type" : "long" },
            "column_id" : { "type" : "text" },
            "value" :{ "type" : "keyword"}
        }
  }
}'
```

用户可以配置 Elasticsearch 来满足业务需求，如果需要定制 Elasticsearch，请参见 Elasticsearch 官方文档。

2022 vesoft Inc.

登录文本搜索客户端

部署 Elasticsearch 集群之后，可以使用 `SIGN IN` 语句登录 Elasticsearch 客户端。必须使用 Elasticsearch 配置文件中的 IP 地址和端口才能正常连接，同时登录多个客户端，请在多个 `elastic_ip:port` 之间用英文逗号（,）分隔。

语法

```
SIGN IN TEXT SERVICE (<elastic_ip:port>, {HTTP | HTTPS} [,"<username>", "<password>"]) [, (<elastic_ip:port>, ...)];
```

示例

```
nebula> SIGN IN TEXT SERVICE (127.0.0.1:9200, HTTP);
```

> **Note**
>
> Elasticsearch 默认没有用户名和密码，如果设置了用户名和密码，请在 `SIGN IN` 语句中指定。

显示文本搜索客户端

`SHOW TEXT SEARCH CLIENTS` 语句可以列出文本搜索客户端。

语法

```
SHOW TEXT SEARCH CLIENTS;
```

示例

```
nebula> SHOW TEXT SEARCH CLIENTS;
+-------------+------+
| Host        | Port |
+-------------+------+
| "127.0.0.1" | 9200 |
+-------------+------+
```

退出文本搜索客户端

`SIGN OUT TEXT SERVICE` 语句可以退出所有文本搜索客户端。

语法

```
SIGN OUT TEXT SERVICE;
```

示例

```
nebula> SIGN OUT TEXT SERVICE;
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.14.3 部署 Raft listener

全文索引的数据是异步写入 Elasticsearch 集群的。流程是通过 Storage 服务的 Raft listener（简称 listener）这个单独部署的进程，从 Storage 服务读取数据，然后将它们写入 Elasticsearch 集群。

前提条件

- 已经了解全文索引的使用限制。
- 已经部署 Nebula Graph 集群。
- 完成部署 Elasticsearch 集群。
- 准备一台或者多台额外的服务器，来部署 Raft listener。

注意事项

- 请保证 Nebula 各组件（Metad、Storaged、Graphd、listener）有相同的版本。
- 只能为一个图空间“一次性添加所有的 listener 机器”。尝试向已经存在有 listener 的图空间再添加新 listener 会失败。因此，需在一个命令语句里完整地添加全部的 listener。

部署流程

第一步：安装 **STORAGE** 服务

listener 进程与 storaged 进程使用相同的二进制文件，但是二者配置文件不同，进程使用端口也不同，可以在所有需要部署 listener 的服务器上都安装 Nebula Graph，但是仅使用 Storage 服务。详情请参见使用 RPM 或 DEB 安装包安装 Nebula Graph。

第二步：准备 **LISTENER** 的配置文件

用户必须在需要部署 listener 的机器上准备对应的配置文件，文件名称必须为 `nebula-storaged-listener.conf`，并保存在安装路径下的 `etc` 目录内。用户可以参考提供的模板。注意去掉文件后缀 `.production`。

大部分配置与 Storage 服务的配置文件相同，本文仅介绍差异部分。

名称	预设值	说明
`daemonize`	`true`	是否启动守护进程。
`pid_file`	`pids_listener/nebula-storaged.pid`	记录进程 ID 的文件。
`meta_server_addrs`	-	全部 Meta 服务的 IP 地址和端口。多个 Meta 服务用英文逗号（,）分隔。
`local_ip`	-	listener 服务的本地 IP 地址。
`port`	-	listener 服务的 RPC 守护进程监听端口。
`heartbeat_interval_secs`	`10`	Meta 服务的心跳间隔。单位：秒（s）。
`listener_path`	`data/listener`	listener 的 WAL 目录。只允许使用一个目录。
`data_path`	`data`	出于兼容性考虑，可以忽略此参数。填充一个默认值 `data`。
`part_man_type`	`memory`	部件管理器类型，可选值为 `memory` 和 `meta`。
`rocksdb_batch_size`	`4096`	批处理操作的默认保留字节。
`rocksdb_block_cache`	`4`	BlockBasedTable 的默认块缓存大小。单位：兆字节（MB）。
`engine_type`	`rocksdb`	存储引擎类型，例如 `rocksdb`、`memory` 等。
`part_type`	`simple`	部件类型，例如 `simple`、`consensus` 等。

2022 vesoft Inc.

Note

在配置文件中请使用真实的（listener 机器）IP 地址替换 `127.0.0.1`。

第三步：启动 **LISTENER**

执行如下命令启动启动 listener：

```
./bin/nebula-storaged --flagfile <listener_config_path>/nebula-storaged-listener.conf
```

`listener_config_path` 是存放 listener 配置文件的路径。

第四步：添加 **LISTENER** 到 **NEBULA GRAPH** 集群

用命令行连接到 Nebula Graph，然后执行 `USE <space>` 进入需要创建全文索引的图空间。然后执行如下命令添加 listener：

```
ADD LISTENER ELASTICSEARCH <listener_ip:port> [,<listener_ip:port>, ...]
```

Warning

listener 必须使用真实的 IP 地址。

请在一个语句里完整地添加所有 listener。例如：

```
nebula> ADD LISTENER ELASTICSEARCH 192.168.8.5:9789,192.168.8.6:9789;
```

查看 listener

执行 `SHOW LISTENER` 语句可以列出所有的 listener。

示例

```
nebula> SHOW LISTENER;
+--------+-----------------+----------------------+----------+
| PartId | Type            | Host                 | Status   |
+--------+-----------------+----------------------+----------+
| 1      | "ELASTICSEARCH" | "[192.168.8.5:46780]" | "ONLINE" |
| 2      | "ELASTICSEARCH" | "[192.168.8.5:46780]" | "ONLINE" |
| 3      | "ELASTICSEARCH" | "[192.168.8.5:46780]" | "ONLINE" |
+--------+-----------------+----------------------+----------+
```

删除 listener

执行 `REMOVE LISTENER ELASTICSEARCH` 语句可以删除图空间的所有 listener。

示例

```
nebula> REMOVE LISTENER ELASTICSEARCH;
```

Danger

删除 listener 后，将不能重新添加 listener，因此也无法继续向 ES 集群同步，文本索引数据将不完整。如果确实需要，只能重新创建图空间。

下一步

部署全文索引和 listener 后，全文索引会在 Elasticsearch 集群中自动创建，用户可以开始进行全文搜索。详情请参见全文搜索。

最后更新: June 20, 2023

2022 vesoft Inc.

4.14.4 全文搜索

全文搜索是基于全文索引对值为字符串类型的属性进行前缀搜索、通配符搜索、正则表达式搜索和模糊搜索。

在 `LOOKUP` 语句中，使用 `WHERE` 子句指定字符串的搜索条件。

前提条件

请确保已经部署全文索引。详情请参见部署全文索引和部署 listener。

注意事项

使用全文索引前，请确认已经了解全文索引的使用限制。

自然语言全文搜索

自然语言搜索将搜索的字符串解释为自然人类语言中的短语。搜索不区分大小写，且默认是对字符串的每个子字符串（以空格分隔）单独判断搜索。例如，有三个点属于标签 `player`，标签 `player` 含有属性 `name`，这三个点的 `name` 分别为 `Kevin Durant` 、`Tim Duncan` 和 `David Beckham`。现在已经建立好有关 `player.name` 的全文索引，在用全文索引前缀搜索语句 `LOOKUP ON player WHERE PREFIX(player.name,"d");` 查询时，这三个点都会被查询到。

语法

创建全文索引

```
CREATE FULLTEXT {TAG | EDGE} INDEX <index_name> ON {<tag_name> | <edge_name>} ([<prop_name_list>]);
```

显示全文索引

```
SHOW FULLTEXT INDEXES;
```

重建全文索引

```
REBUILD FULLTEXT INDEX;
```

删除全文索引

```
DROP FULLTEXT INDEX <index_name>;
```

使用查询选项

```
LOOKUP ON {<tag> | <edge_type>} WHERE <expression> [YIELD <return_list>];

<expression> ::=
    PREFIX | WILDCARD | REGEXP | FUZZY

<return_list>
    <prop_name> [AS <prop_alias>] [, <prop_name> [AS <prop_alias>] ...]
```

- PREFIX(schema_name.prop_name, prefix_string, row_limit, timeout)
- WILDCARD(schema_name.prop_name, wildcard_string, row_limit, timeout)
- REGEXP(schema_name.prop_name, regexp_string, row_limit, timeout)
- FUZZY(schema_name.prop_name, fuzzy_string, fuzziness, operator, row_limit, timeout)
- `fuzziness`： 可选项。允许匹配的最大编辑距离。默认值为 `AUTO`。查看其他可选值和更多信息，请参见 Elasticsearch 官方文档。
- `operator`： 可选项。解释文本的布尔逻辑。可选值为 `OR`（默认）和 `and`。
- `row_limit`： 可选项。指定要返回的行数。默认值为 `100`。
- `timeout`： 可选项。指定超时时间。单位：毫秒（ms）。默认值为 `200`。

2022 vesoft Inc.

示例

```
//创建图空间。
nebula> CREATE SPACE IF NOT EXISTS basketballplayer (partition_num=3,replica_factor=1, vid_type=fixed_string(30));

//登录文本搜索客户端。
nebula> SIGN IN TEXT SERVICE (127.0.0.1:9200, HTTP);

//切换图空间。
nebula> USE basketballplayer;

//添加 listener 到 Nebula Graph 集群。
nebula> ADD LISTENER ELASTICSEARCH 192.168.8.5:9789;

//创建 Tag。
nebula> CREATE TAG IF NOT EXISTS player(name string, age int);

//创建原生索引。
nebula> CREATE TAG INDEX IF NOT EXISTS name ON player(name(20));

//重建原生索引。
nebula> REBUILD TAG INDEX;

//创建全文索引，索引名称需要以 nebula 开头。
nebula> CREATE FULLTEXT TAG INDEX nebula_index_1 ON player(name);

//重建全文索引。
nebula> REBUILD FULLTEXT INDEX;

//查看全文索引。
nebula> SHOW FULLTEXT INDEXES;
+-----------------+-------------+-------------+--------+
| Name            | Schema Type | Schema Name | Fields |
+-----------------+-------------+-------------+--------+
| "nebula_index_1" | "Tag"       | "player"    | "name" |
+-----------------+-------------+-------------+--------+

//插入测试数据。
nebula> INSERT VERTEX player(name, age) VALUES \
  "Russell Westbrook": ("Russell Westbrook", 30), \
  "Chris Paul": ("Chris Paul", 33),\
  "Boris Diaw": ("Boris Diaw", 36),\
  "David West": ("David West", 38),\
  "Danny Green": ("Danny Green", 31),\
  "Tim Duncan": ("Tim Duncan", 42),\
  "James Harden": ("James Harden", 29),\
  "Tony Parker": ("Tony Parker", 36),\
  "Aron Baynes": ("Aron Baynes", 32),\
  "Ben Simmons": ("Ben Simmons", 22),\
  "Blake Griffin": ("Blake Griffin", 30);

//测试查询
nebula> LOOKUP ON player WHERE PREFIX(player.name, "B") YIELD id(vertex);
+----------------+
| id(VERTEX)     |
+----------------+
| "Boris Diaw"   |
| "Ben Simmons"  |
| "Blake Griffin" |
+----------------+

nebula> LOOKUP ON player WHERE WILDCARD(player.name, "*ri*") YIELD player.name, player.age;
+----------------+-----+
| name           | age |
+----------------+-----+
| "Chris Paul"   | 33  |
| "Boris Diaw"   | 36  |
| "Blake Griffin" | 30  |
+----------------+-----+

nebula> LOOKUP ON player WHERE WILDCARD(player.name, "*ri*") | YIELD count(*);
+----------+
| count(*) |
+----------+
| 3        |
+----------+

nebula> LOOKUP ON player WHERE REGEXP(player.name, "R.*") YIELD player.name, player.age;
+--------------------+-----+
| name               | age |
+--------------------+-----+
| "Russell Westbrook" | 30  |
+--------------------+-----+

nebula> LOOKUP ON player WHERE REGEXP(player.name, ".*") YIELD id(vertex);
+--------------------+
| id(VERTEX)         |
+--------------------+
| "Danny Green"      |
| "David West"       |
| "Russell Westbrook" |
+--------------------+
...
```


2022 vesoft Inc.

```
nebula> LOOKUP ON player WHERE FUZZY(player.name, "Tim Dunncan", AUTO, OR) YIELD player.name;
+--------------+
| name         |
+--------------+
| "Tim Duncan" |
+--------------+

//删除全文索引。
nebula> DROP FULLTEXT INDEX nebula_index_1;
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.15 子图和路径

4.15.1 GET SUBGRAPH

`GET SUBGRAPH` 语句检索指定 Edge type 的起始点可以到达的点和边的信息，返回子图信息。

语法

```
GET SUBGRAPH [WITH PROP] [<step_count> STEPS] FROM {<vid>, <vid>...}
[{IN | OUT | BOTH} <edge_type>, <edge_type>...]
YIELD [VERTICES AS <vertex_alias>] [, EDGES AS <edge_alias>];
```

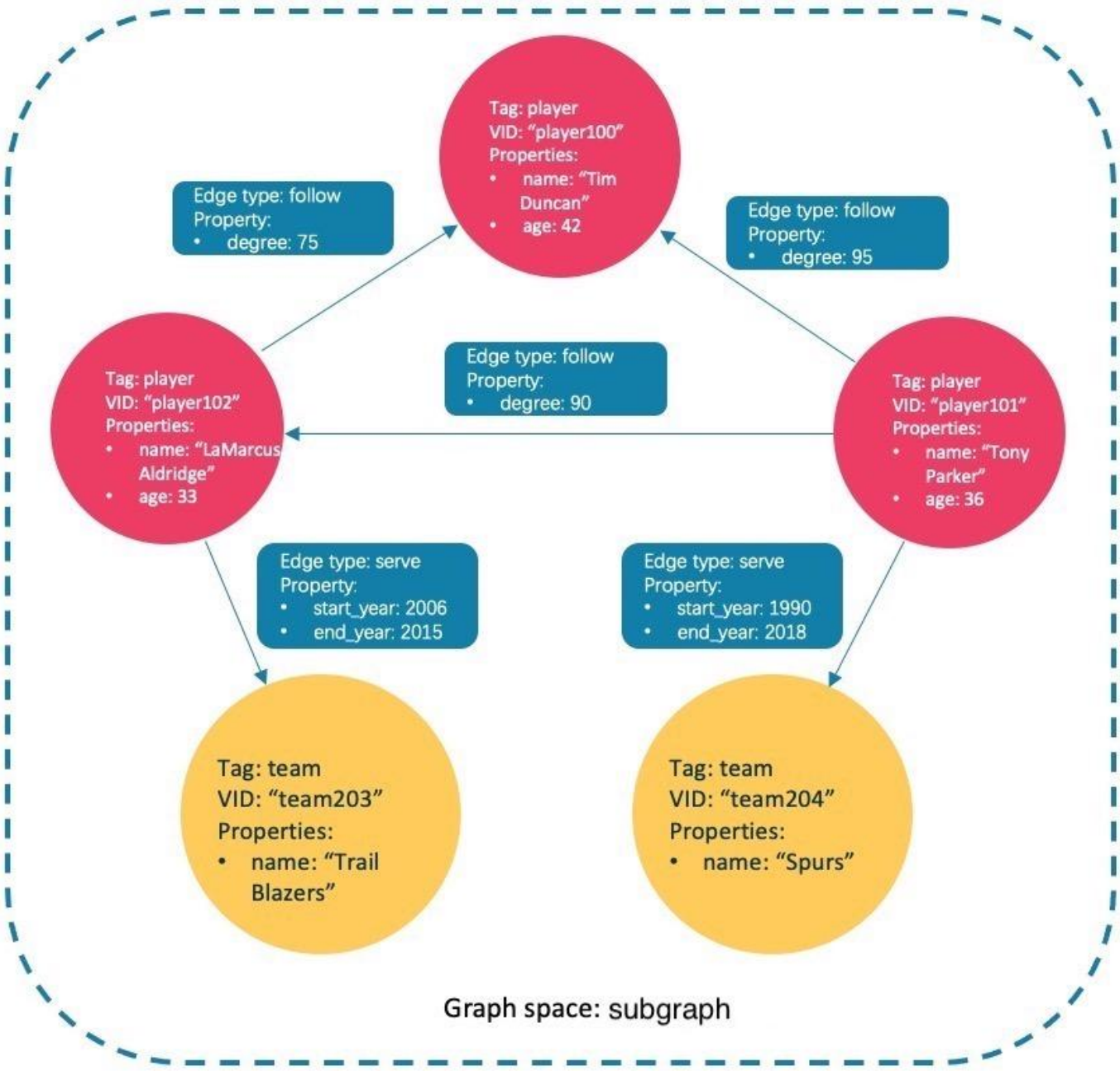
- `WITH PROP`：展示属性。不添加本参数则隐藏属性。
- `step_count`：指定从起始点开始的跳数，返回从 0 到 `step_count` 跳的子图。必须是非负整数。默认值为 1。
- `vid`：指定起始点 ID。
- `edge_type`：指定 Edge type。可以用 `IN`、`OUT` 和 `BOTH` 来指定起始点上该 Edge type 的方向。默认为 `BOTH`。
- `YIELD`：定义需要返回的输出。可以仅返回点或边。必须设置别名。

> **Note**
>
> `GET SUBGRAPH` 语句检索的路径类型为 `trail`，即检索的路径只有点可以重复，边不可以重复。详情请参见路径。

示例

以下面的示例图进行演示。

2022 vesoft Inc.



插入测试数据:

```
nebula> CREATE SPACE IF NOT EXISTS subgraph(partition_num=15, replica_factor=1, vid_type=fixed_string(30));
nebula> USE subgraph;
nebula> CREATE TAG IF NOT EXISTS player(name string, age int);
nebula> CREATE TAG IF NOT EXISTS team(name string);
nebula> CREATE EDGE IF NOT EXISTS follow(degree int);
nebula> CREATE EDGE IF NOT EXISTS serve(start_year int, end_year int);
nebula> INSERT VERTEX player(name, age) VALUES "player100":("Tim Duncan", 42);
nebula> INSERT VERTEX player(name, age) VALUES "player101":("Tony Parker", 36);
nebula> INSERT VERTEX player(name, age) VALUES "player102":("LaMarcus Aldridge", 33);
nebula> INSERT VERTEX team(name) VALUES "team203":("Trail Blazers"), "team204":("Spurs");
nebula> INSERT EDGE follow(degree) VALUES "player101" -> "player100":(95);
nebula> INSERT EDGE follow(degree) VALUES "player101" -> "player102":(90);
```

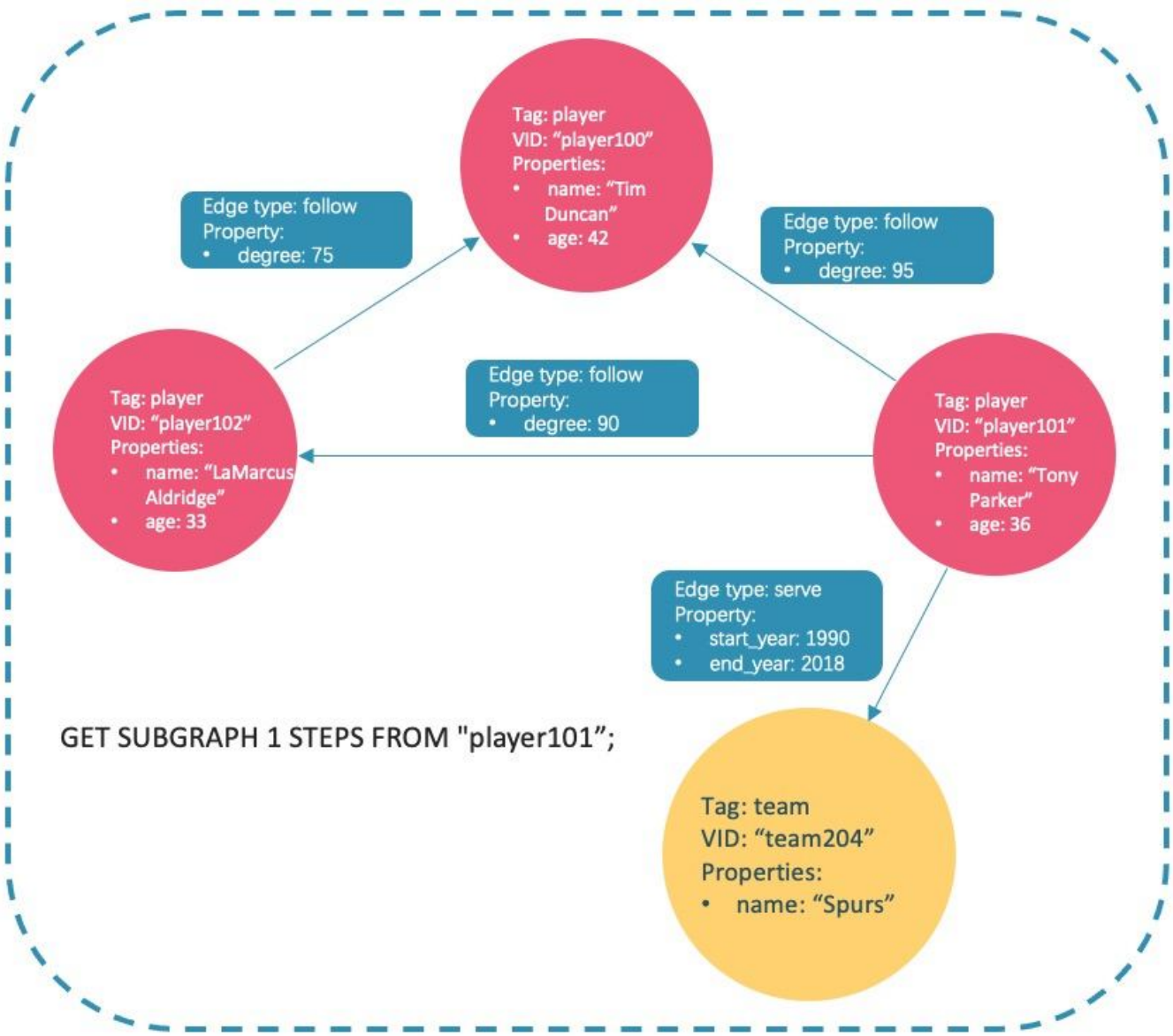

2022 vesoft Inc.

```
nebula> INSERT EDGE follow(degree) VALUES "player102" -> "player100":(75);
nebula> INSERT EDGE serve(start_year, end_year) VALUES "player101" -> "team204":(1999, 2018),"player102" -> "team203":(2006,  2015);
```

- 查询从点 `player101` 开始、0~1 跳、所有 Edge type 的子图。

```
nebula> GET SUBGRAPH 1 STEPS FROM "player101" YIELD VERTICES AS nodes, EDGES AS relationships;
+-----------------------------------------------------------------------
+-----------------------------------------------------------------------------------------------------------------------+
| nodes                                                                 |
relationships                                                                                                          |
+-----------------------------------------------------------------------
+-----------------------------------------------------------------------------------------------------------------------+
| [("player101" :player{})]                                             | [[:serve "player101"->"team204" @0 {}], [:follow "player101"->"player100" @0 {}], [:follow "player101"-
>"player102" @0 {}]] |
| [("team204" :team{}), ("player100" :player{}), ("player102" :player{})] | [[:follow "player102"->"player100" @0
{}]]                                                                                  |
+-----------------------------------------------------------------------
+-----------------------------------------------------------------------------------------------------------------------+
```

返回的子图如下。



2022 vesoft Inc.

- 查询从点 `player101` 开始、0~1 跳、 `follow` 类型的入边的子图。

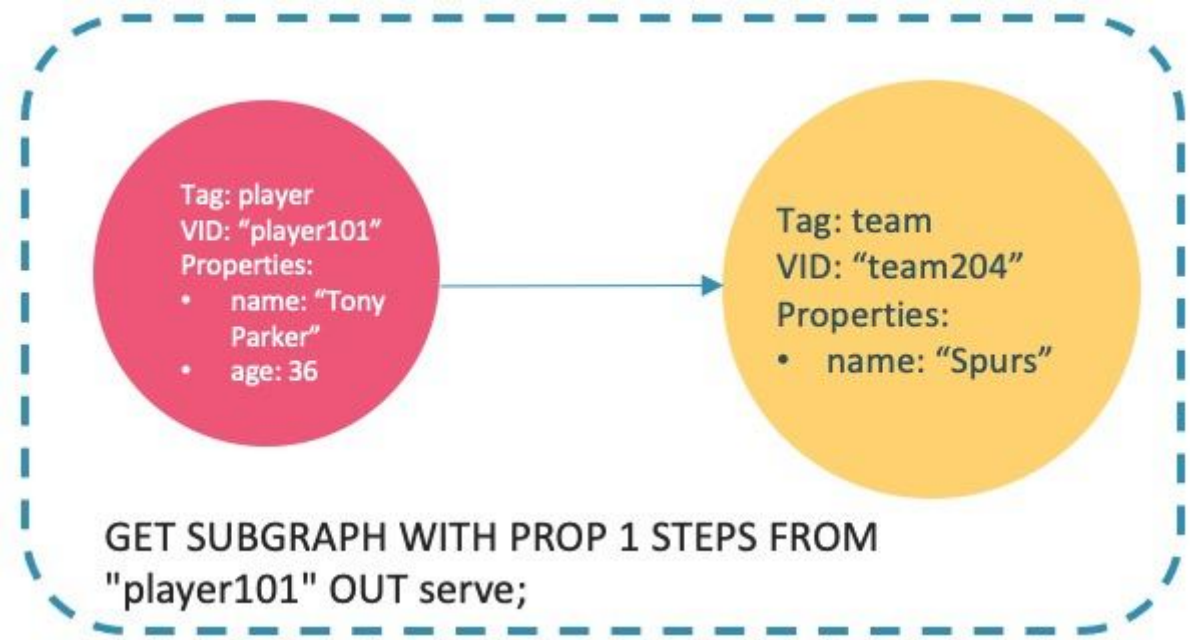
```
nebula> GET SUBGRAPH 1 STEPS FROM "player101" IN follow YIELD VERTICES AS nodes, EDGES AS relationships;
+--------------------------+---------------+
| nodes                    | relationships |
+--------------------------+---------------+
| [("player101" :player{})] | []            |
| []                       | []            |
+--------------------------+---------------+
```

因为 `player101` 没有 `follow` 类型的入边。所以仅返回点 `player101` 。

- 查询从点 `player101` 开始、0~1 跳、 `serve` 类型的出边的子图，同时展示边的属性。

```
nebula> GET SUBGRAPH WITH PROP 1 STEPS FROM "player101" OUT serve YIELD VERTICES AS nodes, EDGES AS relationships;
+-------------------------------------------------+-------------------------------------------------------------------------+
| nodes                                           | relationships                                                           |
+-------------------------------------------------+-------------------------------------------------------------------------+
| [("player101" :player{age: 36, name: "Tony Parker"})] | [[:serve "player101"->"team204" @0 {end_year: 2018, start_year: 1999}]] |
| [("team204" :team{name: "Spurs"})]              | []                                                                      |
+-------------------------------------------------+-------------------------------------------------------------------------+
```

返回的子图如下。

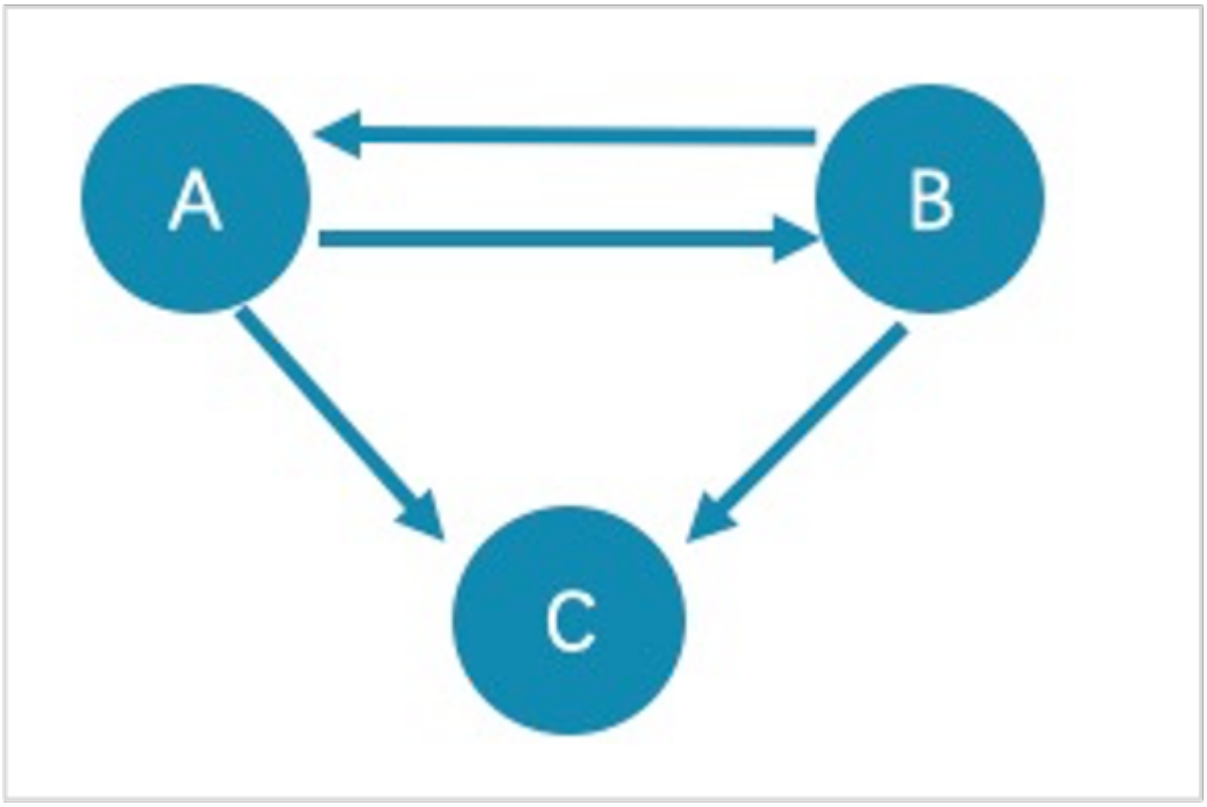


FAQ

为什么返回结果中会出现超出 **STEP_COUNT** 跳数之外的关系?

为了展示子图的完整性，会在满足条件的所有点上额外查询一跳。例如下图。

2022 vesoft Inc.



- 用 `GET SUBGRAPH 1 STEPS FROM "A";` 查询的满足结果的路径是 `A->B` 、 `B->A` 和 `A->C` ，为了子图的完整性，会在满足结果的点上额外查询一跳，即 `B->C` 。
- 用 `GET SUBGRAPH 1 STEPS FROM "A" IN follow;` 查询的满足结果的路径是 `B->A` ，在满足结果的点上额外查询一跳，即 `A->B` 。

如果只是查询满足条件的路径或点，建议使用 MATCH 或 GO 语句。例如：

```
nebula> MATCH p= (v:player) -- (v2) WHERE id(v)=="A" RETURN p;

nebula> GO 1 STEPS FROM "A" OVER follow YIELD src(edge),dst(edge);
```

为什么返回结果中会出现低于 **STEP_COUNT** 跳数的关系?

查询到没有多余子图数据时会停止查询，且不会返回空值。

```
nebula> GET SUBGRAPH 100 STEPS FROM "player101" OUT follow YIELD VERTICES AS nodes, EDGES AS relationships;
+-------------------------------------------------+-------------------------------------------------------------------------------------------+
| nodes                                           | relationships                                                                             |
+-------------------------------------------------+-------------------------------------------------------------------------------------------+
| [("player101" :player{})]                       | [[:follow "player101"->"player100" @0 {}], [:follow "player101"->"player102" @0 {}]] |
| [("player100" :player{}), ("player102" :player{})] | [[:follow "player102"->"player100" @0 {}]]                                                |
+-------------------------------------------------+-------------------------------------------------------------------------------------------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.15.2 FIND PATH

`FIND PATH` 语句查找指定起始点和目的点之间的路径。

语法

```
FIND { SHORTEST | ALL | NOLOOP } PATH [WITH PROP] FROM <vertex_id_list> TO <vertex_id_list>
OVER <edge_type_list> [REVERSELY | BIDIRECT]
[<WHERE clause>] [UPTO <N> STEPS]
YIELD path as <alias>
[| ORDER BY $-.path] [| LIMIT <M>];

<vertex_id_list> ::=
    [vertex_id [, vertex_id] ...]
```

- `SHORTEST`： 查找最短路径。
- `ALL`： 查找所有路径。
- `NOLOOP`： 查找非循环路径。
- `WITH PROP`： 展示点和边的属性。不添加本参数则隐藏属性。
- `<vertex_id_list>`： 点 ID 列表。多个点用英文逗号（,）分隔。支持 `$-` 和 `$var` 。
- `<edge_type_list>`： Edge type 列表。多个 Edge type 用英文逗号（,）分隔。`*` 表示所有 Edge type。
- `REVERSELY | BIDIRECT`： `REVERSELY` 表示反向， `BIDIRECT` 表示双向。
- `<WHERE clause>`： 可以使用 `WHERE` 子句过滤边属性。
- `<N>`： 路径的最大跳数。默认值为 `5` 。
- `<M>`： 指定返回的最大行数。

> **Note**
>
> `FIND PATH` 语句检索的路径类型为 `trail`，即检索的路径只有点可以重复，边不可以重复。详情请参见路径。

限制

- 指定起始点和目的点的列表后，会返回起始点和目的点所有组合的路径。
- 搜索所有路径时可能会出现循环。
- 使用 `WHERE` 子句时只能过滤边属性，暂不支持过滤点属性，且不支持函数。
- graphd 是单进程查询，会占用很多内存。

示例

返回的路径格式类似于 `(<vertex_id>)-[:<edge_type_name>@<rank>]->(<vertex_id)` 。

```
nebula> FIND SHORTEST PATH FROM "player102" TO "team204" OVER * YIELD path AS p;
+------------------------------------------+
| p                                        |
+------------------------------------------+
| <("player102")-[:serve@0 {}]->("team204")> |
+------------------------------------------+
```

```
nebula> FIND SHORTEST PATH WITH PROP FROM "team204" TO "player100" OVER * REVERSELY YIELD path AS p;
+-------------------------------------------------------------------------------------------------------------------------------+
| p                                                                                                                             |
+-------------------------------------------------------------------------------------------------------------------------------+
```


2022 vesoft Inc.

```
| <("team204" :team{name: "Spurs"})<-[:serve@0 {end_year: 2016, start_year: 1997}]-("player100" :player{age: 42, name: "Tim Duncan"})> |
+-------------------------------------------------------------------------------------------------------------------------------------+
```

```
nebula> FIND ALL PATH FROM "player100" TO "team204" OVER * WHERE follow.degree is EMPTY or follow.degree >=0 YIELD path AS p;
+-----------------------------------------------------------------------------+
| p                                                                           |
+-----------------------------------------------------------------------------+
| <("player100")-[:serve@0 {}]->("team204")>                                  |
| <("player100")-[:follow@0 {}]->("player125")-[:serve@0 {}]->("team204")>     |
| <("player100")-[:follow@0 {}]->("player101")-[:serve@0 {}]->("team204")>     |
| ...                                                                         |
+-----------------------------------------------------------------------------+
```

```
nebula> FIND NOLOOP PATH FROM "player100" TO "team204" OVER * YIELD path AS p;
+-------------------------------------------------------------------------------------------------------+
| p                                                                                                     |
+-------------------------------------------------------------------------------------------------------+
| <("player100")-[:serve@0 {}]->("team204")>                                                            |
| <("player100")-[:follow@0 {}]->("player125")-[:serve@0 {}]->("team204")>                               |
| <("player100")-[:follow@0 {}]->("player101")-[:serve@0 {}]->("team204")>                               |
| <("player100")-[:follow@0 {}]->("player101")-[:follow@0 {}]->("player125")-[:serve@0 {}]->("team204")> |
| <("player100")-[:follow@0 {}]->("player101")-[:follow@0 {}]->("player102")-[:serve@0 {}]->("team204")> |
| ...                                                                                                   |
+-------------------------------------------------------------------------------------------------------+
```

FAQ

是否支持 **WHERE** 子句，以实现图遍历过程中的条件过滤?

支持使用 `WHERE` 子句过滤，但只能过滤边属性，不支持过滤点属性。

如示例中的 `WHERE follow.degree is EMPTY or follow.degree >= 0` 。

最后更新: June 20, 2023

2022 vesoft Inc.

4.16 查询调优与终止

4.16.1 EXPLAIN 和 PROFILE

`EXPLAIN` 语句输出 nGQL 语句的执行计划，但不会执行 nGQL 语句。

`PROFILE` 语句执行 nGQL 语句，然后输出执行计划和执行概要。用户可以根据执行计划和执行概要优化查询性能。

执行计划

执行计划由 Nebula Graph 查询引擎中的执行计划器决定。

执行计划器将解析后的 nGQL 语句处理为 `action`。`action` 是最小的执行单元。典型的 `action` 包括获取指定点的所有邻居、获取边的属性、根据条件过滤点或边等。每个 `action` 都被分配给一个 `operator`。

例如 `SHOW TAGS` 语句分为两个 `action`，`operator` 为 `Start` 和 `ShowTags`。更复杂的 `GO` 语句可能会被处理成 10 个以上的 `action`。

语法

- `EXPLAIN`

```
EXPLAIN [format= {"row" | "dot"}] <your_nGQL_statement>;
```

- `PROFILE`

```
PROFILE [format= {"row" | "dot"}] <your_nGQL_statement>;
```

输出格式

`EXPLAIN` 或 `PROFILE` 语句的输出有两种格式：`row`（默认）和 `dot`。用户可以使用 `format` 选项修改输出格式。

2022 vesoft Inc.

row 格式

row 格式将返回信息输出到一个表格中。

- EXPLAIN

```
nebula> EXPLAIN format="row" SHOW TAGS;
Execution succeeded (time spent 327/892 us)

Execution Plan

-----+----------+--------------+---------------+-------------------------------------------------------------------
| id | name     | dependencies | profiling data | operator info                                                    |
-----+----------+--------------+---------------+-------------------------------------------------------------------
|  1 | ShowTags | 0            |               | outputVar: [{"colNames":[],"name":"__ShowTags_1","type":"DATASET"}] |
|    |          |              |               | inputVar:                                                        |
-----+----------+--------------+---------------+-------------------------------------------------------------------
|  0 | Start    |              |               | outputVar: [{"colNames":[],"name":"__Start_0","type":"DATASET"}]    |
-----+----------+--------------+---------------+-------------------------------------------------------------------
```

- PROFILE

```
nebula> PROFILE format="row" SHOW TAGS;
+--------+
| Name   |
+--------+
| player |
| team   |
+--------+
Got 2 rows (time spent 2038/2728 us)

Execution Plan

-----+----------+--------------+-------------------------------------------------+-------------------------------------------------------------------
| id | name     | dependencies | profiling data                                  | operator info                                                    |
-----+----------+--------------+-------------------------------------------------+-------------------------------------------------------------------
|  1 | ShowTags | 0            | ver: 0, rows: 1, execTime: 42us, totalTime: 1177us | outputVar: [{"colNames":[],"name":"__ShowTags_1","type":"DATASET"}] |
|    |          |              |                                                 | inputVar:                                                        |
-----+----------+--------------+-------------------------------------------------+-------------------------------------------------------------------
|  0 | Start    |              | ver: 0, rows: 0, execTime: 1us, totalTime: 57us | outputVar: [{"colNames":[],"name":"__Start_0","type":"DATASET"}]    |
-----+----------+--------------+-------------------------------------------------+-------------------------------------------------------------------
```

参数	说明
id	operator 的 ID。
name	operator 的名称。
dependencies	当前 operator 所依赖的 operator 的 ID。
profiling data	执行概要文件内容。 ver 表示 operator 的版本； rows 表示 operator 输出结果的行数； execTime 表示执行 action 的时间；totalTime 表示执行 action 的时间、系统调度时间、排队时间的总和。
operator info	operator 的详细信息。

dot 格式

dot 格式将返回 DOT 语言的信息，然后用户可以使用 Graphviz 生成计划图。

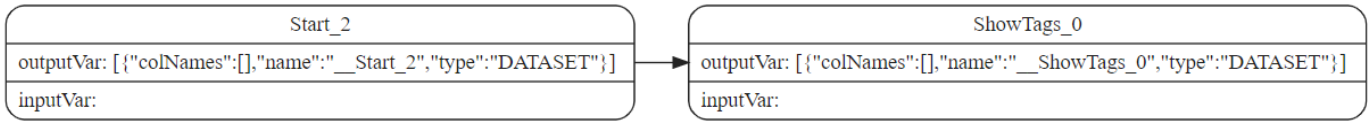
> **Note**
>
> Graphviz 是一款开源可视化图工具，可以绘制 DOT 语言脚本描述的图。Graphviz 提供一个在线工具，可以预览 DOT 语言文件，并将它们导出为 SVG 或 JSON 等其他格式。详情请参见 Graphviz Online。

```
nebula> EXPLAIN format="dot" SHOW TAGS;
Execution succeeded (time spent 161/665 us)
Execution Plan
-------------------------------------------------------------------------------------------------------------------------------------------  -------------
  plan
-------------------------------------------------------------------------------------------------------------------------------------------  -------------
  digraph exec_plan {
      rankdir=LR;
      "ShowTags_0"[label="ShowTags_0|outputVar: \[\{\"colNames\":\[\],\"name\":\"__ShowTags_0\",\"type\":\"DATASET\"\}\]\l|inputVar:\l",   shape=Mrecord];
      "Start_2"->"ShowTags_0";
      "Start_2"[label="Start_2|outputVar: \[\{\"colNames\":\[\],\"name\":\"__Start_2\",\"type\":\"DATASET\"\}\]\l|inputVar: \l",   shape=Mrecord];
```


2022 vesoft Inc.

```
}
------------------------------------------------------------------------------------------------------------------------ ------------
```

将上述示例的 DOT 语言转换为 Graphviz 图，如下所示。



最后更新: June 20, 2023

2022 vesoft Inc.

4.16.2 终止查询

`KILL QUERY` 命令可以终止正在执行的查询，常用于终止慢查询。

> **Note**
>
> 仅 God 角色权限可以终止任意查询，其他角色只能终止自己的查询。

语法

```
KILL QUERY (session=<session_id>, plan=<plan_id>);
```

- `session_id`：会话 ID。
- `plan_id`：执行计划 ID。

会话 ID 和执行计划 ID 可以唯一确定一个查询。二者可以通过 SHOW QUERIES 语句获取。

示例

在一个会话中执行命令终止另一个会话中的查询:

```
nebula> KILL QUERY(SESSION=1625553545984255,PLAN=163);
```

另一个会话中的查询会终止，并返回如下信息:

```
[ERROR (-1005)]: ExecutionPlanId[1001] does not exist in current Session.
```

最后更新: June 20, 2023

2022 vesoft Inc.

4.17 作业管理

在 Storage 服务上长期运行的任务称为作业，例如 `COMPACT` 、 `FLUSH` 和 `STATS` 。 如果图空间的数据量很大，这些作业可能耗时很长。作业管理可以帮助执行、查看、停止和恢复作业。

> **Note**
>
> 所有作业管理命令都需要先选择图空间后才能执行。

4.17.1 SUBMIT JOB BALANCE DATA

> **Enterpriseonly**
>
> 仅企业版支持本功能。

> **Caution**

- 作业执行前，建议创建快照。
- 作业执行过程中，不要执行其他作业，例如 `SUBMIT JOB STATS` 、 `REBUILD INDEX` 等。
- 作业执行过程中，不建议大批量写入和读取数据。

`SUBMIT JOB BALANCE DATA` 语句会在当前图空间内启动任务均衡分布分片。该命令会返回任务 ID。

示例：

```
nebula> SUBMIT JOB BALANCE DATA;
+------------+
| New Job Id |
+------------+
| 28         |
+------------+
```

4.17.2 SUBMIT JOB COMPACT

`SUBMIT JOB COMPACT` 语句会在当前图空间内触发 RocksDB 的长耗时 `compact` 操作。

`compact` 配置详情请参见 Storage 服务配置。

示例：

```
nebula> SUBMIT JOB COMPACT;
+------------+
| New Job Id |
+------------+
| 40         |
+------------+
```

4.17.3 SUBMIT JOB FLUSH

`SUBMIT JOB FLUSH` 语句将当前图空间内存中的 RocksDB memfile 写入硬盘。

示例：

```
nebula> SUBMIT JOB FLUSH;
+------------+
| New Job Id |
+------------+
```


2022 vesoft Inc.

```
| 96         |
+------------+
```

4.17.4 SUBMIT JOB STATS

`SUBMIT JOB STATS` 语句会在当前图空间内启动一个作业，该作业对当前图空间进行统计。作业完成后，用户可以使用 `SHOW STATS` 语句列出统计结果。详情请参见 SHOW STATS。

> **Note**
>
> 如果存储在 Nebula Graph 中的数据有变化，为了获取最新的统计结果，请重新执行 `SUBMIT JOB STATS` 。

示例：

```
nebula> SUBMIT JOB STATS;
+------------+
| New Job Id |
+------------+
| 9          |
+------------+
```

4.17.5 SUBMIT JOB DOWNLOAD/INGEST

`SUBMIT JOB DOWNLOAD HDFS` 和 `SUBMIT JOB INGEST` 命令用于将 SST 文件导入 Nebula Graph。详情参见导入 SST 文件数据。

`SUBMIT JOB DOWNLOAD HDFS` 语句会下载指定的 HDFS 上的 SST 文件。

`SUBMIT JOB INGEST` 语句会将下载的 SST 文件导入图空间。

示例：

```
nebula> SUBMIT JOB DOWNLOAD HDFS "hdfs://192.168.10.100:9000/sst";
+------------+
| New Job Id |
+------------+
| 10         |
+------------+

nebula> SUBMIT JOB INGEST;
+------------+
| New Job Id |
+------------+
| 11         |
+------------+
```

4.17.6 SHOW JOB

Meta 服务将 `SUBMIT JOB` 请求解析为多个任务，然后分配给进程 nebula-storaged。 `SHOW JOB <job_id>` 语句显示当前图空间内指定作业和相关任务的信息。

`job_id` 在执行 `SUBMIT JOB` 语句时会返回。

示例：

```
nebula> SHOW JOB 9;
+---------------+-----------------+------------+----------------------------+----------------------------+-------------+
| Job Id(TaskId) | Command(Dest)   | Status     | Start Time                 | Stop Time                  | Error Code  |
+---------------+-----------------+------------+----------------------------+----------------------------+-------------+
| 9              | "STATS"         | "FINISHED" | 2022-04-12T08:47:33.000000 | 2022-04-12T08:47:33.000000 | "SUCCEEDED" |
| 0              | "192.168.8.100" | "FINISHED" | 2022-04-12T08:47:33.000000 | 2022-04-12T08:47:34.000000 | "SUCCEEDED" |


```


2022 vesoft Inc.

```
| 1             | "192.168.8.101" | "FINISHED" | 2022-04-12T08:47:33.000000 | 2022-04-12T08:47:34.000000 | "SUCCEEDED" |
+---------------+-----------------+------------+----------------------------+----------------------------+-------------+
```

参数	说明
`Job Id(TaskId)`	第一行显示作业 ID，其他行显示作业相关的任务 ID。
`Command(Dest)`	第一行显示执行的作业命令名称，其他行显示任务对应的 nebula-storaged 进程。
`Status`	显示作业或任务的状态。详情请参见作业状态。
`Start Time`	显示作业或任务开始执行的时间。
`Stop Time`	显示作业或任务结束执行的时间，结束后的状态包括 `FINISHED` 、`FAILED` 或 `STOPPED` 。
`Error Code`	显示作业或任务的错误码。

作业状态

作业状态的说明如下。

状态	说明
QUEUE	作业或任务在等待队列中。此阶段 `Start Time` 为空。
RUNNING	作业或任务在执行中。`Start Time` 为该阶段的起始时间。
FINISHED	作业或任务成功完成。`Stop Time` 为该阶段的起始时间。
FAILED	作业或任务失败。`Stop Time` 为该阶段的起始时间。
STOPPED	作业或任务停止。`Stop Time` 为该阶段的起始时间。
REMOVED	作业或任务被删除。

状态转换的说明如下。

```
Queue -- running -- finished -- removed
     \         \                /
      \         \ -- failed -- /
       \         \            /
        \ ---------- stopped -/
```

4.17.7 SHOW JOBS

`SHOW JOBS` 语句列出当前图空间内所有未过期的作业。

作业的默认过期时间为一周。如果需要修改过期时间，请修改 Meta 服务的参数 `job_expired_secs` 。详情请参见 Meta 服务配置。

示例：

```
nebula> SHOW JOBS;
+--------+---------------------+------------+----------------------------+----------------------------+
| Job Id | Command             | Status     | Start Time                 | Stop Time                  |
+--------+---------------------+------------+----------------------------+----------------------------+
| 34     | "STATS"             | "FINISHED" | 2021-11-01T03:32:27.000000 | 2021-11-01T03:32:27.000000 |
| 33     | "FLUSH"             | "FINISHED" | 2021-11-01T03:32:15.000000 | 2021-11-01T03:32:15.000000 |
| 32     | "COMPACT"           | "FINISHED" | 2021-11-01T03:32:06.000000 | 2021-11-01T03:32:06.000000 |
| 31     | "REBUILD_TAG_INDEX" | "FINISHED" | 2021-10-29T05:39:16.000000 | 2021-10-29T05:39:17.000000 |
| 10     | "COMPACT"           | "FINISHED" | 2021-10-26T02:27:05.000000 | 2021-10-26T02:27:05.000000 |
+--------+---------------------+------------+----------------------------+----------------------------+
```

4.17.8 STOP JOB

`STOP JOB <job_id>` 语句可以停止当前图空间内未完成的作业。

示例：

```
nebula> STOP JOB 22;
+--------------+
```


2022 vesoft Inc.

```
| Result       |
+--------------+
| "Job stopped" |
+--------------+
```

4.17.9 RECOVER JOB

`RECOVER JOB [<job_id>]` 语句可以重新执行当前图空间内状态为 `QUEUE` 、 `FAILED` 、 `STOPPED` 的作业，未指定 `<job_id>` 时，会从最早的作业开始尝试重新执行，并返回已恢复的作业数量。

示例：

```
nebula> RECOVER JOB;
+-------------------+
| Recovered job num |
+-------------------+
| 5 job recovered   |
+-------------------+
```

4.17.10 FAQ

如何排查作业问题?

`SUBMIT JOB` 操作使用的是 HTTP 端口，请检查 Storage 服务机器上的 HTTP 端口是否正常工作。用户可以执行如下命令调试:

```
curl "http://{storaged-ip}:19779/admin?space={space_name}&op=compact"
```

最后更新: June 20, 2023

2022 vesoft Inc.

5. 安装部署

5.1 准备编译、安装和运行 Nebula Graph 的环境

本文介绍编译、安装 Nebula Graph 的要求和建议，以及如何预估集群运行所需的资源。

5.1.1 关于存储硬件

Nebula Graph 是针对 NVMe SSD 进行设计和实现的，所有默认参数都是基于 SSD 设备进行调优，要求极高的 IOPS 和极低的 Latency。

- 不建议使用 HDD；因为其 IOPS 性能差，随机寻道延迟高；会遇到大量问题。
- 不要使用远端存储设备（如 NAS 或 SAN），不要外接基于 HDFS 或者 Ceph 的虚拟硬盘。
- 不要使用磁盘阵列（RAID）。
- 使用本地 SSD 设备；或 AWS Provisioned IOPS SSD 或等价云产品。

5.1.2 关于 CPU 架构

Enterpriseonly

企业版支持在 ARM 架构（包括 Apple Mac M1 和华为鲲鹏）上运行。访问官网获取商业支持。

Note

社区用户可以在 ARM MacOS 的 Docker Desktop 上或者 ARM Linux Server 上运行容器化的 Nebula Graph。

5.1.3 编译源码要求

硬件要求

类型	要求
CPU 架构	x86_64
内存	4 GB
硬盘	10 GB，SSD

操作系统要求

当前仅支持在 Linux 系统中编译 Nebula Graph，建议使用内核版本为 `4.15` 及以上版本的 Linux 系统。

Note

在内核版本低于要求的 Linux 系统中安装 Nebula Graph 可使用 RPM、DEB 或者 TAR 文件。

2022 vesoft Inc.

软件要求

软件版本需要如下表所示，如果版本不符合要求，请按照安装编译所需软件中的步骤进行操作。

软件名称	版本	备注
glibc	2.17 及以上	执行命令 `ldd --version` 检查版本。
make	任意稳定版本	-
m4	任意稳定版本	-
git	任意稳定版本	-
wget	任意稳定版本	-
unzip	任意稳定版本	-
xz	任意稳定版本	-
readline-devel	任意稳定版本	-
ncurses-devel	任意稳定版本	-
zlib-devel	任意稳定版本	-
g++	8.5.0 及以上	执行命令 `g++ -v` 检查版本。
cmake	3.14.0 及以上	执行命令 `cmake --version` 检查版本。
curl	任意稳定版本	-
redhat-lsb-core	任意稳定版本	-
libstdc++-static	任意稳定版本	仅在 CentOS 8+、RedHat 8+、Fedora 中需要。
libasan	任意稳定版本	仅在 CentOS 8+、RedHat 8+、Fedora 中需要。
bzip2	任意稳定版本	-

其他第三方软件将在安装（cmake）阶段自动下载并安装到 `build` 目录中。

2022 vesoft Inc.

安装编译所需软件

如果部分依赖软件缺失或者版本不满足要求，根据如下步骤手动安装。可根据实际情况删减命令中要安装的软件、跳过无需执行的步骤。

1. 安装依赖包。

- CentOS、RedHat、Fedora 用户请执行如下命令：

```
$ yum update
$ yum install -y make \
                 m4 \
                 git \
                 wget \
                 unzip \
                 xz \
                 readline-devel \
                 ncurses-devel \
                 zlib-devel \
                 gcc \
                 gcc-c++ \
                 cmake \
                 curl \
                 redhat-lsb-core \
                 bzip2
// 仅 CentOS 8+、RedHat 8+、Fedora 需要安装 libstdc++-static 和 libasan。
$ yum install -y libstdc++-static libasan
```

- Debian 和 Ubuntu 用户请执行如下命令：

```
$ apt-get update
$ apt-get install -y make \
                     m4 \
                     git \
                     wget \
                     unzip \
                     xz-utils \
                     curl \
                     lsb-core \
                     build-essential \
                     libreadline-dev \
                     ncurses-dev \
                     cmake
```

2. 检查主机上的 G++ 和 CMake 版本是否正确。版本信息请参见软件要求。

```
$ g++ --version
$ cmake --version
```

如果版本正确，则软件依赖已准备完毕，忽略后续步骤；如果不正确，根据不符合版本要求的软件执行后续步骤。

3. 如果 CMake 版本不符合要求，访问 CMake 官网以获取符合需要的版本。
4. 如果 G++ 版本不符合要求，自行到 GCC 官网获取相应版本或根据以下方法安装合适版本的 GCC。

- CentOS 系统的用户依次执行以下命令：

```
yum install centos-release-scl
yum install devtoolset-11
scl enable devtoolset-11 'bash'
```

- Ubuntu 系统的用户依次执行以下命令：

```
add-apt-repository ppa:ubuntu-toolchain-r/test
apt install gcc-11 g++-11
```


2022 vesoft Inc.

5.1.4 测试环境要求

硬件要求

类型	要求
CPU 架构	x86_64
CPU 核数	4
内存	8 GB
硬盘	100 GB，SSD

操作系统要求

当前仅支持在 Linux 系统中安装 Nebula Graph，建议在测试环境中使用内核版本为 `3.9` 及以上版本的 Linux 系统。

服务架构建议

进程	建议数量
metad（meta 数据服务进程）	1
storaged（存储服务进程）	≥1
graphd（查询引擎服务进程）	≥1

例如单机测试环境，用户可以在机器上部署 1 个 metad、1 个 storaged 和 1 个 graphd 进程。

对于更常见的测试环境，例如三台机器构成的集群，用户可以按照如下方案部署 Nebula Graph。

机器名称	**metad** 进程数量	**storaged** 进程数量	**graphd** 进程数量
A	1	1	1
B	-	1	1
C	-	1	1

5.1.5 生产环境运行要求

硬件要求

类型	要求
CPU 架构	x86_64
CPU 核数	48
内存	96 GB
硬盘	2 * 900 GB，NVMe SSD

操作系统要求

当前仅支持在 Linux 系统中安装 Nebula Graph，建议在生产环境中使用内核版本为 `3.9` 及以上版本的 Linux 系统。

用户可以通过调整一些内核参数来提高 Nebula Graph 性能，详情请参见内核配置。

2022 vesoft Inc.

服务架构建议

Danger

不要跨机房部署集群。

进程	数量
metad（meta 数据服务进程）	3
storaged（存储服务进程）	≥3
graphd（查询引擎服务进程）	≥3

有且仅有 3 个 metad 进程，每个 metad 进程会自动创建并维护 meta 数据的一个副本。

storaged 进程的数量不会影响图空间副本的数量。

用户可以在一台机器上部署多个不同进程，例如五台机器构成的集群，用户可以按照如下方案部署 Nebula Graph。

机器名称	**metad** 进程数量	**storaged** 进程数量	**graphd** 进程数量
A	1	1	1
B	1	1	1
C	1	1	1
D	-	1	1
E	-	1	1

2022 vesoft Inc.

5.1.6 Nebula Graph 资源要求

用户可以预估一个 3 副本 Nebula Graph 集群所需的内存、硬盘空间和分区数量。

资源	单位	计算公式	说明
硬盘空间	Bytes	`点和边的总数` * `属性的平均字节大小` * 6 * 120%	由于边存在存储放大现象，所以需要 `点和边的总数` * `属性的平均字节大小` * 6 的空间，详情请参见切边与存储放大。
内存	Bytes	[`点和边的总数` * 16 + `RocksDB 实例数量` * (`write_buffer_size` * `max_write_buffer_number` + `块缓存大小`)] * 120%	`点和边的总数` * 16 是 BloomFilter 需要占用的内存空间，`write_buffer_size` 和 `max_write_buffer_number` 是 RocksDB 内存相关参数，详情请参见 MemTable。块缓存大小请参见 Memory usage in RocksDB。
分区数量	-	`集群硬盘数量` * `disk_partition_num_multiplier`	`disk_partition_num_multiplier` 是一个用于衡量硬盘性能的整数，取值范围 2~20。建议在计算 SSD 硬盘的分区数量时使用 20 做参数值，HDD 硬盘使用 2。

- 问题 1：为什么磁盘空间和内存都要乘以 120%?

  答：额外的 20% 用于缓冲。

- 问题 2：如何获取 RocksDB 实例数量?

  答：在 `etc` 目录内查看配置文件 `nebula-storaged.conf`，`--data_path` 选项中的每个目录对应一个 RocksDB 实例，目录总数即是 RocksDB 实例数量。

> **Note**
>
> 用户可以在配置文件 `nebula-storaged.conf` 中添加 `--enable_partitioned_index_filter=true` 来降低 bloom 过滤器占用的内存大小，但是在某些随机寻道（random-seek）的情况下，可能会降低读取性能。

最后更新: June 20, 2023

2022 vesoft Inc.

5.2 编译与安装

5.2.1 使用源码安装 Nebula Graph

使用源码安装 Nebula Graph 允许自定义编译和安装设置，并测试最新特性。

前提条件

- 准备正确的编译环境。参见软硬件要求和安装三方库依赖包。

  Note

  暂不支持离线编译 Nebula Graph。

- 待安装 Nebula Graph 的主机可以访问互联网。

安装步骤

1. 克隆 Nebula Graph 的源代码到主机。

- [推荐] 如果需要安装3.1.0版本的 Nebula Graph，执行如下命令：

```
$ git clone --branch release-3.1 https://github.com/vesoft-inc/nebula.git
```

- 如果需要安装最新的开发版本用于测试，执行如下命令克隆 master 分支的代码：

```
$ git clone https://github.com/vesoft-inc/nebula.git
```

2. 进入目录 `nebula`。

```
$ cd nebula
```

3. 创建目录 `build` 并进入该目录。

```
$ mkdir build && cd build
```

4. 使用 CMake 生成 makefile 文件。

   Note

   默认安装路径为 `/usr/local/nebula`，如果需要修改路径，请在下方命令内增加参数 `-DCMAKE_INSTALL_PREFIX=<installation_path>`。

   更多 CMake 参数说明，请参见 CMake 参数。

```
$ cmake -DCMAKE_INSTALL_PREFIX=/usr/local/nebula -DENABLE_TESTING=OFF -DCMAKE_BUILD_TYPE=Release ..
```

5. 编译 Nebula Graph。

   Note

   检查软硬件要求和安装三方库依赖包。

为了适当地加快编译速度，可以使用选项 `-j` 并行编译。并行数量 `N` 建议为$\min(\text{CPU}核数，\frac{内存 (GB)}{2})$。

```
$ make -j{N} # E.g., make -j2
```


2022 vesoft Inc.

6. 安装 Nebula Graph。

```
$ sudo make install
```

7. 安装目录下 `etc/` 目录中（默认为 `/usr/local/nebula/etc`）的配置文件为参考模版，用户可以根据需要创建自己的配置文件。如果要使用 `script` 目录下的脚本，启动、停止、重启、中止和查看服务，配置文件需要命名为 `nebula-graph.conf`，`nebula-metad.conf` 和 `nebula-storaged.conf`。

更新 **master** 版本

master 分支的代码更新速度快，如果安装了 master 分支对应的开发版 Nebula Graph，可根据以下步骤更新版本。

1. 在目录 `nebula` 中，执行命令 `git pull upstream master` 更新源码。
2. 在目录 `nebula/build` 中，重新执行 `make -j{N}` 和 `make install`。

下一步

- （企业版）设置 License
- 管理 Nebula Graph 服务

CMake 参数

使用方法

```
$ cmake -D<variable>=<value> ...
```

下文的 CMake 参数可以在配置 (CMake) 阶段用来调整编译设置。

CMAKE_INSTALL_PREFIX

`CMAKE_INSTALL_PREFIX` 指定 Nebula Graph 服务模块、脚本和配置文件的安装路径，默认路径为 `/usr/local/nebula`。

ENABLE_WERROR

默认值为 `ON`，表示将所有警告（warning）变为错误（error）。如果有必要，用户可以设置为 `OFF`。

ENABLE_TESTING

默认值为 `ON`，表示单元测试服务由 Nebula Graph 服务构建。如果只需要服务模块，可以设置为 `OFF`。

ENABLE_ASAN

默认值为 `OFF`，表示关闭内存问题检测工具 ASan（AddressSanitizer）。该工具是为 Nebula Graph 开发者准备的，如果需要开启，可以设置为 `ON`。

2022 vesoft Inc.

CMAKE_BUILD_TYPE

`CMAKE_BUILD_TYPE` 控制 Nebula Graph 的 build 方法，取值说明如下：

- `Debug`

  `CMAKE_BUILD_TYPE` 的默认值，表示 build 过程中只记录 debug 信息，不使用优化选项。

- `Release`

  build 过程中使用优化选项，不记录 debug 信息。

- `RelWithDebInfo`

  build 过程中既使用优化选项，也记录 debug 信息。

- `MinSizeRel`

  build 过程中仅通过优化选项控制代码大小，不记录 debug 信息。

ENABLE_INCLUDE_WHAT_YOU_USE

默认值为 `OFF` 。当其值为 `ON` 且系统中安装了include-what-you-use，系统将在生成 makefile 过程中报告工程源码中包含的冗余头文件。

NEBULA_USE_LINKER

指定链接程序的方式：

- 默认值为 `bfd` ，表示使用 ld.bfd 链接程序。
- 如果系统中安装了 lld 链接器，可设置参数值为 `lld` ，表示使用 ld.lld 链接程序。
- 如果系统中安装了 gold 链接器，可设为 `gold` ，表示使用 ld.gold 链接程序。

CMAKE_C_COMPILER/CMAKE_CXX_COMPILER

通常情况下，CMake 会自动查找并使用主机上的 C/C++编译器，但是如果编译器没有安装在标准路径，或者想使用其他编译器，请执行如下命令指定目标编译器的安装路径：

```
$ cmake -DCMAKE_C_COMPILER=<path_to_gcc/bin/gcc> -DCMAKE_CXX_COMPILER=<path_to_gcc/bin/g++> ..
$ cmake -DCMAKE_C_COMPILER=<path_to_clang/bin/clang> -DCMAKE_CXX_COMPILER=<path_to_clang/bin/clang++> ..
```

ENABLE_CCACHE

`ENABLE_CCACHE` 默认值为 `ON` ，表示使用 Ccache（compiler cache）工具加速编译。

如果想要禁用 ccache，仅仅设置 `ENABLE_CCACHE=OFF` 是不行的，因为在某些平台上，ccache 会代理当前编译器，因此还需要设置环境变量 `export CCACHE_DISABLE=true` ，或者在文件 `~/.ccache/ccache.conf` 中添加 `disable=true` 。更多信息请参见 ccache official documentation。

NEBULA_THIRDPARTY_ROOT

`NEBULA_THIRDPARTY_ROOT` 指定第三方软件的安装路径，默认路径为 `/opt/vesoft/third-party` 。

问题排查

如果出现编译失败，请参考以下建议：

1. 检查操作系统版本是否符合要求、内存和硬盘空间是否足够。
2. 检查 third-party 是否正确安装。
3. 使用 `make -j1` 降低编译并发度。

最后更新: June 20, 2023

2022 vesoft Inc.

5.2.2 使用 RPM 或 DEB 包安装 Nebula Graph

RPM 和 DEB 是 Linux 系统下常见的两种安装包格式，本文介绍如何使用 RPM 或 DEB 文件在一台机器上快速安装 Nebula Graph。

Note

部署 Nebula Graph 集群的方式参见使用 RPM/DEB 包部署集群。

Enterpriseonly

企业版请发送邮件至 inquiry@vesoft.com。

前提条件

安装 wget

下载安装包

阿里云 OSS 下载

- 下载 release 版本

  URL 格式如下：

```
//Centos 7
https://oss-cdn.nebula-graph.com.cn/package/<release_version>/nebula-graph-<release_version>.el7.x86_64.rpm

//Centos 8
https://oss-cdn.nebula-graph.com.cn/package/<release_version>/nebula-graph-<release_version>.el8.x86_64.rpm

//Ubuntu 1604
https://oss-cdn.nebula-graph.com.cn/package/<release_version>/nebula-graph-<release_version>.ubuntu1604.amd64.deb

//Ubuntu 1804
https://oss-cdn.nebula-graph.com.cn/package/<release_version>/nebula-graph-<release_version>.ubuntu1804.amd64.deb

//Ubuntu 2004
https://oss-cdn.nebula-graph.com.cn/package/<release_version>/nebula-graph-<release_version>.ubuntu2004.amd64.deb
```

  例如要下载适用于 `Centos 7.5` 的 `3.1.0` 安装包：

```
wget https://oss-cdn.nebula-graph.com.cn/package/3.1.0/nebula-graph-3.1.0.el7.x86_64.rpm
wget https://oss-cdn.nebula-graph.com.cn/package/3.1.0/nebula-graph-3.1.0.el7.x86_64.rpm.sha256sum.txt
```

  下载适用于 `ubuntu 1804` 的 `3.1.0` 安装包：

```
wget https://oss-cdn.nebula-graph.com.cn/package/3.1.0/nebula-graph-3.1.0.ubuntu1804.amd64.deb
wget https://oss-cdn.nebula-graph.com.cn/package/3.1.0/nebula-graph-3.1.0.ubuntu1804.amd64.deb.sha256sum.txt
```


2022 vesoft Inc.

- 下载日常开发版本 (nightly)

  Danger

  - nightly 版本通常用于测试新功能、新特性，请不要在生产环境中使用 nightly 版本。
  - nightly 版本不保证每日都能完整发布，也不保证是否会更改文件名。

  URL 格式如下：

  ```
  //Centos 7
  https://oss-cdn.nebula-graph.com.cn/package/nightly/<yyyy.mm.dd>/nebula-graph-<yyyy.mm.dd>-nightly.el7.x86_64.rpm

  //Centos 8
  https://oss-cdn.nebula-graph.com.cn/package/nightly/<yyyy.mm.dd>/nebula-graph-<yyyy.mm.dd>-nightly.el8.x86_64.rpm

  //Ubuntu 1604
  https://oss-cdn.nebula-graph.com.cn/package/nightly/<yyyy.mm.dd>/nebula-graph-<yyyy.mm.dd>-nightly.ubuntu1604.amd64.deb

  //Ubuntu 1804
  https://oss-cdn.nebula-graph.com.cn/package/nightly/<yyyy.mm.dd>/nebula-graph-<yyyy.mm.dd>-nightly.ubuntu1804.amd64.deb

  //Ubuntu 2004
  https://oss-cdn.nebula-graph.com.cn/package/nightly/<yyyy.mm.dd>/nebula-graph-<yyyy.mm.dd>-nightly.ubuntu2004.amd64.deb
  ```

  例如要下载 `2021.11.24` 适用于 `Centos 7.5` 的 `2.x` 安装包：

  ```
  wget https://oss-cdn.nebula-graph.com.cn/package/nightly/2021.11.24/nebula-graph-2021.11.24-nightly.el7.x86_64.rpm
  wget https://oss-cdn.nebula-graph.com.cn/package/nightly/2021.11.24/nebula-graph-2021.11.24-nightly.el7.x86_64.rpm.sha256sum.txt
  ```

  要下载 `2021.11.24` 适用于 `Ubuntu 1804` 的 `2.x` 安装包：

  ```
  wget https://oss-cdn.nebula-graph.com.cn/package/nightly/2021.11.24/nebula-graph-2021.11.24-nightly.ubuntu1804.amd64.deb
  wget https://oss-cdn.nebula-graph.com.cn/package/nightly/2021.11.24/nebula-graph-2021.11.24-nightly.ubuntu1804.amd64.deb.sha256sum.txt
  ```

安装 **Nebula Graph**

- 安装 RPM 包

  ```
  $ sudo rpm -ivh --prefix=<installation_path> <package_name>
  ```

  `--prefix` 为可选项，用于指定安装路径。如不设置，系统会将 Nebula Graph 安装到默认路径 `/usr/local/nebula/` 。

  例如，要在默认路径下安装3.1.0版本的 RPM 包，运行如下命令：

  ```
  sudo rpm -ivh nebula-graph-3.1.0.el7.x86_64.rpm
  ```

- 安装 DEB 包

  ```
  $ sudo dpkg -i <package_name>
  ```

  Note

  使用 DEB 包安装 Nebula Graph 时不支持自定义安装路径。默认安装路径为 `/usr/local/nebula/` 。

  例如安装3.1.0版本的 DEB 包：

  ```
  sudo dpkg -i nebula-graph-3.1.0.ubuntu1804.amd64.deb
  ```


2022 vesoft Inc.

后续操作

- （企业版）设置 License
- 启动 Nebula Graph
- 连接 Nebula Graph

最后更新: June 20, 2023

2022 vesoft Inc.

5.2.3 使用 tar.gz 文件安装 Nebula Graph

用户可以下载打包好的 tar.gz 文件快速安装 Nebula Graph。

> **Note**
>
> Nebula Graph 从 2.6.0 版本起提供 tar.gz 文件。

操作步骤

1. 使用如下地址下载 Nebula Graph 的 tar.gz 文件。

   下载前需将 `<release_version>` 替换为需要下载的版本。

```
//Centos 7
https://oss-cdn.nebula-graph.com.cn/package/<release_version>/nebula-graph-<release_version>.el7.x86_64.tar.gz
//Checksum
https://oss-cdn.nebula-graph.com.cn/package/<release_version>/nebula-graph-<release_version>.el7.x86_64.tar.gz.sha256sum.txt

//Centos 8
https://oss-cdn.nebula-graph.com.cn/package/<release_version>/nebula-graph-<release_version>.el8.x86_64.tar.gz
//Checksum
https://oss-cdn.nebula-graph.com.cn/package/<release_version>/nebula-graph-<release_version>.el8.x86_64.tar.gz.sha256sum.txt

//Ubuntu 1604
https://oss-cdn.nebula-graph.com.cn/package/<release_version>/nebula-graph-<release_version>.ubuntu1604.amd64.tar.gz
//Checksum
https://oss-cdn.nebula-graph.com.cn/package/<release_version>/nebula-graph-<release_version>.ubuntu1604.amd64.tar.gz.sha256sum.txt

//Ubuntu 1804
https://oss-cdn.nebula-graph.com.cn/package/<release_version>/nebula-graph-<release_version>.ubuntu1804.amd64.tar.gz
//Checksum
https://oss-cdn.nebula-graph.com.cn/package/<release_version>/nebula-graph-<release_version>.ubuntu1804.amd64.tar.gz.sha256sum.txt

//Ubuntu 2004
https://oss-cdn.nebula-graph.com.cn/package/<release_version>/nebula-graph-<release_version>.ubuntu2004.amd64.tar.gz
//Checksum
https://oss-cdn.nebula-graph.com.cn/package/<release_version>/nebula-graph-<release_version>.ubuntu2004.amd64.tar.gz.sha256sum.txt
```

   例如，要下载适用于 `CentOS 7.5` 的 Nebula Graph release-3.1 tar.gz 文件，运行以下命令：

```
wget https://oss-cdn.nebula-graph.com.cn/package/3.1.0/nebula-graph-3.1.0.el7.x86_64.tar.gz
```

2. 解压 tar.gz 文件到 Nebula Graph 安装目录。

```
tar -xvzf <tar.gz_file_name> -C <install_path>
```

- `tar.gz_file_name` 表示 tar.gz 文件的名称。
- `install_path` 表示安装路径。

  例如：

```
tar -xvzf nebula-graph-3.1.0.el7.x86_64.tar.gz -C /home/joe/nebula/install
```

3. 修改配置文件名以应用配置。

   进入解压出的目录，将子目录 `etc` 中的文件 `nebula-graphd.conf.default` 、`nebula-metad.conf.default` 和 `nebula-storaged.conf.default` 重命名，删除 `.default`，即可应用 Nebula Graph 的默认配置。如需修改配置，参见配置管理。

至此，Nebula Graph 安装完毕。

下一步

- （企业版）设置 License
- 管理 Nebula Graph 服务

2022 vesoft Inc.

最后更新: June 20, 2023

2022 vesoft Inc.

5.2.4 使用 Docker Compose 部署 Nebula Graph

使用 Docker Compose 可以基于准备好的配置文件快速部署 Nebula Graph 服务，仅建议在测试 Nebula Graph 功能时使用该方式。

前提条件

- 主机上安装如下应用程序。

应用程序	推荐版本	官方安装参考
Docker	最新版本	Install Docker Engine
Docker Compose	最新版本	Install Docker Compose
Git	最新版本	Download Git

- 如果使用非 root 用户部署 Nebula Graph，请授权该用户 Docker 相关的权限。详细信息，请参见 Manage Docker as a non-root user。
- 启动主机上的 Docker 服务。
- 如果已经通过 Docker Compose 在主机上部署了另一个版本的 Nebula Graph，为避免兼容性问题，需要删除目录 `nebula-docker-compose/data` 。

部署和连接 **Nebula Graph**

1. 通过 Git 克隆 `nebula-docker-compose` 仓库的 `3.1.0` 分支到主机。

Danger

`master` 分支包含最新的未测试代码。请不要在生产环境使用此版本。

```
$ git clone -b release-3.1 https://github.com/vesoft-inc/nebula-docker-compose.git
```

Note

Docker Compose 的 `x.y` 版本对齐内核的 `x.y` 版本，对于内核 `z` 版本，Docker Compose 不会发布对应的 `z` 版本，但是会拉取 `z` 版本的内核镜像。

2. 切换至目录 `nebula-docker-compose` 。

```
$ cd nebula-docker-compose/
```

3. 执行如下命令启动 Nebula Graph 服务。

从 3.0.2 开始，Nebula Graph 在 Docker Hub 上的 Docker 支持 ARM64 架构。用户可以在 ARM macOS 的 Docker Desktop 上或者 ARM Linux Server 上运行容器化的 Nebula Graph。

Note

如果长期未内核更新镜像，请先更新 Nebula Graph 镜像和 Nebula Console 镜像。

```
[nebula-docker-compose]$ docker-compose up -d
Creating nebula-docker-compose_metad0_1 ... done
Creating nebula-docker-compose_metad2_1 ... done
Creating nebula-docker-compose_metad1_1 ... done
Creating nebula-docker-compose_graphd2_1   ... done
Creating nebula-docker-compose_graphd_1    ... done
Creating nebula-docker-compose_graphd1_1   ... done
Creating nebula-docker-compose_storaged0_1 ... done
Creating nebula-docker-compose_storaged2_1 ... done
Creating nebula-docker-compose_storaged1_1 ... done
```


2022 vesoft Inc.

Note

上述服务的更多信息，请参见架构总览。

4. 连接 Nebula Graph。

Compatibility

从 3.1 版本开始，Docker-compose 会自动启动 Nebula Console 镜像的容器，并将 Storage 主机增加至集群中（即 `ADD HOSTS` 命令）。

a. 使用 `docker-compose ps` 命令查看 Nebula Console 容器名称。

```
$ docker-compose ps
          Name                        Command             State                 Ports
-------------------------------------------------------------------------------------------
nebula-docker-compose_console_1     sh -c sleep 3 &&          Up
                                    nebula-co ...
......
```

b. 进入 Nebula Console 容器中。

```
$ docker exec -it nebula-docker-compose_console_1 /bin/sh
/ #
```

c. 通过 Nebula Console 连接 Nebula Graph。

```
/ # ./usr/local/bin/nebula-console -u <user_name> -p <password> --address=graphd --port=9669
```

Note

默认情况下，身份认证功能是关闭的，只能使用已存在的用户名（默认为 `root`）和任意密码登录。如果想使用身份认证，请参见身份认证。

d. 查看集群状态。

```
nebula> SHOW HOSTS;
+-------------+------+-----------+----------+--------------+----------------------+------------------------+---------+
| Host        | Port | HTTP port | Status   | Leader count | Leader distribution  | Partition distribution | Version |
+-------------+------+-----------+----------+--------------+----------------------+------------------------+---------+
| "storaged0" | 9779 | 19669     | "ONLINE" | 0            | "No valid partition" | "No valid partition"   | "3.1.0" |
| "storaged1" | 9779 | 19669     | "ONLINE" | 0            | "No valid partition" | "No valid partition"   | "3.1.0" |
| "storaged2" | 9779 | 19669     | "ONLINE" | 0            | "No valid partition" | "No valid partition"   | "3.1.0" |
+-------------+------+-----------+----------+--------------+----------------------+------------------------+---------+
```

5. 执行两次 `exit` 可以退出容器。

查看 **Nebula Graph** 服务的状态和端口

执行命令 `docker-compose ps` 可以列出 Nebula Graph 服务的状态和端口。

```
$ docker-compose ps
nebula-docker-compose_console_1     sh -c sleep 3 &&                Up
                                    nebula-co ...
nebula-docker-compose_graphd1_1     /usr/local/nebula/bin/nebu ...   Up      0.0.0.0:49174->19669/tcp,:::49174->19669/tcp, 0.0.0.0:49171->19670/tcp,:::49171->19670/tcp, 0.0.0.0:49177->9669/
tcp,:::49177->9669/tcp
nebula-docker-compose_graphd2_1     /usr/local/nebula/bin/nebu ...   Up      0.0.0.0:49175->19669/tcp,:::49175->19669/tcp, 0.0.0.0:49172->19670/tcp,:::49172->19670/tcp, 0.0.0.0:49178->9669/
tcp,:::49178->9669/tcp
nebula-docker-compose_graphd_1      /usr/local/nebula/bin/nebu ...   Up      0.0.0.0:49180->19669/tcp,:::49180->19669/tcp, 0.0.0.0:49179->19670/tcp,:::49179->19670/tcp, 0.0.0.0:9669->9669/
tcp,:::9669->9669/tcp
nebula-docker-compose_metad0_1      /usr/local/nebula/bin/nebu ...   Up      0.0.0.0:49157->19559/tcp,:::49157->19559/tcp, 0.0.0.0:49154->19560/tcp,:::49154->19560/tcp, 0.0.0.0:49160->9559/
tcp,:::49160->9559/tcp, 9560/tcp
nebula-docker-compose_metad1_1      /usr/local/nebula/bin/nebu ...   Up      0.0.0.0:49156->19559/tcp,:::49156->19559/tcp, 0.0.0.0:49153->19560/tcp,:::49153->19560/tcp, 0.0.0.0:49159->9559/
tcp,:::49159->9559/tcp, 9560/tcp
nebula-docker-compose_metad2_1      /usr/local/nebula/bin/nebu ...   Up      0.0.0.0:49158->19559/tcp,:::49158->19559/tcp, 0.0.0.0:49155->19560/tcp,:::49155->19560/tcp, 0.0.0.0:49161->9559/
tcp,:::49161->9559/tcp, 9560/tcp
nebula-docker-compose_storaged0_1   /usr/local/nebula/bin/nebu ...   Up      0.0.0.0:49166->19779/tcp,:::49166->19779/tcp, 0.0.0.0:49163->19780/tcp,:::49163->19780/tcp, 9777/tcp, 9778/tcp,
0.0.0.0:49169->9779/tcp,:::49169->9779/tcp, 9780/tcp
nebula-docker-compose_storaged1_1   /usr/local/nebula/bin/nebu ...   Up      0.0.0.0:49165->19779/tcp,:::49165->19779/tcp, 0.0.0.0:49162->19780/tcp,:::49162->19780/tcp, 9777/tcp, 9778/tcp,
0.0.0.0:49168->9779/tcp,:::49168->9779/tcp, 9780/tcp
```


2022 vesoft Inc.

```
nebula-docker-compose_storaged2_1   /usr/local/nebula/bin/nebu ...   Up      0.0.0.0:49167->19779/tcp,:::49167->19779/tcp, 0.0.0.0:49164->19780/tcp,:::49164->19780/tcp, 9777/tcp, 9778/tcp,
0.0.0.0:49170->9779/tcp,:::49170->9779/tcp, 9780/tcp
```

Nebula Graph 默认使用 9669 端口为客户端提供服务，如果需要修改端口，请修改目录 nebula-docker-compose 内的文件 docker-compose.yaml ，然后重启 Nebula Graph 服务。

查看 **Nebula Graph** 服务的数据和日志

Nebula Graph 的所有数据和日志都持久化存储在 nebula-docker-compose/data 和 nebula-docker-compose/logs 目录中。

目录的结构如下：

```
nebula-docker-compose/
  |-- docker-compose.yaml
  ├──── data
  │     ├──── meta0
  │     ├──── meta1
  │     ├──── meta2
  │     ├──── storage0
  │     ├──── storage1
  │     └──── storage2
  └──── logs
        ├──── graph
        ├──── graph1
        ├──── graph2
        ├──── meta0
        ├──── meta1
        ├──── meta2
        ├──── storage0
        ├──── storage1
        └──── storage2
```

停止 **Nebula Graph** 服务

用户可以执行如下命令停止 Nebula Graph 服务：

```
$ docker-compose down
```

如果返回如下信息，表示已经成功停止服务。

```
Stopping nebula-docker-compose_console_1   ... done
Stopping nebula-docker-compose_graphd1_1   ... done
Stopping nebula-docker-compose_graphd2_1   ... done
Stopping nebula-docker-compose_storaged0_1 ... done
Stopping nebula-docker-compose_storaged1_1 ... done
Stopping nebula-docker-compose_storaged2_1 ... done
Stopping nebula-docker-compose_metad0_1    ... done
Stopping nebula-docker-compose_metad2_1    ... done
Stopping nebula-docker-compose_metad1_1    ... done
Removing nebula-docker-compose_console_1             ... done
Removing nebula-docker-compose_graphd1_1             ... done
Removing nebula-docker-compose_graphd2_1             ... done
Removing nebula-docker-compose_graphd_1              ... done
Removing nebula-docker-compose_storaged0_1           ... done
Removing nebula-docker-compose_storaged1_1           ... done
Removing nebula-docker-compose_storaged2_1           ... done
Removing nebula-docker-compose_metad0_1              ... done
Removing nebula-docker-compose_metad2_1              ... done
Removing nebula-docker-compose_metad1_1              ... done
Removing network nebula-docker-compose_nebula-net
```

Danger

命令 docker-compose down -v 的参数 -v 将会删除所有本地的数据。如果使用的是 nightly 版本，并且有一些兼容性问题，请尝试这个命令。

修改配置

Docker Compose 部署的 Nebula Graph，配置文件位置为 nebula-docker-compose/docker-compose.yaml ，修改该文件内的配置并重启服务即可使新配置生效。

具体的配置说明请参见配置管理。

2022 vesoft Inc.

常见问题

如何固定 DOCKER 映射到外部的端口?

在目录 nebula-docker-compose 内修改文件 docker-compose.yaml，将对应服务的 ports 设置为固定映射，例如：

```
graphd:
    image: vesoft/nebula-graphd:release-3.1
    ...
    ports:
      - 9669:9669
      - 19669
      - 19670
```

9669:9669 表示内部的 9669 映射到外部的端口也是 9669，下方的 19669 表示内部的 19669 映射到外部的端口是随机的。

如何升级/更新 NEBULA GRAPH 服务的 DOCKER 镜像?

1. 在文件 nebula-docker-compose/docker-compose.yaml 中，找到所有服务的 image 并修改其值为相应的镜像版本。
2. 在目录 nebula-docker-compose 内执行命令 docker-compose pull，更新 Graph 服务、Storage 服务、Meta 服务和 Nebula Console 的镜像。
3. 执行命令 docker-compose up -d 启动 Nebula Graph 服务。
4. 通过 Nebula Console 连接 Nebula Graph 后，分别执行命令 SHOW HOSTS GRAPH 、 SHOW HOSTS STORAGE 、 SHOW HOSTS META 查看各服务版本。

执行命令 DOCKER-COMPOSE PULL 报错 ERROR: TOOMANYREQUESTS

可能遇到如下错误:

```
ERROR: toomanyrequests: You have reached your pull rate limit. You may increase the limit by authenticating and upgrading: https://www.docker.com/increase-rate-limit
```

以上错误表示已达到 Docker Hub 的速率限制。解决方案请参见 Understanding Docker Hub Rate Limiting。

如何更新 NEBULA CONSOLE?

docker-compose pull 会同时更新 Nebula Graph 服务和 Nebula Console。

最后更新: June 20, 2023

2022 vesoft Inc.

5.2.5 使用 RPM/DEB 包部署 Nebula Graph 多机集群

Nebula Graph 不提供官方的集群部署工具，用户可以使用 RPM 或 DEB 文件手动部署集群。本文提供了部署集群的示例。

部署方案

机器名称	**IP** 地址	**graphd** 进程数量	**storaged** 进程数量	**metad** 进程数量
A	192.168.10.111	1	1	1
B	192.168.10.112	1	1	1
C	192.168.10.113	1	1	1
D	192.168.10.114	1	1	-
E	192.168.10.115	1	1	-

前提条件

- 准备 5 台用于部署集群的机器。
- 在集群中通过 NTP 服务同步时间。

手动部署流程

1. 安装 NEBULA GRAPH

在集群的每一台服务器上都安装 Nebula Graph，安装后暂不需要启动服务。安装方式请参见：

- 使用 RPM 或 DEB 包安装 Nebula Graph
- 使用源码安装 Nebula Graph

2. 修改配置文件

修改每个服务器上的 Nebula Graph 配置文件。

Nebula Graph 的所有配置文件均位于安装目录的 `etc` 目录内，包括 `nebula-graphd.conf` 、 `nebula-metad.conf` 和 `nebula-storaged.conf` ，用户可以只修改所需服务的配置文件。各个机器需要修改的配置文件如下。

机器名称	待修改配置文件
A	`nebula-graphd.conf` 、 `nebula-storaged.conf` 、 `nebula-metad.conf`
B	`nebula-graphd.conf` 、 `nebula-storaged.conf` 、 `nebula-metad.conf`
C	`nebula-graphd.conf` 、 `nebula-storaged.conf` 、 `nebula-metad.conf`
D	`nebula-graphd.conf` 、 `nebula-storaged.conf`
E	`nebula-graphd.conf` 、 `nebula-storaged.conf`

用户可以参考如下配置文件的内容，仅展示集群通信的部分设置，未展示的内容为默认设置，便于用户了解集群间各个服务器的关系。

2022 vesoft Inc.

Note

主要修改的配置是`meta_server_addrs`，所有配置文件都需要填写所有 Meta 服务的 IP 地址和端口，同时需要修改`local_ip`为机器本身的联网 IP 地址。配置参数的详细说明请参见：

- Meta 服务配置
- Graph 服务配置
- Storage 服务配置

- 机器 A 配置

- `nebula-graphd.conf`

```
########## networking ##########
# Comma separated Meta Server Addresses
--meta_server_addrs=192.168.10.111:9559,192.168.10.112:9559,192.168.10.113:9559
# Local IP used to identify the nebula-graphd process.
# Change it to an address other than loopback if the service is distributed or
# will be accessed remotely.
--local_ip=192.168.10.111
# Network device to listen on
--listen_netdev=any
# Port to listen on
--port=9669
```

- `nebula-storaged.conf`

```
########## networking ##########
# Comma separated Meta server addresses
--meta_server_addrs=192.168.10.111:9559,192.168.10.112:9559,192.168.10.113:9559
# Local IP used to identify the nebula-storaged process.
# Change it to an address other than loopback if the service is distributed or
# will be accessed remotely.
--local_ip=192.168.10.111
# Storage daemon listening port
--port=9779
```

- `nebula-metad.conf`

```
########## networking ##########
# Comma separated Meta Server addresses
--meta_server_addrs=192.168.10.111:9559,192.168.10.112:9559,192.168.10.113:9559
# Local IP used to identify the nebula-metad process.
# Change it to an address other than loopback if the service is distributed or
# will be accessed remotely.
--local_ip=192.168.10.111
# Meta daemon listening port
--port=9559
```


2022 vesoft Inc.

- 机器 B 配置

- `nebula-graphd.conf`

```
########## networking ##########
# Comma separated Meta Server Addresses
--meta_server_addrs=192.168.10.111:9559,192.168.10.112:9559,192.168.10.113:9559
# Local IP used to identify the nebula-graphd process.
# Change it to an address other than loopback if the service is distributed or
# will be accessed remotely.
--local_ip=192.168.10.112
# Network device to listen on
--listen_netdev=any
# Port to listen on
--port=9669
```

- `nebula-storaged.conf`

```
########## networking ##########
# Comma separated Meta server addresses
--meta_server_addrs=192.168.10.111:9559,192.168.10.112:9559,192.168.10.113:9559
# Local IP used to identify the nebula-storaged process.
# Change it to an address other than loopback if the service is distributed or
# will be accessed remotely.
--local_ip=192.168.10.112
# Storage daemon listening port
--port=9779
```

- `nebula-metad.conf`

```
########## networking ##########
# Comma separated Meta Server addresses
--meta_server_addrs=192.168.10.111:9559,192.168.10.112:9559,192.168.10.113:9559
# Local IP used to identify the nebula-metad process.
# Change it to an address other than loopback if the service is distributed or
# will be accessed remotely.
--local_ip=192.168.10.112
# Meta daemon listening port
--port=9559
```


2022 vesoft Inc.

- 机器 C 配置

- `nebula-graphd.conf`

```
########## networking ##########
# Comma separated Meta Server Addresses
--meta_server_addrs=192.168.10.111:9559,192.168.10.112:9559,192.168.10.113:9559
# Local IP used to identify the nebula-graphd process.
# Change it to an address other than loopback if the service is distributed or
# will be accessed remotely.
--local_ip=192.168.10.113
# Network device to listen on
--listen_netdev=any
# Port to listen on
--port=9669
```

- `nebula-storaged.conf`

```
########## networking ##########
# Comma separated Meta server addresses
--meta_server_addrs=192.168.10.111:9559,192.168.10.112:9559,192.168.10.113:9559
# Local IP used to identify the nebula-storaged process.
# Change it to an address other than loopback if the service is distributed or
# will be accessed remotely.
--local_ip=192.168.10.113
# Storage daemon listening port
--port=9779
```

- `nebula-metad.conf`

```
########## networking ##########
# Comma separated Meta Server addresses
--meta_server_addrs=192.168.10.111:9559,192.168.10.112:9559,192.168.10.113:9559
# Local IP used to identify the nebula-metad process.
# Change it to an address other than loopback if the service is distributed or
# will be accessed remotely.
--local_ip=192.168.10.113
# Meta daemon listening port
--port=9559
```

- 机器 D 配置

- `nebula-graphd.conf`

```
########## networking ##########
# Comma separated Meta Server Addresses
--meta_server_addrs=192.168.10.111:9559,192.168.10.112:9559,192.168.10.113:9559
# Local IP used to identify the nebula-graphd process.
# Change it to an address other than loopback if the service is distributed or
# will be accessed remotely.
--local_ip=192.168.10.114
# Network device to listen on
--listen_netdev=any
# Port to listen on
--port=9669
```

- `nebula-storaged.conf`

```
########## networking ##########
# Comma separated Meta server addresses
--meta_server_addrs=192.168.10.111:9559,192.168.10.112:9559,192.168.10.113:9559
# Local IP used to identify the nebula-storaged process.
# Change it to an address other than loopback if the service is distributed or
# will be accessed remotely.
--local_ip=192.168.10.114
# Storage daemon listening port
--port=9779
```


2022 vesoft Inc.

- 机器 E 配置

- `nebula-graphd.conf`

```
########## networking ##########
# Comma separated Meta Server Addresses
--meta_server_addrs=192.168.10.111:9559,192.168.10.112:9559,192.168.10.113:9559
# Local IP used to identify the nebula-graphd process.
# Change it to an address other than loopback if the service is distributed or
# will be accessed remotely.
--local_ip=192.168.10.115
# Network device to listen on
--listen_netdev=any
# Port to listen on
--port=9669
```

- `nebula-storaged.conf`

```
########## networking ##########
# Comma separated Meta server addresses
--meta_server_addrs=192.168.10.111:9559,192.168.10.112:9559,192.168.10.113:9559
# Local IP used to identify the nebula-storaged process.
# Change it to an address other than loopback if the service is distributed or
# will be accessed remotely.
--local_ip=192.168.10.115
# Storage daemon listening port
--port=9779
```

3. 启动集群

依次启动各个服务器上的对应进程。

机器名称	待启动的进程
A	graphd、storaged、metad
B	graphd、storaged、metad
C	graphd、storaged、metad
D	graphd、storaged
E	graphd、storaged

启动 Nebula Graph 进程的命令如下：

```
sudo /usr/local/nebula/scripts/nebula.service start <metad|graphd|storaged|all>
```

> **Note**

- 确保每个服务器中的对应进程都已启动，否则服务将启动失败。
- 当需都启动 graphd、storaged 和 metad 时，可以用 all 代替。
- `/usr/local/nebula` 是 Nebula Graph 的默认安装路径，如果修改过安装路径，请使用实际路径。更多启停服务的内容，请参见管理 Nebula Graph 服务。

4. 检查集群

安装原生 CLI 客户端 Nebula Console，然后连接任何一个已启动 graphd 进程的机器，添加 Storage 主机，然后执行命令 `SHOW HOSTS` 检查集群状态。例如：

```
$ ./nebula-console --addr 192.168.10.111 --port 9669 -u root -p nebula

2021/05/25 01:41:19 [INFO] connection pool is initialized successfully
Welcome to Nebula Graph!

> ADD HOSTS 192.168.10.111:9779, 192.168.10.112:9779, 192.168.10.113:9779, 192.168.10.114:9779, 192.168.10.115:9779;
> SHOW HOSTS;
+-----------------+------+-----------+----------+--------------+----------------------+------------------------+---------+
| Host            | Port | HTTP port | Status   | Leader count | Leader distribution  | Partition distribution | Version |
```


2022 vesoft Inc.

```
+-----------------+-----+----------+---------+-------------+--------------------+-----------------------+---------+
| "192.168.10.111" | 9779 | 19669    | "ONLINE" | 0           | "No valid partition" | "No valid partition"  | "3.1.0" |
| "192.168.10.112" | 9779 | 19669    | "ONLINE" | 0           | "No valid partition" | "No valid partition"  | "3.1.0" |
| "192.168.10.113" | 9779 | 19669    | "ONLINE" | 0           | "No valid partition" | "No valid partition"  | "3.1.0" |
| "192.168.10.114" | 9779 | 19669    | "ONLINE" | 0           | "No valid partition" | "No valid partition"  | "3.1.0" |
| "192.168.10.115" | 9779 | 19669    | "ONLINE" | 0           | "No valid partition" | "No valid partition"  | "3.1.0" |
+-----------------+-----+----------+---------+-------------+--------------------+-----------------------+---------+
```

最后更新: June 20, 2023

2022 vesoft Inc.

5.2.6 使用生态工具安装 Nebula Graph

用户可以使用以下生态工具安装企业版和社区版的 Nebula Graph：

- 企业版 Nebula Dashboard
- Nebula Operator

安装详情

- 使用企业版 **Nebula Dashboard** 安装 Nebula Graph 的详情，参见创建集群。
- 使用 **Nebula Operator** 安装 Nebula Graph 的详情，参见使用 Kubectl 部署 Nebula Graph 集群 或 使用 Helm 部署 Nebula Graph 集群。

> **Note**
>
> 联系销售（inqury@vesoft.com）获取企业版 Nebula Graph 的安装包。

最后更新: June 20, 2023

2022 vesoft Inc.

5.3 存算合并版 Nebula Graph

存算合并版 Nebula Graph 将存储服务（Meta 和 Storage）和计算服务（Graph）合并至一个进程，用于部署在单台机器上。本文介绍存算合并版 Nebula Graph 的使用场景、安装步骤等。

Danger

存算合并版 Nebula Graph 不用于生产环境。

5.3.1 背景信息

传统的 Nebula Graph 架构由 3 个服务构成，每个服务都有可执行的二进制文件和对应的进程，进程之间通过 RPC 协议进行调用。而在存算合并版 Nebula Graph 中，Nebula Graph 中 3 个服务对应的 3 个进程被合为 1 个进程。

关于 Nebula Graph 的更多信息，参见架构总览。

5.3.2 使用场景

数据规模小，可用性需求不大的场景。例如，受限于机器数量的测试环境或者仅用于验证功能的场景。

5.3.3 使用限制

- 仅支持单副本服务。
- 不支持高可用和可靠性。

5.3.4 环境准备

关于安装存算合并版 Nebula Graph 所需的环境，参见编译 Nebula Graph 源码要求。

5.3.5 安装步骤

目前仅支持使用源码安装存算合并版 Nebula Graph。其安装步骤与多进程的 Nebula Graph 步骤类似，用户只需在使用 **CMake** 生成 **makefile** 文件步骤的命令中添加 `-DENABLE_STANDALONE_VERSION=on`。示例如下：

```
cmake -DCMAKE_INSTALL_PREFIX=/usr/local/nebula -DENABLE_TESTING=OFF -DENABLE_STANDALONE_VERSION=on -DCMAKE_BUILD_TYPE=Release ..
```

有关具体的安装步骤，参见使用源码安装。

用户完成存算合并版 Nebula Graph 后，可以参见连接服务连接 Nebula Graph。

5.3.6 配置文件

存算合并版 Nebula Graph 的配置文件的路径默认为 `/usr/local/nebula/etc`。

用户可执行 `sudo cat nebula-standalone.conf.default` 查看配置文件内容。配置文件参数和描述和多进程的 Nebula Graph 大体一致，除以下参数外：

参数	预设值	说明
`meta_port`	`9559`	Meta 服务的端口号。
`storage_port`	`9779`	Storage 服务的端口号。
`meta_data_path`	`data/meta`	Meta 数据存储路径。

用户可以执行命令查看配置项列表与说明。具体操作，请参见配置管理。

2022 vesoft Inc.

最后更新: June 20, 2023

2022 vesoft Inc.

5.4 设置 Nebula Graph 企业版 License

Nebula Graph 企业版需要用户设置 License 才可以正常启动并使用企业版功能，本文介绍如何设置企业版的 License 文件。

Enterpriseonly

License 是为企业版用户提供的软件授权证书，企业版用户可以发送邮件至 `inquiry@vesoft.com` 申请 License 文件。

5.4.1 注意事项

- 没有设置 License 时，Nebula Graph 企业版无法启动。
- 请勿修改 License 文件，否则会导致 License 失效。
- License 快过期时，请及时发送邮件至 `inquiry@vesoft.com` 申请续期。
- License 的过期缓冲为 14 天：
- 过期前 30 天和过期当天，服务启动时会打印日志进行提醒。
- 过期后仍可继续使用 14 天。
- 过期 14 天后，服务无法启动，并会打印日志进行提醒。

5.4.2 Nebula Graph 企业版 License 说明

License 文件（`nebula.license`）内容示例如下：

```
----------License Content Start----------
{
  "vendor": "vesoft",
  "organization": "doc",
  "issuedDate": "2022-04-06T16:00:00.000Z",
  "expirationDate": "2022-05-31T15:59:59.000Z",
  "product": "nebula_graph",
  "version": ">3.0.0",
  "licenseType": "enterprise",
  "gracePeriod": 14,
  "graphdSpec": {
    "nodes": 3
  },
  "storagedSpec": {
    "nodes": 3
  },
  "clusterCode": "BAIAEAiAQAAG"
}
----------License Content End----------

----------License Key Start----------
cofFcOxxxxxxxxxxxxxhnZgaxrQ==
----------License Key End----------
```


2022 vesoft Inc.

License 文件包含生效时间、过期时间等信息。说明如下。

参数	说明
`vendor`	发放渠道。
`organization`	用户名称。
`issuedDate`	License 生效时间。
`expirationDate`	License 过期时间。
`product`	产品类型。Nebula Graph 的产品类型为 `nebula_graph`。
`version`	版本支持的信息。
`licenseType`	License 类型。包括 `enterprise`、`samll_bussiness`、`pro`、`individual`。预留参数。
`gracePeriod`	证书过期后可继续使用服务的缓冲时间（单位天），超过缓冲期后停止服务。试用版的 License 过期后无缓冲期，默认值为 0。
`graphdSpec`	集群中 Graph 服务的数量限制。Nebula Graph 会实时监测当前活动的 Graph 服务数量，超过限制的 Graph 服务无法连接集群。
`storagedSpec`	集群中 Storage 服务的数量限制。Nebula Graph 会实时监测当前活动的 Storage 服务数量，超过限制的 Storage 服务无法连接集群。
`clusterCode`	用户的硬件信息，也是集群的唯一标识码。试用版的 License 中无此参数。

5.4.3 设置 Nebula Graph 企业版 License

1. 发送邮件至 `inquiry@vesoft.com` 申请 Nebula Graph 企业版安装包。
2. 安装 Nebula Graph 企业版。安装方式与社区版相同，请参见使用 RPM 或 DEB 包安装 Nebula Graph。
3. 发送邮件至 `inquiry@vesoft.com` 申请 License 文件 `nebula.license`。
4. 将 License 文件上传到所有包含 Meta 服务的机器上，路径为每个 Meta 服务安装目录的 `share/resources/` 内。

> **Note**
>
> 周边工具的 License 文件上传位置，请参见具体周边工具的说明文档。

5.4.4 续期 Nebula Graph 企业版 License

1. 发送邮件至 `inqury@vesoft.com` 申请新的 Nebula Graph 企业版 License。
2. 在所有包含 Meta 服务的机器上，路径为每个 Meta 服务安装目录的 `share/resources/` 内，使用新的 License 文件 `nebula.license` 替换旧的 License 文件。
3. 重启 Storage 和 Graph 服务。关于重启操作，参见启动服务。如果用户的 License 的过期时间在到期后的缓冲期内（默认 14 天），则无需重启 Storage 和 Graph 服务。

> **Note**
>
> 当用户的 License 过期时间超过到期后的缓冲期，Graph 和 Storage 服务会自动停止。为了确保服务正常运行，请及时更新 License。

5.4.5 查看 Nebula Graph 企业版 License

- 直接查看 License 文件

2022 vesoft Inc.

可以使用 cat 等命令直接查看 License 文件，例如 `cat share/resources/nebula.license` 。

- 通过 HTTP 接口查看 License 文件

  当 Nebula Graph 正常运行时，可以请求 Meta 服务的 HTTP 接口（默认为19559）获取 License 文件内容。例如 `curl -G "http://192.168.10.101:19559/license"` 。

5.4.6 下一步

启动 Nebula Graph

最后更新: June 20, 2023

2022 vesoft Inc.

5.5 管理 Nebula Graph 服务

Nebula Graph 支持通过脚本或 systemd 管理服务。本文详细介绍这两种方式。

Enterpriseonly

仅企业版支持使用 systemd 管理服务。

Danger

这两种方式互不兼容，选择使用其中一种。

5.5.1 使用脚本管理服务

使用脚本 `nebula.service` 管理服务，包括启动、停止、重启、中止和查看。

Note

`nebula.service` 的默认路径是 `/usr/local/nebula/scripts`，如果修改过安装路径，请使用实际路径。

语法

```
$ sudo /usr/local/nebula/scripts/nebula.service
[-v] [-c <config_file_path>]
<start | stop | restart | kill | status>
<metad | graphd | storaged | all>
```

参数	说明
`-v`	显示详细调试信息。
`-c`	指定配置文件路径，默认路径为 `/usr/local/nebula/etc/`。
`start`	启动服务。
`stop`	停止服务。
`restart`	重启服务。
`kill`	中止服务。
`status`	查看服务状态。
`metad`	管理 Meta 服务。
`graphd`	管理 Graph 服务。
`storaged`	管理 Storage 服务。
`all`	管理所有服务。

5.5.2 使用 systemd 管理服务

为方便使用，Nebula Graph 企业版支持用 systemd 管理服务，通过 `systemctl` 启动、停止、重启和查看服务。

2022 vesoft Inc.

Note

- 安装 Nebula Graph 企业版后，systemd 所需的 `.service` 文件在安装目录的 `etc/unit` 目录内，使用 RPM/DEB 包安装的 Nebula Graph，会自动将这些 `.service` 文件放入 `/usr/lib/systemd/system` 目录内，并且 `ExecStart` 也会根据指定的 Nebula Graph 安装路径进行生成，因此可以直接使用 `systemctl` 命令。
- 对于使用企业版 Dashboard 安装的企业版 Nebula Graph，不支持使用 `systemctl` 管理服务。
- 对于其他方式安装的企业版 Nebula Graph，需要用户手动将 `.service` 文件移动到 `/usr/lib/systemd/system` 目录内，并修改 `.service` 文件内的 `ExecStart` 的文件路径，才可以正常使用 `systemctl` 命令。

语法

```
$ systemctl <start | stop | restart | status > <nebula | nebula-metad | nebula-graphd | nebula-storaged>
```

参数	说明
`start`	启动服务。
`stop`	停止服务。
`restart`	重启服务。
`status`	查看服务状态。
`nebula`	管理所有服务。
`nebula-metad`	管理 Meta 服务。
`nebula-graphd`	管理 Graph 服务。
`nebula-storaged`	管理 Storage 服务。

5.5.3 启动 Nebula Graph 服务

非容器部署

对于非容器部署的 Nebula Graph，执行如下命令启动服务：

```
$ sudo /usr/local/nebula/scripts/nebula.service start all
[INFO] Starting nebula-metad...
[INFO] Done
[INFO] Starting nebula-graphd...
[INFO] Done
[INFO] Starting nebula-storaged...
[INFO] Done
```

或者：

```
$ systemctl start nebula
```

如果需要设置开机自动启动，命令如下：

```
$ systemctl enable nebula
```

容器部署

对于使用 Docker Compose 部署的 Nebula Graph，在 `nebula-docker-compose/` 目录内执行如下命令启动服务：

```
[nebula-docker-compose]$ docker-compose up -d
Building with native build. Learn about native build in Compose here: https://docs.docker.com/go/compose-native-build/
Creating network "nebula-docker-compose_nebula-net" with the default driver
Creating nebula-docker-compose_metad0_1 ... done
Creating nebula-docker-compose_metad2_1 ... done
Creating nebula-docker-compose_metad1_1 ... done
Creating nebula-docker-compose_storaged2_1 ... done
Creating nebula-docker-compose_graphd1_1   ... done
```


2022 vesoft Inc.

```
Creating nebula-docker-compose_storaged1_1 ... done
Creating nebula-docker-compose_storaged0_1 ... done
Creating nebula-docker-compose_graphd2_1   ... done
Creating nebula-docker-compose_graphd_1    ... done
```

5.5.4 停止 Nebula Graph 服务

Danger

请勿使用 `kill -9` 命令强制终止进程，否则可能较小概率出现数据丢失。

非容器部署

执行如下命令停止 Nebula Graph 服务：

```
$ sudo /usr/local/nebula/scripts/nebula.service stop all
[INFO] Stopping nebula-metad...
[INFO] Done
[INFO] Stopping nebula-graphd...
[INFO] Done
[INFO] Stopping nebula-storaged...
[INFO] Done
```

或者：

```
$ systemctl stop nebula
```

容器部署

在 `nebula-docker-compose/` 目录内执行如下命令停止 Nebula Graph 服务：

```
[nebula-docker-compose]$ docker-compose down
Stopping nebula-docker-compose_graphd_1    ... done
Stopping nebula-docker-compose_graphd2_1   ... done
Stopping nebula-docker-compose_storaged0_1 ... done
Stopping nebula-docker-compose_storaged1_1 ... done
Stopping nebula-docker-compose_graphd1_1   ... done
Stopping nebula-docker-compose_storaged2_1 ... done
Stopping nebula-docker-compose_metad1_1    ... done
Stopping nebula-docker-compose_metad2_1    ... done
Stopping nebula-docker-compose_metad0_1    ... done
Removing nebula-docker-compose_graphd_1    ... done
Removing nebula-docker-compose_graphd2_1   ... done
Removing nebula-docker-compose_storaged0_1 ... done
Removing nebula-docker-compose_storaged1_1 ... done
Removing nebula-docker-compose_graphd1_1   ... done
Removing nebula-docker-compose_storaged2_1 ... done
Removing nebula-docker-compose_metad1_1    ... done
Removing nebula-docker-compose_metad2_1    ... done
Removing nebula-docker-compose_metad0_1    ... done
Removing network nebula-docker-compose_nebula-net
```

Note

命令 `docker-compose down -v` 将会删除所有本地 Nebula Graph 的数据。如果使用的是 developing 或 nightly 版本，并且有一些兼容性问题，请尝试这个命令。

5.5.5 查看 Nebula Graph 服务

非容器部署

执行如下命令查看 Nebula Graph 服务状态：

2022 vesoft Inc.

```
$ sudo /usr/local/nebula/scripts/nebula.service status all
```

- 如果返回如下结果，表示 Nebula Graph 服务正常运行。

```
[INFO] nebula-metad(33fd35e): Running as 29020, Listening on 9559
[INFO] nebula-graphd(33fd35e): Running as 29095, Listening on 9669
[WARN] nebula-storaged after v3.0.0 will not start service until it is added to cluster.
[WARN] See Manage Storage hosts:ADD HOSTS in https://docs.nebula-graph.io/
[INFO] nebula-storaged(33fd35e): Running as 29147, Listening on 9779
```

> **Note**
>
> 正常启动 Nebula Graph 后，`nebula-storaged` 进程的端口显示红色。这是因为 `nebula-storaged` 在启动流程中会等待 `nebula-metad` 添加当前 Storage 服务，当前 Storage 服务收到 Ready 信号后才会正式启动服务。从 3.0.0 版本开始，在配置文件中添加的 Storage 节点无法直接读写，配置文件的作用仅仅是将 Storage 节点注册至 Meta 服务中。必须使用 `ADD HOSTS` 命令后，才能正常读写 Storage 节点。更多信息，参见管理 Storage 主机。

- 如果返回类似如下结果，表示 Nebula Graph 服务异常，可以根据异常服务信息进一步排查，或者在 Nebula Graph 社区寻求帮助。

```
[INFO] nebula-metad: Running as 25600, Listening on 9559
[INFO] nebula-graphd: Exited
[INFO] nebula-storaged: Running as 25646, Listening on 9779
```

也可以使用 `systemctl` 命令查看 Nebula Graph 服务状态:

```
$ systemctl status nebula
● nebula.service
   Loaded: loaded (/usr/lib/systemd/system/nebula.service; disabled; vendor preset: disabled)
   Active: active (exited) since 一 2022-03-28 04:13:24 UTC; 1h 47min ago
  Process: 21772 ExecStart=/usr/local/ent-nightly/scripts/nebula.service start all (code=exited, status=0/SUCCESS)
 Main PID: 21772 (code=exited, status=0/SUCCESS)
    Tasks: 325
   Memory: 424.5M
   CGroup: /system.slice/nebula.service
           ├─21789 /usr/local/ent-nightly/bin/nebula-metad --flagfile /usr/local/ent-nightly/etc/nebula-metad.conf
           ├─21827 /usr/local/ent-nightly/bin/nebula-graphd --flagfile /usr/local/ent-nightly/etc/nebula-graphd.conf
           └─21900 /usr/local/ent-nightly/bin/nebula-storaged --flagfile /usr/local/ent-nightly/etc/nebula-storaged.conf

3月 28 04:13:24 xxxxxx systemd[1]: Started nebula.service.
...
```

Nebula Graph 服务由 Meta 服务、Graph 服务和 Storage 服务共同提供，这三种服务的配置文件都保存在安装目录的 `etc` 目录内，默认路径为 `/usr/local/nebula/etc/`，用户可以检查相应的配置文件排查问题。

容器部署

在 `nebula-docker-compose` 目录内执行如下命令查看 Nebula Graph 服务状态:

```
[nebula-docker-compose]$ docker-compose ps
            Name                           Command                State                                        Ports
-------------------------------------------------------------------------------------------------------------------------------------------------------
nebula-docker-compose_graphd1_1     /usr/local/nebula/bin/nebu ...   Up (healthy)   0.0.0.0:49223->19669/tcp, 0.0.0.0:49222->19670/tcp, 0.0.0.0:49224->9669/tcp
nebula-docker-compose_graphd2_1     /usr/local/nebula/bin/nebu ...   Up (healthy)   0.0.0.0:49229->19669/tcp, 0.0.0.0:49228->19670/tcp, 0.0.0.0:49230->9669/tcp
nebula-docker-compose_graphd_1      /usr/local/nebula/bin/nebu ...   Up (healthy)   0.0.0.0:49221->19669/tcp, 0.0.0.0:49220->19670/tcp, 0.0.0.0:9669->9669/tcp
nebula-docker-compose_metad0_1      ./bin/nebula-metad --flagf ...   Up (healthy)   0.0.0.0:49212->19559/tcp, 0.0.0.0:49211->19560/tcp, 0.0.0.0:49213->9559/tcp,
                                                                                    9560/tcp
nebula-docker-compose_metad1_1      ./bin/nebula-metad --flagf ...   Up (healthy)   0.0.0.0:49209->19559/tcp, 0.0.0.0:49208->19560/tcp, 0.0.0.0:49210->9559/tcp,
                                                                                    9560/tcp
nebula-docker-compose_metad2_1      ./bin/nebula-metad --flagf ...   Up (healthy)   0.0.0.0:49206->19559/tcp, 0.0.0.0:49205->19560/tcp, 0.0.0.0:49207->9559/tcp,
                                                                                    9560/tcp
nebula-docker-compose_storaged0_1   ./bin/nebula-storaged --fl ...   Up (healthy)   0.0.0.0:49218->19779/tcp, 0.0.0.0:49217->19780/tcp, 9777/tcp, 9778/tcp,
                                                                                    0.0.0.0:49219->9779/tcp, 9780/tcp
nebula-docker-compose_storaged1_1   ./bin/nebula-storaged --fl ...   Up (healthy)   0.0.0.0:49215->19779/tcp, 0.0.0.0:49214->19780/tcp, 9777/tcp, 9778/tcp,
                                                                                    0.0.0.0:49216->9779/tcp, 9780/tcp
nebula-docker-compose_storaged2_1   ./bin/nebula-storaged --fl ...   Up (healthy)   0.0.0.0:49226->19779/tcp, 0.0.0.0:49225->19780/tcp, 9777/tcp, 9778/tcp,
                                                                                    0.0.0.0:49227->9779/tcp, 9780/tcp
```

如果服务有异常，用户可以先确认异常的容器名称（例如 `nebula-docker-compose_graphd2_1`），

然后执行 `docker ps` 查看对应的 `CONTAINER ID`（示例为 `2a6c56c405f5`）。

```
[nebula-docker-compose]$ docker ps
CONTAINER ID   IMAGE                             COMMAND                  CREATED         STATUS
PORTS                                                                                 NAMES
```


2022 vesoft Inc.

```
2a6c56c405f5   vesoft/nebula-graphd:nightly     "/usr/local/nebula/b..."   36 minutes ago   Up 36 minutes (healthy)   0.0.0.0:49230->9669/tcp, 0.0.0.0:49229->19669/tcp, 0.0.0.0:49228->19670/
tcp                             nebula-docker-compose_graphd2_1
7042e0a8e83d   vesoft/nebula-storaged:nightly   "./bin/nebula-storag..."   36 minutes ago   Up 36 minutes (healthy)   9777-9778/tcp, 9780/tcp, 0.0.0.0:49227->9779/tcp, 0.0.0.0:49226->19779/
tcp, 0.0.0.0:49225->19780/tcp   nebula-docker-compose_storaged2_1
18e3ea63ad65   vesoft/nebula-storaged:nightly   "./bin/nebula-storag..."   36 minutes ago   Up 36 minutes (healthy)   9777-9778/tcp, 9780/tcp, 0.0.0.0:49219->9779/tcp, 0.0.0.0:49218->19779/
tcp, 0.0.0.0:49217->19780/tcp   nebula-docker-compose_storaged0_1
4dcabfe8677a   vesoft/nebula-graphd:nightly     "/usr/local/nebula/b..."   36 minutes ago   Up 36 minutes (healthy)   0.0.0.0:49224->9669/tcp, 0.0.0.0:49223->19669/tcp, 0.0.0.0:49222->19670/
tcp                             nebula-docker-compose_graphd1_1
a74054c6ae25   vesoft/nebula-graphd:nightly     "/usr/local/nebula/b..."   36 minutes ago   Up 36 minutes (healthy)   0.0.0.0:9669->9669/tcp, 0.0.0.0:49221->19669/tcp, 0.0.0.0:49220->19670/
tcp                             nebula-docker-compose_graphd_1
880025a3858c   vesoft/nebula-storaged:nightly   "./bin/nebula-storag..."   36 minutes ago   Up 36 minutes (healthy)   9777-9778/tcp, 9780/tcp, 0.0.0.0:49216->9779/tcp, 0.0.0.0:49215->19779/
tcp, 0.0.0.0:49214->19780/tcp   nebula-docker-compose_storaged1_1
45736a32a23a   vesoft/nebula-metad:nightly      "./bin/nebula-metad ..."   36 minutes ago   Up 36 minutes (healthy)   9560/tcp, 0.0.0.0:49213->9559/tcp, 0.0.0.0:49212->19559/tcp, 0.
0.0.0:49211->19560/tcp          nebula-docker-compose_metad0_1
3b2c90eb073e   vesoft/nebula-metad:nightly      "./bin/nebula-metad ..."   36 minutes ago   Up 36 minutes (healthy)   9560/tcp, 0.0.0.0:49207->9559/tcp, 0.0.0.0:49206->19559/tcp, 0.
0.0.0:49205->19560/tcp          nebula-docker-compose_metad2_1
7bb31b7a5b3f   vesoft/nebula-metad:nightly      "./bin/nebula-metad ..."   36 minutes ago   Up 36 minutes (healthy)   9560/tcp, 0.0.0.0:49210->9559/tcp, 0.0.0.0:49209->19559/tcp, 0.
0.0.0:49208->19560/tcp          nebula-docker-compose_metad1_1
```

最后登录容器排查问题

```
[nebula-docker-compose]$ docker exec -it 2a6c56c405f5 bash
[root@2a6c56c405f5 nebula]#
```

5.5.6 下一步

连接 Nebula Graph

最后更新: June 20, 2023

2022 vesoft Inc.

5.6 连接 Nebula Graph 服务

本文介绍如何使用原生命令行客户端 Nebula Console 连接 Nebula Graph。

Caution

首次连接到 Nebula Graph 后，必须先注册 Storage 服务，才能正常查询数据。

Nebula Graph 支持多种类型的客户端，包括命令行客户端、可视化界面客户端和流行编程语言客户端。详情参见客户端列表。

5.6.1 前提条件

- Nebula Graph 服务已启动。
- 运行 Nebula Console 的机器和运行 Nebula Graph 的服务器网络互通。
- Nebula Console 的版本兼容 Nebula Graph 的版本。

Note

版本相同的 Nebula Console 和 Nebula Graph 兼容程度最高，版本不同的 Nebula Console 连接 Nebula Graph 时，可能会有兼容问题，或者无法连接并报错 `incompatible version between client and server`。

5.6.2 操作步骤

1. 在 Nebula Console 下载页面，确认需要的版本，单击 **Assets**。

   Note

   建议选择最新版本。

2. 在 **Assets** 区域找到机器运行所需的二进制文件，下载文件到机器上。
3. （可选）为方便使用，重命名文件为 `nebula-console`。

   Note

   在 Windows 系统中，请重命名为 `nebula-console.exe`。

4. 在运行 Nebula Console 的机器上执行如下命令，为用户授予 nebula-console 文件的执行权限。

   Note

   Windows 系统请跳过此步骤。

```
$ chmod 111 nebula-console
```

5. 在命令行界面中，切换工作目录至 nebula-console 文件所在目录。

2022 vesoft Inc.

6. 执行如下命令连接 Nebula Graph。

- Linux 或 macOS

```
$ ./nebula-console -addr <ip> -port <port> -u <username> -p <password>
  [-t 120] [-e "nGQL_statement" | -f filename.nGQL]
```

- Windows

```
> nebula-console.exe -addr <ip> -port <port> -u <username> -p <password>
  [-t 120] [-e "nGQL_statement" | -f filename.nGQL]
```

参数说明如下。

参数	说明
`-h/-help`	显示帮助菜单。
`-addr/-address`	设置要连接的 Graph 服务的 IP 地址。默认地址为 127.0.0.1。
`-P/-port`	设置要连接的 Graph 服务的端口。默认端口为 9669。
`-u/-user`	设置 Nebula Graph 账号的用户名。未启用身份认证时，可以使用任意已存在的用户名（默认为 `root`）。
`-p/-password`	设置用户名对应的密码。未启用身份认证时，密码可以填写任意字符。
`-t/-timeout`	设置整数类型的连接超时时间。单位为秒，默认值为 120。
`-e/-eval`	设置字符串类型的 nGQL 语句。连接成功后会执行一次该语句并返回结果，然后自动断开连接。
`-f/-file`	设置存储 nGQL 语句的文件的路径。连接成功后会执行该文件内的 nGQL 语句并返回结果，执行完毕后自动断开连接。
`-enable_ssl`	连接 Nebula Graph 时使用 SSL 加密。
`-ssl_root_ca_path`	指定 CA 证书的存储路径。
`-ssl_cert_path`	指定 CRT 证书的存储路径。
`-ssl_private_key_path`	指定私钥文件的存储路径。

更多参数参见项目仓库。

最后更新: June 20, 2023

2022 vesoft Inc.

5.7 管理 Storage 主机

从 3.0.0 版本开始，在配置文件中添加的 Storage 主机无法直接读写，配置文件的作用仅仅是将 Storage 主机注册至 Meta 服务中。必须使用 `ADD HOSTS` 命令后，才能正常读写 Storage 主机。

5.7.1 增加 Storage 主机

向集群中增加 Storage 主机。

```
ADD HOSTS <ip>:<port> [,<ip>:<port> ...];
ADD HOSTS "<hostname>":<port> [,"<hostname>":<port> ...];
```

Note

- 增加 Storage 主机在下一个心跳周期之后才能生效，为确保数据同步，请等待 2 个心跳周期（20 秒），然后执行 `SHOW HOSTS` 查看是否在线。
- IP地址和端口请和配置文件中的设置保持一致，例如单机部署的默认为 `127.0.0.1:9779`。
- 使用域名时，需要用引号包裹，例如 `ADD HOSTS "foo-bar":9779`。

5.7.2 删除 Storage 主机

从集群中删除 Storage 主机。

Note

无法直接删除正在使用的 Storage 主机，需要先删除关联的图空间，才能删除 Storage 主机。

```
DROP HOSTS <ip>:<port> [,<ip>:<port> ...];
DROP HOSTS "<hostname>":<port> [,"<hostname>":<port> ...];
```

最后更新: June 20, 2023

2022 vesoft Inc.

5.8 升级版本

5.8.1 升级 Nebula Graph 2.x 至 3.1.0 版本

本文以 Nebula Graph 2.6.1 版本升级到 3.1.0 版本为例，介绍 Nebula Graph 2.x 版本升级到 3.x 版本的方法。

适用版本

本文适用于将 Nebula Graph 从 2.0.0 及之后的 2.x 版本升级到 3.1.0 版本。不适用于 2.0.0 之前的历史版本（含 1.x 版本）。如需升级历史版本，将其根据最新的 2.x 版本文档升级到最新的 2.x 版本，然后根据本文的说明升级到 3.x 版本。

Caution

如需从 2.0.0 之前的版本（含 1.x 版本）升级到 3.1.0，还需找到 3.1.0 版本文件中 `share/resources` 目录下的 `date_time_zonespec.csv` 文件，将其复制到 Nebula Graph 安装路径下的相同目录内。也可从 GitHub 下载该文件。

升级限制

- 不支持轮转热升级，需完全停止整个集群服务。
- 未提供升级脚本，需手动在每台服务器上依次执行。
- 不支持基于 Docker 容器（包括 Docker Swarm、Docker Compose、K8s）的升级。
- 必须在原服务器上原地升级，不能修改原机器的 IP 地址、配置文件，不可更改集群拓扑。
- 硬盘空间要求：各机器硬盘剩余空间都需要是原数据目录的二倍。其中一倍空间用于容纳手动备份的数据，另一倍空间用于容纳复制到 `dst_db_path` 的 WAL，以及原有数据中新增的用于支持无 tag 点的 key。
- 已知会造成数据丢失的 4 种场景，和 alter schema 以及 default value 相关，参见 github known issues。
- 数据目录不要使用软连接切换，避免失效。
- 部分升级操作需要有 sudo 权限。

升级影响

- 数据膨胀

  Nebula Graph 3.x 版本扩展了原有的数据格式，每个点多出一个 key，所以升级后数据会占用更大的空间。

2022 vesoft Inc.

新增 key 的格式为： Type 字段（1 字节）+ Partition ID 字段（3 字节）+ VID（大小根据类型而定）。key 的 value 为空。多占用的空间可以根据点的数量和 VID 的数据类型计算。例如，数据集中有 1 亿个点，且 VID 为 INT64，则升级后这个 key 会占用 1 亿 * （1 + 3 + 8）= 12 亿字节，约等于 1.2 GB。

- 客户端兼容

  升级后旧版本客户端将无法连接 Nebula Graph，需将所有客户端都升级到兼容 Nebula Graph 3.1.0 的版本。

- 配置变化

  少数配置参数发生改变，详情参考版本发布说明和参数文档。

- 语法兼容

  nGQL 语法有部分不兼容：

- 禁用 `YIELD` 子句返回自定义变量。
- `FETCH` 、`GO` 、`LOOKUP` 、`FIND PATH` 、`GET SUBGRAPH` 语句中必须添加 `YIELD` 子句。
- MATCH 语句中获取点属性时，必须指定 Tag，例如从 `return v.name` 变为 `return v.player.name` 。

Caution

可能存在其它暂未发现的影响，建议升级前详细查看版本发布说明和产品手册，并密切关注论坛与 GitHub 的最新动态。

升级准备

- 根据操作系统和架构下载 Nebula Graph 3.1.0 版本的 TAR 文件并解压，升级过程中需要其中的二进制文件。TAR 包下载地址参见 Download 页面。

  Note

  编译源码或者下载RPM/DEB包也可以获取新版二进制文件。

- 根据 Storage 和 Meta 服务配置中 `data_path` 参数的值找到数据文件的位置，并备份数据。默认路径为 `nebula/data/storage` 和 `nebula/data/meta` 。

  Danger

  升级时不会自动备份原有数据。务必手动备份数据，防止丢失。

- 备份配置文件。

- 统计所有图空间升级前的数据量，供升级后比较。统计方法如下：

a. 运行 `SUBMIT JOB STATS` 。

b. 运行 `SHOW JOBS` 并记录返回结果。

升级步骤

1. 停止所有 Nebula Graph 服务。

```
<nebula_install_path>/scripts/nebula.service stop all
```

`nebula_install_path` 代表 Nebula Graph 的安装目录。

2022 vesoft Inc.

`storaged` 进程 flush 数据要等待约 1 分钟。运行命令后可继续运行 `nebula.service status all` 命令以确认所有服务都已停止。启动和停止服务的详细说明参见管理服务。

> **Note**
>
> 如果超过 20 分钟不能停止服务，放弃本次升级，在论坛或 GitHub 提问。

2. 在升级准备中解压 TAR 包的目的路径下，用此处 `bin` 目录中的新版二进制文件替换 Nebula Graph 安装路径下 `bin` 目录中的旧版二进制文件。

> **Note**
>
> 每台部署了 Nebula Graph 服务的机器上都要更新相应服务的二进制文件。

3. 编辑所有 Graph 服务的配置文件，修改以下参数以适应新版本的取值范围。如参数值已在规定范围内，忽略该步骤。

- 为 `session_idle_timeout_secs` 参数设置一个在 [1,604800] 区间的值，推荐值为 28800。
- 为 `client_idle_timeout_secs` 参数设置一个在 [1,604800] 区间的值，推荐值为 28800。

这些参数在 2.x 版本中的默认值不在新版本的取值范围内，如不修改会升级失败。详细参数说明参见Graph 服务配置。

4. 启动所有 Meta 服务。

```
<nebula_install_path>/scripts/nebula-metad.service start
```

启动后，Meta 服务选举 leader。该过程耗时数秒。

启动后可以任意启动一个 Graph 服务节点，使用 Nebula Graph 连接该节点并运行 `SHOW HOSTS meta` 和 `SHOW META LEADER`，如果能够正常返回 Meta 节点的状态，则 Meta 服务启动成功。

> **Note**
>
> 如果启动异常，放弃本次升级，并在论坛或 GitHub 提问。

5. 使用 `bin` 目录下的新版 db_upgrader 文件升级数据格式。

> **Danger**
>
> 本步骤不会备份 Storage 服务中保存的数据。升级数据格式前，务必按照本文升级准备部分的说明备份数据。

命令语法：

```
<nebula_install_path>/bin/db_upgrader \
--src_db_path=<old_storage_data_path> \
--dst_db_path=<data_backup_path> \
--upgrade_meta_server=<meta_server_ip>:<port>[, <meta_server_ip>:<port> ...] \
--upgrade_version=2:3
```

- `old_storage_data_path` 代表数据的存储路径，由 Storage 服务配置文件中的 `data_path` 参数定义。
- `data_backup_path` 代表自定义的数据备份路径。当前版本该设置不生效，升级数据格式时不会将原有数据备份到任何路径。
- `meta_server_ip` 和 `port` 分别代表 Meta 服务各节点的 IP 地址和端口号。
- `2:3` 代表从 Nebula Graph 2.x 版本升级到 3.x 版本。

本文示例：

```
<nebula_install_path>/bin/db_upgrader \
--src_db_path=/usr/local/nebula/data/storage \
--dst_db_path=/home/vesoft/nebula/data-backup \
--upgrade_meta_server=192.168.8.132:9559 \
--upgrade_version=2:3
```


2022 vesoft Inc.

> **Note**
>
> 如果出现异常，放弃本次升级，并在论坛或 GitHub 提问。

6. 启动所有 Graph 和 Storage 服务。

> **Note**
>
> 如果启动异常，放弃本次升级，并在论坛或 GitHub 提问。

7. 连接新版 Nebula Graph，验证服务是否可用、数据是否正常。连接方法参见连接服务。

   目前尚无有效方式判断升级是否完全成功，可用于测试的参考命令如下：

```
nebula> SHOW HOSTS;
nebula> SHOW HOSTS storage;
nebula> SHOW SPACES;
nebula> USE <space_name>
nebula> SHOW PARTS;
nebula> SUBMIT JOB STATS;
nebula> SHOW STATS;
nebula> MATCH (v) RETURN v LIMIT 5;
```

   也可根据 3.1.0 版本的新功能测试，新功能列表参见发布说明。

升级失败回滚

如果升级失败，停止新版本的所有服务，从备份中恢复配置文件和二进制文件，启动历史版本的服务。

所有周边客户端也切换为旧版。

FAQ

Q：升级过程中是否可以通过客户端写入数据?

A：不可以。升级过程中需要停止所有服务。

Q：如果某台机器只有 Graph 服务，没有 Storage 服务，如何升级?

A：只需要升级 Graph 服务对应的二进制文件和配置文件。

Q：操作报错 `Permission denied`。

A：部分命令需要有 sudo 权限。

Q：是否有工具或者办法验证新旧版本数据是否一致?

A：没有。如果只是检查数据量，可以在升级完成后再次运行 `SUBMIT JOB STATS` 和 `SHOW STATS` 统计数据量，并与升级之前做对比。

Q: Storage `OFFLINE` 并且 `Leader count` 是 `0` 怎么处理?

A：运行以下命令手动添加 Storage 主机：

```
ADD HOSTS <ip>:<port>[, <ip>:<port> ...];
```

例如：

```
ADD HOSTS 192.168.10.100:9779, 192.168.10.101:9779, 192.168.10.102:9779;
```

如果有多个 Meta 服务节点，手动 `ADD HOSTS` 之后，部分 Storage 节点需等待数个心跳（`heartbeat_interval_secs`）的时间才能正常连接到集群。

如果添加 Storage 主机后问题仍然存在，在论坛或 GitHub 提问。

Q：为什么升级后用 `SHOW JOBS` 查询到的 Job 的 ID 与升级前一样，但 Job 名称等信息不同了?

2022 vesoft Inc.

A： Nebula Graph 2.5.0 版本调整了 Job 的定义，详情参见 Pull request。如果是从 2.5.0 之前的版本升级，会出现该问题。

Q: 有哪些语法不兼容？

A: 参见Release Note Incompatibility 部分。

最后更新: June 20, 2023

2022 vesoft Inc.

5.8.2 升级 Nebula Graph v3.0.x 至 v3.1.0

Nebula Graph v3.0.x 升级至 v3.1.0，只需要使用 v3.1.0的 RPM/DEB 包进行升级操作即可，或者编译 v3.1.0 之后重新安装。

RPM/DEB 包升级步骤

1. 下载 RPM/DEB 包。
2. 停止所有 Nebula Graph 服务。详情请参见管理 Nebula Graph 服务。建议更新前备份配置文件。
3. 执行如下命令升级：

- RPM 包

```
$ sudo rpm -Uvh <package_name>
```

若安装时指定路径，那么升级时也需要指定路径

```
$ sudo rpm  -Uvh --prefix=<installation_path> <package_name>
```

- DEB 包

```
$ sudo dpkg -i <package_name>
```

4. 在每台服务器上启动所需的服务。详情请参见管理 Nebula Graph 服务。

编译新版本源码升级步骤

1. 备份旧版本的配置文件。配置文件保存在 Nebula Graph 安装路径的 `etc` 目录内。
2. 更新仓库并编译源码。详情请参见使用源码安装 Nebula Graph。

> **Note**
>
> 编译时注意设置安装路径，和旧版本的安装路径保持一致。

Docker Compose 部署升级步骤

1. 修改目录 `nebula-docker-compose` 内的文件 `docker-compose.yaml`，将 `image` 后的所有版本都修改为 `release-3.1`。
2. 在目录 `nebula-docker-compose` 内执行命令 `docker-compose pull`，更新所有服务的镜像版本。
3. 执行命令 `docker-compose down` 停止 Nebula Graph 服务。
4. 执行命令 `docker-compose up -d` 启动 Nebula Graph 服务。

最后更新: June 20, 2023

2022 vesoft Inc.

5.9 卸载 Nebula Graph

本文介绍如何卸载 Nebula Graph。

Caution

如果需要重新部署 Nebula Graph，请务必完全卸载后再重新部署，否则可能会出现问题，包括 Meta 不一致等。

5.9.1 前提条件

停止 Nebula Graph 服务。详情参见管理 Nebula Graph 服务。

5.9.2 步骤 1：删除数据和元数据文件

如果在配置文件内修改了数据文件的路径，可能会导致安装路径和数据文件保存路径不一致，因此需要查看配置文件，确认数据文件保存路径，然后手动删除数据文件目录。

Note

如果是集群架构，需要删除所有 Storage 和 Meta 服务节点的数据文件。

1. 检查 Storage 服务的 disk 配置。例如：

```
########## Disk ##########
# Root data path. Split by comma. e.g. --data_path=/disk1/path1/,/disk2/path2/
# One path per Rocksdb instance.
--data_path=/nebula/data/storage
```

2. 检查 metad 服务的配置文件，找到对应元数据目录。
3. 删除以上数据和元数据目录。

5.9.3 步骤 2：卸载安装目录

Note

删除整个安装目录，包括 `cluster.id` 文件。

安装路径为参数 `--prefix` 指定的路径。默认路径为 `/usr/local/nebula`。

卸载编译安装的 Nebula Graph

找到 Nebula Graph 的安装目录，删除整个安装目录。

卸载 RPM 包安装的 Nebula Graph

1. 使用如下命令查看 Nebula Graph 版本。

```
$ rpm -qa | grep "nebula"
```

返回类似如下结果。

```
nebula-graph-3.1.0-1.x86_64
```


2022 vesoft Inc.

2. 使用如下命令卸载 Nebula Graph。

```
sudo rpm -e <nebula_version>
```

例如：

```
sudo rpm -e nebula-graph-3.1.0-1.x86_64
```

3. 删除安装目录。

卸载 DEB 包安装的 Nebula Graph

1. 使用如下命令查看 Nebula Graph 版本。

```
$ dpkg -l | grep "nebula"
```

返回类似如下结果。

```
ii  nebula-graph  3.1.0  amd64     Nebula Package built using CMake
```

2. 使用如下命令卸载 Nebula Graph。

```
sudo dpkg -r <nebula_version>
```

例如：

```
sudo dpkg -r nebula-graph
```

3. 删除安装目录。

卸载 Docker Compose 部署的 Nebula Graph

1. 在目录 `nebula-docker-compose` 内执行如下命令停止 Nebula Graph 服务。

```
docker-compose down -v
```

2. 删除目录 `nebula-docker-compose` 。

最后更新: June 20, 2023

2022 vesoft Inc.

6. 配置与日志

6.1 配置

6.1.1 配置管理

Nebula Graph 基于 gflags 库打造了系统配置，多数配置项都是其中的 flags。Nebula Graph 服务启动时，默认会从配置文件中获取配置信息。对于文件中没有的配置项，系统使用默认值。

Enterpriseonly

性能、参数、查询语句的调优方式及服务仅在企业版提供。

Note

- 由于配置项多且可能随着 Nebula Graph 的开发发生变化，文档不会介绍所有配置项。按下文说明可在命令行获取配置项的详细说明。
- 不建议修改文档未介绍的配置项，除非已经熟悉源代码并完全了解配置项的作用。

历史版本兼容性

1.x 版本的文档提供了使用 CONFIGS 命令修改缓存中配置的方法，但在生产环境中使用该方法容易导致集群配置与本地配置文件不一致。因此，自 2.x 版本开始文档中将不再介绍 CONFIGS 命令的使用方法。

查看配置项列表与说明

使用以下命令获取二进制文件对应服务的所有配置项信息：

```
<binary> --help
```

例如：

```
# 获取 Meta 配置项的帮助信息
$ /usr/local/nebula/bin/nebula-metad  --help

# 获取 Graph 配置项的帮助信息
$ /usr/local/nebula/bin/nebula-graphd --help

# 获取 Storage 配置项的帮助信息
$ /usr/local/nebula/bin/nebula-storaged --help
```

以上示例使用了二进制文件的默认存储路径 `/usr/local/nebula/bin/`。如果修改了 Nebula Graph 安装路径，使用实际路径查询配置项信息。

查看运行配置

使用 `curl` 命令获取运行中的配置项的值，即 Nebula Graph 的运行配置。

例如：

```
# 获取 Meta 服务的运行配置
curl 127.0.0.1:19559/flags

# 获取 Graph 服务的运行配置
curl 127.0.0.1:19669/flags

# 获取 Storage 服务的运行配置
curl 127.0.0.1:19779/flags
```


2022 vesoft Inc.

Note

实际环境中需使用真实的主机 IP 地址取代以上示例中的 `127.0.0.1`。

配置文件简介

源码、RPM/DEB、TAR 包集群的配置文件

Nebula Graph 为每个服务都提供了两份初始配置文件 `<service_name>.conf.default` 和 `<service_name>.conf.production`，方便用户在不同场景中使用。使用源码和 RPM/DEB 安装集群的配置文件的默认路径为 `/usr/local/nebula/etc/`；使用 TAR 包安装集群的配置文件路径为 `<install_path>/<tar_package_directory>/etc` TAR 包的安装路径。

初始配置文件中的配置值仅供参考，使用时可根据实际需求调整。如需使用初始配置文件，从上述两个文件选择其一，删除后缀 `.default` 或 `.production` 使其生效。

Caution

为确保服务的可用性，同类服务的配置需保持一致，本机 IP 地址 `local_ip` 除外。例如，一个 Nebula Graph 集群中部署了 3 个 Storage 服务器，3 者除 IP 地址外的其它配置需相同。

下表列出了各服务对应的初始配置文件。

Nebula Graph 服务	初始配置文件	配置说明
Meta	`nebula-metad.conf.default` 和 `nebula-metad.conf.production`	Meta 服务配置
Graph	`nebula-graphd.conf.default` 和 `nebula-graphd.conf.production`	Graph 服务配置
Storage	`nebula-storaged.conf.default` 和 `nebula-storaged.conf.production`	Storage 服务配置

所有服务的初始配置文件中都包含 `local_config` 参数，预设值为 `true`，表示 Nebula Graph 服务会从其配置文件获取配置并启动。

Caution

不建议修改 `local_config` 的值为 `false`。修改后 Nebula Graph 服务启动后会先尝试从 Meta 服务获取缓存的配置信息，可能导致集群配置不一致，造成未知风险。

DOCKER COMPOSE 集群的配置文件

对于使用 Docker Compose 创建的集群，集群的配置文件的默认路径为 `<install_path>/nebula-docker-compose/docker-compose.yaml`。配置文件中的 `command` 下面的参数为各服务的启动参数。

NEBULA OPERATOR 集群的配置文件

对于通过 Nebula Operator 使用 Kubectl 方式创建的集群，集群的配置文件的路径为用户创建集群 YAML 文件的路径。用户可通过配置文件中的 `spec.{graphd|storaged|metad}.config` 参数修改集群各个服务的相关配置。

Note

通过 Helm 部署的集群，暂不支持修改集群服务的相关配置。

关于 Nebula Graph 的配置相关的更多信息，参见 Nebula Config。

2022 vesoft Inc.

修改配置

默认情况下，所有 Nebula Graph 服务从配置文件获取配置。用户可以按照以下步骤修改配置并使其生效。

- 针对使用源码、RPM/DEB、TAR 包安装的集群:

a. 使用文本编辑器修改目标服务的配置文件并保存。

b. 选择合适的时间重启所有 Nebula Graph 服务使修改生效。

- 针对使用 Docker Compose 安装的集群:

a. 在文件 `<install_path>/nebula-docker-compose/docker-compose.yaml` 中，修改服务配置。

b. 在目录 `nebula-docker-compose` 内执行命令 `docker-compose up -d` 重启涉及配置变化的服务。

- 针对使用 Kubectl 方式创建的集群:

  具体操作，参见自定义集群的配置参数。

最后更新: June 20, 2023

2022 vesoft Inc.

6.1.2 Meta 服务配置

Meta 服务提供了两份初始配置文件 `nebula-metad.conf.default` 和 `nebula-metad.conf.production`，方便在不同场景中使用。文件的默认路径为 `/usr/local/nebula/etc/`。

Caution

- 不建议修改 `local_config` 的值为 `false`。修改后 Nebula Graph 服务启动后会先尝试从 Meta 服务获取缓存的配置信息，可能导致集群配置不一致，造成未知风险。
- 不建议修改文档未介绍的配置项，除非已经熟悉源代码并清楚了解配置项作用。

配置文件使用方式

如需使用初始配置文件，从上述两个文件选择其一，删除后缀 `.default` 或 `.production`，Meta 服务才能将其识别为配置文件并从中获取配置信息。

配置文件参数值说明

配置文件内没有设置某个参数表示参数使用的是默认值。文件内只预设了部分参数的值，而且两份初始配置文件内的参数值也略有不同，本文的预设值以 `nebula-metad.conf.default` 为准。

如需查看所有的参数及其当前值，参见配置管理。

basics 配置

名称	预设值	说明
`daemonize`	`true`	是否启动守护进程。
`pid_file`	`pids/nebula-metad.pid`	记录进程 ID 的文件。
`timezone_name`	-	指定 Nebula Graph 的时区。初始配置文件中未设置该参数，如需使用请手动添加。系统默认值为 `UTC+00:00:00`。格式请参见 Specifying the Time Zone with TZ。例如，东八区的设置方式为 `--timezone_name=UTC+08:00`。
`license_path`	`share/resources/nebula.license`	企业版 License 路径。企业版需要用户设置 License 才可以正常启动并使用企业版功能。仅企业版 Nebula Graph 需要设置本参数，其他周边工具如何设置 License，请参见相应周边工具的部署文档。

Note

- 在插入时间类型的属性值时，Nebula Graph 会根据 `timezone_name` 设置的时区将该时间值（TIMESTAMP 类型例外）转换成相应的 UTC 时间，因此在查询中返回的时间类型属性值为 UTC 时间。
- `timezone_name` 参数只用于转换 Nebula Graph 中存储的数据，Nebula Graph 进程中其它时区相关数据，例如日志打印的时间等，仍然使用主机系统默认的时区。

2022 vesoft Inc.

logging 配置

名称	预设值	说明
`log_dir`	`logs`	存放 Meta 服务日志的目录，建议和数据保存在不同硬盘。
`minloglevel`	`0`	最小日志级别，即不会记录低于这个级别的日志。可选值为 `0`（INFO）、`1`（WARNING）、`2`（ERROR）、`3`（FATAL）。建议在调试时设置为 `0`，生产环境中设置为 `1`。如果设置为 `4`，Nebula Graph 不会记录任何日志。
`v`	`0`	日志详细级别，值越大，日志记录越详细。可选值为 `0`、`1`、`2`、`3`。
`logbufsecs`	`0`	缓冲日志的最大时间，超时后输出到日志文件。`0` 表示实时输出。单位：秒。
`redirect_stdout`	`true`	是否将标准输出和标准错误重定向到单独的输出文件。
`stdout_log_file`	`metad-stdout.log`	标准输出日志文件名称。
`stderr_log_file`	`metad-stderr.log`	标准错误日志文件名称。
`stderrthreshold`	`2`	要复制到标准错误中的最小日志级别（`minloglevel`）。
`timestamp_in_logfile_name`	`true`	日志文件名称中是否包含时间戳。`true` 表示包含，`false` 表示不包含。

networking 配置

名称	预设值	说明
`meta_server_addrs`	`127.0.0.1:9559`	全部 Meta 服务的 IP 地址和端口。多个 Meta 服务用英文逗号（,）分隔。
`local_ip`	`127.0.0.1`	Meta 服务的本地 IP 地址。本地 IP 地址用于识别 nebula-metad 进程，如果是分布式集群或需要远程访问，请修改为对应地址。
`port`	`9559`	Meta 服务的 RPC 守护进程监听端口。Meta 服务对外端口为 `9559`，对内端口为 `对外端口 +1`，即 `9560`，Nebula Graph 使用内部端口进行多副本间的交互。
`ws_ip`	`0.0.0.0`	HTTP 服务的 IP 地址。
`ws_http_port`	`19559`	HTTP 服务的端口。
`ws_storage_http_port`	`19779`	HTTP 协议监听 Storage 服务的端口，需要和 Storage 服务配置文件中的 `ws_http_port` 保持一致。
`heartbeat_interval_secs`	`10`	默认心跳间隔。请确保所有服务的 `heartbeat_interval_secs` 取值相同，否则会导致系统无法正常工作。单位：秒。

Caution

必须在配置文件中使用真实的 IP 地址。否则某些情况下 `127.0.0.1/0.0.0.0` 无法正确解析。

storage 配置

名称	预设值	说明
`data_path`	`data/meta`	meta 数据存储路径。

2022 vesoft Inc.

misc 配置

名称	预设值	说明
`default_parts_num`	`100`	创建图空间时的默认分片数量。
`default_replica_factor`	`1`	创建图空间时的默认副本数量。

rocksdb options 配置

名称	预设值	说明
`rocksdb_wal_sync`	`true`	是否同步 RocksDB 的 WAL 日志。

最后更新: June 20, 2023

2022 vesoft Inc.

6.1.3 Graph 服务配置

Graph 服务提供了两份初始配置文件 `nebula-graphd.conf.default` 和 `nebula-graphd.conf.production`，方便在不同场景中使用。文件的默认路径为 `/usr/local/nebula/etc/`。

Caution

- 不建议修改 `local_config` 的值为 `false`。修改后 Nebula Graph 服务启动后会先尝试从 Meta 服务获取缓存的配置信息，可能导致集群配置不一致，造成未知风险。
- 不建议修改文档未介绍的配置项，除非已经熟悉源代码并完全了解配置项的作用。

配置文件使用方式

如需使用初始配置文件，从上述两个文件选择其一，删除后缀 `.default` 或 `.production`，Graph 服务才能将其识别为配置文件并从中获取配置信息。

配置文件参数值说明

配置文件内没有设置某个参数表示参数使用的是默认值。文件内只预设了部分参数的值，而且两份初始配置文件内的参数值也略有不同，本文的预设值以 `nebula-graphd.conf.default` 为准。

如需查看所有的参数及其当前值，参见配置管理。

basics 配置

名称	预设值	说明
`daemonize`	`true`	是否启动守护进程。
`pid_file`	`pids/nebula-graphd.pid`	记录进程 ID 的文件。
`enable_optimizer`	`true`	是否启用优化器。
`timezone_name`	-	指定 Nebula Graph 的时区。初始配置文件中未设置该参数，使用需手动添加。系统默认值为 `UTC+00:00:00`。格式请参见 Specifying the Time Zone with TZ。例如，东八区的设置方式为 `--timezone_name=UTC+08:00`。
`local_config`	`true`	是否从配置文件获取配置信息。

Note

- 在插入时间类型 的属性值时，Nebula Graph 会根据 `timezone_name` 设置的时区将该时间值（TIMESTAMP 类型例外）转换成相应的 UTC 时间，因此在查询中返回的时间类型属性值为 UTC 时间。
- `timezone_name` 参数只用于转换 Nebula Graph 中存储的数据，Nebula Graph 进程中其它时区相关数据，例如日志打印的时间等，仍然使用主机系统默认的时区。

2022 vesoft Inc.

logging 配置

名称	预设值	说明
`log_dir`	`logs`	存放 Graph 服务日志的目录，建议和数据保存在不同硬盘。
`minloglevel`	`0`	最小日志级别，即不会记录低于这个级别的日志。可选值为 `0`（INFO）、`1`（WARNING）、`2`（ERROR）、`3`（FATAL）。建议在调试时设置为 `0`，生产环境中设置为 `1`。如果设置为 `4`，Nebula Graph 不会记录任何日志。
`v`	`0`	日志详细级别，值越大，日志记录越详细。可选值为 `0`、`1`、`2`、`3`。
`logbufsecs`	`0`	缓冲日志的最大时间，超时后输出到日志文件。`0` 表示实时输出。单位：秒。
`redirect_stdout`	`true`	是否将标准输出和标准错误重定向到单独的输出文件。
`stdout_log_file`	`graphd-stdout.log`	标准输出日志文件名称。
`stderr_log_file`	`graphd-stderr.log`	标准错误日志文件名称。
`stderrthreshold`	`2`	要复制到标准错误中的最小日志级别（`minloglevel`）。
`timestamp_in_logfile_name`	`true`	日志文件名称中是否包含时间戳。`true` 表示包含，`false` 表示不包含。

query 配置

名称	预设值	说明
`accept_partial_success`	`false`	是否将部分成功视为错误。此配置仅适用于只读请求，写请求总是将部分成功视为错误。
`session_reclaim_interval_secs`	`10`	将 Session 信息发送给 Meta 服务的间隔。单位：秒。
`max_allowed_query_size`	`4194304`	最大查询语句长度。单位：字节。默认为 `4194304`，即 4MB。

2022 vesoft Inc.

networking 配置

名称	预设值	说明
`meta_server_addrs`	`127.0.0.1:9559`	全部 Meta 服务的 IP 地址和端口。多个 Meta 服务用英文逗号（,）分隔。
`local_ip`	`127.0.0.1`	Graph 服务的本地 IP 地址。本地 IP 地址用于识别 nebula-graphd 进程，如果是分布式集群或需要远程访问，请修改为对应地址。
`listen_netdev`	`any`	监听的网络设备。
`port`	`9669`	Graph 服务的 RPC 守护进程监听端口。
`reuse_port`	`false`	是否启用 SO_REUSEPORT。
`listen_backlog`	`1024`	socket 监听的连接队列最大长度，调整本参数需要同时调整 `net.core.somaxconn` 。
`client_idle_timeout_secs`	`28800`	空闲连接的超时时间。取值范围为 1~604800。默认 8 小时。单位：秒。
`session_idle_timeout_secs`	`28800`	空闲会话的超时时间。取值范围为 1~604800。默认 8 小时。单位：秒。
`num_accept_threads`	`1`	接受传入连接的线程数。
`num_netio_threads`	`0`	网络 IO 线程数。 `0` 表示 CPU 核数。
`num_worker_threads`	`0`	执行用户查询的线程数。 `0` 表示 CPU 核数。
`ws_ip`	`0.0.0.0`	HTTP 服务的 IP 地址。
`ws_http_port`	`19669`	HTTP 服务的端口。
`heartbeat_interval_secs`	`10`	默认心跳间隔。请确保所有服务的 `heartbeat_interval_secs` 取值相同，否则会导致系统无法正常工作。单位：秒。
`storage_client_timeout_ms`	-	Graph 服务与 Storage 服务的 RPC 连接超时时间。初始配置文件中未设置该参数，使用需手动添加。默认值为 `60000` 毫秒。
`ws_meta_http_port`	`19559`	HTTP 协议监听 Meta 服务的端口，需要和 Meta 服务配置文件中的 `ws_http_port` 保持一致。

Caution

必须在配置文件中使用真实的 IP 地址。否则某些情况下 `127.0.0.1/0.0.0.0` 无法正确解析。

charset and collate 配置

名称	预设值	说明
`default_charset`	`utf8`	创建图空间时的默认字符集。
`default_collate`	`utf8_bin`	创建图空间时的默认排序规则。

authorization 配置

名称	预设值	说明
`enable_authorize`	`false`	用户登录时是否进行身份验证。身份验证详情请参见身份验证。
`auth_type`	`password`	用户登录的身份验证方式。取值为 `password` 、 `ldap` 、 `cloud` 。

2022 vesoft Inc.

memory 配置

名称	预设值	说明
system_memory_high_watermark_ratio	0.8	内存高水位报警机制的触发阈值。系统内存占用率高于该值会触发报警机制，Nebula Graph 会停止接受查询。

audit 配置

Enterpriseonly

仅 Nebula Graph 企业版支持审计日志功能。

详细参数说明参见审计日志。

metrics 配置

名称	预设值	说明
enable_space_level_metrics	false	开启后可打开图空间级别的监控，对应的监控指标名称中包含图空间的名称，例如 query_latency_us{space=basketballplayer}.avg.3600 。支持的监控指标可用 curl 命令查看，详细说明参见查询监控指标。

session 配置

名称	预设值	说明
max_sessions_per_ip_per_user	300	相同用户和 IP 地址可以创建的最大会话数。

experimental 配置

名称	预设值	说明
enable_experimental_feature	false	实验性功能开关。可选值为 true 和 false 。当前支持的实验性功能请参见下文。

EXPERIMENTAL 功能说明

名称	说明
TOSS	TOSS（Transaction on Storage Side）功能，用于保证对边进行 INSERT 、 UPDATE 、 UPSERT 或 DELETE 操作的最终一致性（因为逻辑上的一条边对应着硬盘上的两个键值对）。开启后会增加相关操作的时延约 1 倍。

最后更新: June 20, 2023

2022 vesoft Inc.

6.1.4 Storage 服务配置

Storage 服务提供了两份初始配置文件 `nebula-storaged.conf.default` 和 `nebula-storaged.conf.production`，方便在不同场景中使用。文件的默认路径为 `/usr/local/nebula/etc/`。

Caution

- 不建议修改 `local_config` 的值为 `false`。修改配置并重启 Storage 服务，会先尝试从 Meta 服务获取缓存的配置信息，可能导致集群配置不一致，造成未知风险。
- 不建议修改文档未介绍的配置项，除非已经熟悉源代码并完全了解配置项的作用。

配置文件使用方式

如需使用初始配置文件，从上述两个文件选择其一，删除后缀 `.default` 或 `.production`，Storage 服务才能将其识别为配置文件并从中获取配置信息。

配置文件参数值说明

配置文件内没有设置某个参数表示参数使用的是默认值。文件内只预设了部分参数的值，而且两份初始配置文件内的参数值也略有不同，本文的预设值以 `nebula-storaged.conf.default` 文件为准，其中没有的参数则以 `nebula-storaged.conf.production` 文件为准。

Note

Raft Listener 的配置和 Storage 服务配置不同，详情请参见部署 Raft listener。

如需查看所有的参数及其当前值，参见配置管理。

basics 配置

名称	预设值	说明
`daemonize`	`true`	是否启动守护进程。
`pid_file`	`pids/nebula-storaged.pid`	记录进程 ID 的文件。
`timezone_name`	-	指定 Nebula Graph 的时区。初始配置文件中未设置该参数，如需使用请手动添加。系统默认值为 `UTC+00:00:00`。格式请参见 Specifying the Time Zone with TZ。例如，东八区的设置方式为 `--timezone_name=UTC+08:00`。
`local_config`	`true`	是否从配置文件获取配置信息。

Note

- 在插入时间类型的属性值时，Nebula Graph 会根据 `timezone_name` 设置的时区将该时间值（TIMESTAMP 类型例外）转换成相应的 UTC 时间，因此在查询中返回的时间类型属性值为 UTC 时间。
- `timezone_name` 参数只用于转换 Nebula Graph 中存储的数据，Nebula Graph 进程中其它时区相关数据，例如日志打印的时间等，仍然使用主机系统默认的时区。

2022 vesoft Inc.

logging 配置

名称	预设值	说明
`log_dir`	`logs`	存放 Storage 服务日志的目录，建议和数据保存在不同硬盘。
`minloglevel`	`0`	最小日志级别，即不会记录低于这个级别的日志。可选值：`0`（INFO）、`1`（WARNING）、`2`（ERROR）、`3`（FATAL）。建议在调试时设置为 `0`，生产环境中设置为 `1`。如果设置为 `4`，Nebula Graph 不会记录任何日志。
`v`	`0`	日志详细级别，值越大，日志记录越详细。可选值：`0`、`1`、`2`、`3`。
`logbufsecs`	`0`	缓冲日志的最大时间，超时后输出到日志文件。`0` 表示实时输出。单位：秒。
`redirect_stdout`	`true`	是否将标准输出和标准错误重定向到单独的输出文件。
`stdout_log_file`	`storaged-stdout.log`	标准输出日志文件名称。
`stderr_log_file`	`storaged-stderr.log`	标准错误日志文件名称。
`stderrthreshold`	`2`	要复制到标准错误中的最小日志级别（`minloglevel`）。
`timestamp_in_logfile_name`	`true`	日志文件名称中是否包含时间戳。`true` 表示包含，`false` 表示不包含。

networking 配置

名称	预设值	说明
`meta_server_addrs`	`127.0.0.1:9559`	全部 Meta 服务的 IP 地址和端口。多个 Meta 服务用英文逗号（,）分隔。
`local_ip`	`127.0.0.1`	Storage 服务的本地 IP 地址。本地 IP 地址用于识别 nebula-storaged 进程，如果是分布式集群或需要远程访问，请修改为对应地址。
`port`	`9779`	Storage 服务的 RPC 守护进程监听端口。Storage 服务对外端口为 `9779`，对内端口为 `9777`、`9778` 和 `9780`，Nebula Graph 使用内部端口进行多副本间的交互。
`ws_ip`	`0.0.0.0`	HTTP 服务的 IP 地址。
`ws_http_port`	`19779`	HTTP 服务的端口。
`heartbeat_interval_secs`	`10`	默认心跳间隔。请确保所有服务的 `heartbeat_interval_secs` 取值相同，否则会导致系统无法正常工作。单位：秒。

Caution

必须在配置文件中使用真实的 IP 地址。否则某些情况下 `127.0.0.1/0.0.0.0` 无法正确解析。

raft 配置

名称	预设值	说明
`raft_heartbeat_interval_secs`	`30`	Raft 选举超时时间。单位：秒。
`raft_rpc_timeout_ms`	`500`	Raft 客户端的远程过程调用（RPC）超时时间。单位：毫秒。
`wal_ttl`	`14400`	Raft WAL 的有效时间。单位：秒。

2022 vesoft Inc.

disk 配置

名称	预设值	说明
data_path	data/storage	数据存储路径，多个路径用英文逗号（,）分隔。一个 RocksDB 实例对应一个路径。
minimum_reserved_bytes	268435456	每个数据存储路径的剩余空间最小值，低于该值时，可能会导致集群数据写入失败。单位：字节。
rocksdb_batch_size	4096	批量操作的缓存大小。单位：字节。
rocksdb_block_cache	4	BlockBasedTable 的默认块缓存大小。单位：兆字节。
disable_page_cache	false	允许或禁止 Nebula Graph 使用操作系统的页缓存。默认值为 false，表示允许使用 page cache。当值为 true 时，禁止 Nebula Graph 使用 page cache，此时须设置充足的块缓存（block cache）空间。
engine_type	rocksdb	存储引擎类型。
rocksdb_compression	lz4	压缩算法，可选值：no 、snappy 、lz4 、lz4hc 、zlib 、bzip2 、zstd 。
rocksdb_compression_per_level	-	为不同级别设置不同的压缩算法。
enable_rocksdb_statistics	false	是否启用 RocksDB 的数据统计。
rocksdb_stats_level	kExceptHistogramOrTimers	RocksDB 的数据统计级别。可选值：kExceptHistogramOrTimers （禁用计时器统计，跳过柱状图统计）、kExceptTimers （跳过计时器统计）、kExceptDetailedTimers （收集除互斥锁和压缩花费时间之外的所有统计数据）、kExceptTimeForMutex 收集除互斥锁花费时间之外的所有统计数据）、kAll （收集所有统计数据）。
enable_rocksdb_prefix_filtering	true	是否启用 prefix bloom filter，启用时可以提升图遍历速度，但是会增加内存消耗。
enable_rocksdb_whole_key_filtering	false	是否启用 whole key bloom filter。
rocksdb_filtering_prefix_length	12	每个 key 的 prefix 长度。可选值：12 （分片 ID+点 ID）、16 （分片 ID+点 ID+TagID/Edge typeID）。单位：字节。
enable_partitioned_index_filter	-	设置为 true 可以降低 bloom 过滤器占用的内存大小，但是在某些随机寻道（random-seek）的情况下，可能会降低读取性能。

Key-Value separation 配置

名称	预设值	说明
rocksdb_enable_kv_separation	false	是否启用 BlobDB KV 分离存储功能。开启后可以提高查询性能。
rocksdb_kv_separation_threshold	100	RocksDB KV 分离的阈值，在 flush 或 compaction 期间，大于或等于该阈值的值将被写入blob文件。单位：字节。
rocksdb_blob_compression	lz4	BlobDB 的压缩算法，可选值：no 、snappy 、lz4 、lz4hc 、zlib 、bzip2 、zstd 。
rocksdb_enable_blob_garbage_collection	true	在 compaction 过程中是否对 BlobDB 进行垃圾收集。

2022 vesoft Inc.

misc 配置

Caution

下表中的 `snapshot` 与 Nebula Graph 快照是不同的概念，这里的 `snapshot` 指 Raft 同步过程中 leader 上的存量数据。

名称	预设值	说明
`snapshot_part_rate_limit`	`8388608`	Raft leader 向 Raft group 中其它成员同步存量数据时的限速。单位：字节/秒。
`snapshot_batch_size`	`1048576`	Raft leader 向 Raft group 中其它成员同步存量数据时每批发送的数据量。单位：字节。
`rebuild_index_part_rate_limit`	`4194304`	重建索引过程中，Raft leader 向 Raft group 中其它成员同步索引数据时的限速。单位：字节/秒。
`rebuild_index_batch_size`	`1048576`	重建索引过程中，Raft leader 向 Raft group 中其它成员同步索引数据时每批发送的数据量。单位：字节。

rocksdb options 配置

名称	预设值	说明
`rocksdb_db_options`	`{}`	RocksDB database 选项。
`rocksdb_column_family_options`	`{"write_buffer_size":"67108864", "max_write_buffer_number":"4", "max_bytes_for_level_base":"268435456"}`	RocksDB column family 选项。
`rocksdb_block_based_table_options`	`{"block_size":"8192"}`	RocksDB block based table 选项。

rocksdb options 配置的格式为 `{"<option_name>":"<option_value>"}`，多个选项用英文逗号（,）隔开。

`rocksdb_db_options` 和 `rocksdb_column_family_options` 支持的选项如下：

- `rocksdb_db_options`

```
max_total_wal_size
delete_obsolete_files_period_micros
max_background_jobs
stats_dump_period_sec
compaction_readahead_size
writable_file_max_buffer_size
bytes_per_sync
wal_bytes_per_sync
delayed_write_rate
avoid_flush_during_shutdown
max_open_files
stats_persist_period_sec
stats_history_buffer_size
strict_bytes_per_sync
enable_rocksdb_prefix_filtering
enable_rocksdb_whole_key_filtering
rocksdb_filtering_prefix_length
num_compaction_threads
rate_limit
```

- `rocksdb_column_family_options`

```
write_buffer_size
max_write_buffer_number
level0_file_num_compaction_trigger
level0_slowdown_writes_trigger
level0_stop_writes_trigger
target_file_size_base
target_file_size_multiplier
max_bytes_for_level_base
max_bytes_for_level_multiplier
disable_auto_compactions
```


2022 vesoft Inc.

参数的详细说明请参见 RocksDB 官方文档。

storage cache 配置

Enterpriseonly

仅企业版支持 storage cache 配置。

名称	预设值	说明
`enable_storage_cache`	`false`	是否开启 Storage 缓存功能。
`storage_cache_capacity`	`0`	为 Storage 缓存预留的内存大小，取值需要稍大于 `vertex_pool_capacity` 和 `empty_key_pool_capacity` 的总和。单位：兆字节。
`storage_cache_buckets_power`	`20`	bucket 的个数。取值为以 2 为底的对数，取值范围：0~32。例如取值为 20，表示 bucket 个数为 2\(^{20}\)。建议值为 `ceil(log2(cacheEntries * 1.6))` 。`cacheEntries` 为要缓存的条目总数。
`storage_cache_locks_power`	`10`	锁的个数。取值为以 2 为底的对数，取值范围：0~32。例如取值为 10，表示锁个数为 2\(^{10}\)。建议值为 `max(1, storage_cache_buckets_power - 10)` 。
`enable_vertex_pool`	`false`	是否使用点的缓存池。仅在启用 Storage 缓存功能时生效。
`vertex_pool_capacity`	`50`	点缓存池的大小。 单位：兆字节。
`vertex_item_ttl`	`300`	点缓存条目的有效时间。单位：秒。
`enable_empty_key_pool`	`false`	是否使用 empty_key 的缓存池。empty_key 表示用户查询过但是实际不存在的 key。
`empty_key_pool_capacity`	`50`	empty_key 缓存池的大小。 单位：兆字节。
`empty_key_item_ttl`	`300`	empty_key 缓存条目的有效时间。单位：秒。

超级节点处理（出入边数量极多的点）

在每个点出发的查询获取到边时，直接截断。目的是避免超级节点的邻边过多，单个查询占用过多的硬盘和内存。截取前 `max_edge_returned_per_vertex` 个边，多余的边不返回。该参数作用于全局，不用于单个 space。

属性名	默认值	说明
max_edge_returned_per_vertex	2147483647	每个稠密点，最多返回多少条边，多余的边截断不返回。配置文件默认未设置。

数据量大而内存不够时

如果数据量很大但内存不够，则推荐把 storage 配置中的 `enable_partitioned_index_filter` 设置为 `true`；但由于缓存了较少的 RocksDB 索引，性能会受影响。

最后更新: June 20, 2023

2022 vesoft Inc.

6.1.5 Linux 内核配置

本文介绍与 Nebula Graph 相关的 Linux 内核配置，并介绍如何修改配置。

资源控制

ULIMIT 注意事项

命令 `ulimit` 用于为当前 shell 会话设置资源阈值，注意事项如下：

- `ulimit` 所做的更改仅对当前会话或子进程生效。
- 资源的阈值（软阈值）不能超过硬阈值。
- 普通用户不能使用命令调整硬阈值，即使使用 `sudo` 也不能调整。
- 修改系统级别或调整硬性阈值，请编辑文件 `/etc/security/limits.conf` 。这种方式需要重新登录才生效。

ULIMIT -C

`ulimit -c` 用于限制 core 文件的大小，建议设置为 `unlimited` ，命令如下：

```
ulimit -c unlimited
```

ULIMIT -N

`ulimit -n` 用于限制打开文件的数量，建议设置为超过 10 万，例如：

```
ulimit -n 130000
```

内存

VM.SWAPPINESS

`vm.swappiness` 是触发虚拟内存（swap）的空闲内存百分比。值越大，使用 swap 的可能性就越大，建议设置为 0，表示首先删除页缓存。需要注意的是，0 表示尽量不使用 swap。

VM.MIN_FREE_KBYTES

`vm.min_free_kbytes` 用于设置 Linux 内核保留的最小空闲千字节数。如果系统内存足够，建议设置较大值。例如物理内存为 128 GB，可以将 `vm.min_free_kbytes` 设置为 5 GB。如果值太小，会导致系统无法申请足够的连续物理内存。

VM.MAX_MAP_COUNT

`vm.max_map_count` 用于限制单个进程的 VMA（虚拟内存区域）数量。默认值为 `65530` ，对于绝大多数应用程序来说已经足够。如果应用程序因为内存消耗过大而报错，请增大本参数的值。

VM.DIRTY_*

`vm.dirty_*` 是一系列控制系统脏数据缓存的参数。对于写密集型场景，用户可以根据需要进行调整（吞吐量优先或延迟优先），建议使用系统默认值。

TRANSPARENT HUGE PAGE

为了降低延迟，用户必须关闭 THP（transparent huge page）。命令如下：

```
root# echo never > /sys/kernel/mm/transparent_hugepage/enabled
root# echo never > /sys/kernel/mm/transparent_hugepage/defrag
root# swapoff -a && swapon -a
```

为了防止系统重启后该配置失效，可以在GRUB配置文件或 `/etc/rc.local` 中添加相关配置，使系统启动时自动关闭 THP。

2022 vesoft Inc.

网络

NET.IPV4.TCP_SLOW_START_AFTER_IDLE

`net.ipv4.tcp_slow_start_after_idle` 默认值为 1，会导致闲置一段时间后拥塞窗口超时，建议设置为 `0`，尤其适合大带宽高延迟场景。

NET.CORE.SOMAXCONN

`net.core.somaxconn` 用于限制 socket 监听的连接队列数量。默认值为 `128`。对于有大量突发连接的场景，建议设置为不低于 `1024`。

NET.IPV4.TCP_MAX_SYN_BACKLOG

`net.ipv4.tcp_max_syn_backlog` 用于限制处于 SYN_RECV（半连接）状态的 TCP 连接数量。默认值为 `128`。对于有大量突发连接的场景，建议设置为不低于 `1024`。

NET.CORE.NETDEV_MAX_BACKLOG

`net.core.netdev_max_backlog` 用于限制队列中数据包的数量。默认值为 `1000`，建议设置为 `10000` 以上，尤其是万兆网卡。

NET.IPV4.TCP_KEEPALIVE_*

`net.ipv4.tcp_keepalive_*` 是一系列保持 TCP 连接存活的参数。对于使用四层透明负载均衡的应用程序，如果空闲连接异常断开，请增大 `tcp_keepalive_time` 和 `tcp_keepalive_intvl` 的值。

NET.IPV4.TCP_WMEM/RMEM

TCP 套接字发送/接收缓冲池的最小、最大、默认空间。对于大连接，建议设置为 `带宽 (GB) *往返时延 (ms)`。

SCHEDULER

对于 SSD 设备，建议将 `scheduler` 设置为 `noop` 或者 `none`，路径为 `/sys/block/DEV_NAME/queue/scheduler`。

其他参数

KERNEL.CORE_PATTERN

建议设置为 `core`，并且将 `kernel.core_uses_pid` 设置为 `1`。

修改参数

SYSCTL 命令

- `sysctl <conf_name>`

  查看当前参数值。

- `sysctl -w <conf_name>=<value>`

  临时修改参数值，立即生效，重启后恢复原值。

- `sysctl -p [<file_path>]`

  从指定配置文件里加载 Linux 系统参数，默认从 `/etc/sysctl.conf` 加载。

PRLIMIT

命令 `prlimit` 可以获取和设置进程资源的限制，结合 `sudo` 可以修改硬阈值，例如，`prlimit --nofile=140000 --pid=$$` 调整当前进程允许的打开文件的最大数量为 `140000`，立即生效，此命令仅支持 RedHat 7u 或更高版本。

最后更新: June 20, 2023

2022 vesoft Inc.

6.2 日志

6.2.1 运行日志

运行日志通常提供给 DBA 或开发人员查看，当系统出现故障，DBA 或开发人员可以根据运行日志定位问题。

Nebula Graph 默认使用 glog 打印运行日志，使用 gflags 控制日志级别，并在运行时通过 HTTP 接口动态修改日志级别，方便跟踪问题。

运行日志目录

运行日志的默认目录为 `/usr/local/nebula/logs/` 。

如果在 Nebula Graph 运行过程中删除运行日志目录，日志不会继续打印，但是不会影响业务。重启服务可以恢复正常。

配置说明

- `minloglevel`：最小日志级别，即不会记录低于这个级别的日志。可选值为 `0`（INFO）、`1`（WARNING）、`2`（ERROR）、`3`（FATAL）。建议在调试时设置为 `0`，生产环境中设置为 `1`。如果设置为 `4`，Nebula Graph 不会记录任何日志。
- `v`：日志详细级别，值越大，日志记录越详细。可选值为 `0`、`1`、`2`、`3`。

Meta 服务、Graph 服务和 Storage 服务的日志级别可以在各自的配置文件中查看，默认路径为 `/usr/local/nebula/etc/` 。

查看运行日志级别

使用如下命令查看当前所有的 gflags 参数（包括日志参数）：

```
$ curl <ws_ip>:<ws_port>/flags
```

参数	说明
`ws_ip`	HTTP 服务的 IP 地址，可以在配置文件中查看。默认值为 `127.0.0.1` 。
`ws_port`	HTTP 服务的端口，可以在配置文件中查看。默认值分别为 `19559`（Meta）、`19669`（Graph）`19779`（Storage）。

示例如下：

- 查看 Meta 服务当前的最小日志级别：

```
$ curl 127.0.0.1:19559/flags | grep 'minloglevel'
```

- 查看 Storage 服务当前的日志详细级别：

```
$ curl 127.0.0.1:19779/flags | grep -w 'v'
```

修改运行日志级别

使用如下命令修改运行日志级别：

2022 vesoft Inc.

```
$ curl -X PUT -H "Content-Type: application/json" -d '{"<key>":<value>[,"<key>":<value>]}' "<ws_ip>:<ws_port>/flags"
```

参数	说明
`key`	待修改的运行日志类型，可选值请参见配置说明。
`value`	运行日志级别，可选值请参见配置说明。
`ws_ip`	HTTP 服务的 IP 地址，可以在配置文件中查看。默认值为 `127.0.0.1`。
`ws_port`	HTTP 服务的端口，可以在配置文件中查看。默认值分别为 `19559`（Meta）、`19669`（Graph）`19779`（Storage）。

示例如下：

```
$ curl -X PUT -H "Content-Type: application/json" -d '{"minloglevel":0,"v":3}' "127.0.0.1:19779/flags" # storaged
$ curl -X PUT -H "Content-Type: application/json" -d '{"minloglevel":0,"v":3}' "127.0.0.1:19669/flags" # graphd
$ curl -X PUT -H "Content-Type: application/json" -d '{"minloglevel":0,"v":3}' "127.0.0.1:19559/flags" # metad
```

如果在 Nebula Graph 运行时修改了运行日志级别，重启服务后会恢复为配置文件中设置的级别，如果需要永久修改，请修改配置文件。

RocksDB 运行日志

RocksDB 的运行日志通常在 `/usr/local/nebula/data/storage/nebula/$id/data/LOG`，其中 `$id` 为实例号。该日志通常用于调试 RocksDB 参数。

最后更新: June 20, 2023

2022 vesoft Inc.

6.2.2 审计日志

Nebula Graph 的审计日志功能可以将 Graph 服务接受到的所有操作进行分类存储，然后提供给终端用户查看，终端用户可以根据需要，追踪指定类型的操作。

Enterpriseonly

仅企业版支持本功能。

日志类别

类别	语句	说明
`login`	-	客户端尝试连接 Graph 服务时，记录相关信息。
`exit`	-	断开与 Graph 服务的连接时，记录相关信息。
`ddl`	`CREATE SPACE` 、 `DROP SPACE` 、 `CREATE TAG` 、 `DROP TAG` 、 `ALTER TAG` 、 `DELETE TAG` 、 `CREATE EDGE` 、 `DROP EDGE` 、 `ALTER EDGE` 、 `CREATE INDEX` 、 `REBUILD INDEX` 、 `DROP INDEX` 、 `CREATE FULLTEXT INDEX` 、 `REBUILD FULLTEXT INDEX` 、 `DROP FULLTEXT INDEX`	记录 DDL 语句的信息。
`dql`	`MATCH` 、 `LOOKUP` 、 `GO` 、 `FETCH` 、 `GET SUBGRAPH` 、 `FIND PATH` 、 `UNWIND` 、 `GROUP BY` 、 `ORDER BY` 、 `YIELD` 、 `LIMIT` 、 `RETURN`	记录 DQL 语句的信息。
`dml`	`INSERT VERTEX` 、 `DELETE VERTEX` 、 `UPDATE VERTEX` 、 `UPSERT VERTEX` 、 `INSERT EDGE` 、 `DELETE EDGE` 、 `UPDATE EDGE` 、 `UPSERT EDGE`	记录 DML 语句的信息。
`dcl`	`CREATE USER` 、 `GRANT ROLE` 、 `REVOKE ROLE` 、 `CHANGE PASSWORD` 、 `ALTER USER` 、 `DROP USER` 、 `CREATE SNAPSHOT` 、 `DROP SNAPSHOT` 、 `ADD LISTENER` 、 `REMOVE LISTENER` 、 `BALANCE` 、 `SUBMIT JOB` 、 `STOP JOB` 、 `RECOVER JOB` 、 `ADD DRAINER` 、 `REMOVE DRAINER`	记录 DCL 语句的信息。
`util`	`SHOW HOSTS` 、 `SHOW USERS` 、 `SHOW ROLES` 、 `SHOW SNAPSHOTS` 、 `SHOW SPACES` 、 `SHOW PARTS` 、 `SHOW TAGS` 、 `SHOW EDGES` 、 `SHOW INDEXES` 、 `SHOW CREATE SPACE` 、 `SHOW CREATE TAG/EDGE` 、 `SHOW CREATE INDEX` 、 `SHOW INDEX STATUS` 、 `SHOW LISTENER` 、 `SHOW TEXT SEARCH CLIENTS` 、 `SHOW DRAINER CLIENTS` 、 `SHOW FULLTEXT INDEXES` 、 `SHOW CONFIGS` 、 `SHOW CHARSET` 、 `SHOW COLLATION` 、 `SHOW STATS` 、 `SHOW SESSIONS` 、 `SHOW META LEADER` 、 `SHOW DRAINERS` 、 `SHOW QUERIES` 、 `SHOW JOB` 、 `SHOW JOBS` 、 `DESCRIBE INDEX` 、 `DESCRIBE EDGE` 、 `DESCRIBE TAG` 、 `DESCRIBE SPACE` 、 `DESCRIBE USER` 、 `USE SPACE` 、 `SIGN IN TEXT SERVICE` 、 `SIGN OUT TEXT SERVICE` 、 `SIGN IN DRAINER SERVICE` 、 `SIGN OUT DRAINER SERVICE` 、 `EXPLAIN` 、 `PROFILE` 、 `KILL QUERY` 、 `DOWNLOAD HDFS` 、 `INGEST`	记录工具类语句的信息。
`unknown`	-	记录未能识别的语句。

设置审计日志

使用审计日志需要修改集群内的所有 Graph 服务的配置（`nebula-graphd.conf`），默认路径为 `/usr/local/nebula/etc/nebula-graphd.conf` 。

2022 vesoft Inc.

> **Note**
>
> 修改配置后，需要重启 Graph 服务才能生效。

与审计日志相关的参数说明如下。

参数	预设值	说明
enable_audit	false	是否开启审计日志。
audit_log_handler	file	审计日志的存储方案。可选值为 file（本地文件）和 es（Elasticsearch）。
audit_log_file	./logs/audit/audit.log	仅在 audit_log_handler=file 时生效。审计日志的存储路径，支持相对路径或绝对路径。
audit_log_strategy	synchronous	仅在 audit_log_handler=file 时生效。审计日志的同步方案。可选值为 asynchronous 和 synchronous。设置为 asynchronous 时，日志事件使用内存缓冲，不会阻塞主线程，但是可能会因为缓存不够而导致日志缺失；设置为 synchronous 时，日志事件每次都刷新并同步到文件中。
audit_log_max_buffer_size	1048576	仅在 audit_log_handler=file 、audit_log_strategy=asynchronous 时生效。审计日志的缓存大小。单位：字节。
audit_log_format	xml	仅在 audit_log_handler=file 时生效。审计日志的格式。可选值为 xml 、json 和 csv。
audit_log_es_address	-	仅在 audit_log_handler=es 时生效。Elasticsearch 服务器的地址。格式为 IP1:port1, IP2:port2, ...。
audit_log_es_user	-	仅在 audit_log_handler=es 时生效。登录 Elasticsearch 服务器的用户名。
audit_log_es_password	-	仅在 audit_log_handler=es 时生效。Elasticsearch 用户名对应的密码。
audit_log_es_batch_size	1000	仅在 audit_log_handler=es 时生效。每次发送至 Elasticsearch 服务器的日志条数。
audit_log_exclude_spaces	-	不需要记录日志的图空间列表。多个图空间用英文逗号（,）分隔。
audit_log_categories	login,exit	需要记录日志的分类列表。多个类别用英文逗号（,）分隔。

审计日志格式

不同的存储方案和不同的格式，日志内的字段是相同的。以默认路径（ logs/audit/audit.log ）和默认 XML 格式为例说明各个字段的含义。

```
<AUDIT_RECORD
  CATEGORY="util"
  TIMESTAMP="2022-04-07 02:31:38"
  TERMINAL=""
  CONNECTION_ID="1649298693144580"
  CONNECTION_STATUS="0"
  CONNECTION_MESSAGE=""
  USER="root"
  CLIENT_HOST="127.0.0.1"
  HOST="192.168.8.111"
  SPACE=""
  QUERY="use basketballplayer1"
  QUERY_STATUS="-1005"
  QUERY_MESSAGE="SpaceNotFound: "
/>
<AUDIT_RECORD
  CATEGORY="util"
  TIMESTAMP="2022-04-07 02:31:39"
  TERMINAL=""
  CONNECTION_ID="1649298693144580"
  CONNECTION_STATUS="0"
  CONNECTION_MESSAGE=""
  USER="root"
  CLIENT_HOST="127.0.0.1"
  HOST="192.168.8.111"
  SPACE=""
  QUERY="use basketballplayer"
  QUERY_STATUS="0"
```


2022 vesoft Inc.

```
  QUERY_MESSAGE=""
/>
```

字段	说明
CATEGORY	日志类别。
TIMESTAMP	日志生成时间。
TERMINAL	保留字段，暂不支持。
CONNECTION_ID	连接的会话ID。
CONNECTION_STATUS	连接的状态码。0 表示成功，其他数字代表不同的错误信息。
CONNECTION_MESSAGE	如果连接出错，会显示报错信息。
USER	连接的用户名。
CLIENT_HOST	客户端的 IP 地址。
HOST	连接的机器的 IP 地址。
SPACE	执行查询的图空间。
QUERY	查询语句。
QUERY_STATUS	查询状态。0 表示成功，其他数字代表不同的错误信息。
QUERY_MESSAGE	如果查询出错，会显示报错信息。

最后更新: June 20, 2023

2022 vesoft Inc.

7. 监控

7.1 查询 Nebula Graph 监控指标

Nebula Graph 支持多种方式查询服务的监控指标，本文将介绍最基础的方式，即通过 HTTP 端口查询。

7.1.1 监控指标结构说明

Nebula Graph 的每个监控指标都由三个部分组成，中间用英文句号（.）隔开，例如 `num_queries.sum.600` 。不同的 Nebula Graph 服务支持查询的监控指标也不同。指标结构的说明如下。

类别	示例	说明
指标名称	`num_queries`	简单描述指标的含义。
统计类型	`sum`	指标统计的方法。当前支持 SUM、AVG、RATE 和 P 分位数（P75、P95、P99、P999）。
统计时间	`600`	指标统计的时间范围，当前支持 5 秒、60 秒、600 秒和 3600 秒，分别表示最近 5 秒、最近 1 分钟、最近 10 分钟和最近 1 小时。

图空间监控指标

Graph 服务支持一系列基于图空间的监控指标（Space Level Metrics），对不同图空间的数据分别记录。

如需开启图空间监控指标，先在 Graph 服务的配置文件中将 `enable_space_level_metrics` 参数的值修改为 `true`，再启动 Nebula Graph。修改配置的详细方式参见配置管理。

> **Note**
>
> 图空间指标只能通过查询所有监控指标的形式查询到，例如 `curl -G "http://192.168.8.40:19559/stats"`，返回结果中以 `{space=space_name}` 的形式包含图空间名称，例如 `num_active_queries{space=basketballplayer}.sum.5=0` 。

7.1.2 通过 HTTP 端口查询监控指标

语法

```
curl -G "http://<ip>:<port>/stats?stats=<metric_name_list> [&format=json]"
```

选项	说明
`ip`	服务器的 IP 地址，可以在安装目录内查看配置文件获取。
`port`	服务器的 HTTP 端口，可以在安装目录内查看配置文件获取。默认情况下，Meta 服务端口为 19559，Graph 服务端口为 19669，Storage 服务端口为 19779。
`metric_name_list`	监控指标名称，多个监控指标用英文逗号（,）隔开。
`&format=json`	将结果以 JSON 格式返回。

> **Note**
>
> 如果 Nebula Graph 服务部署在容器中，需要执行 `docker-compose ps` 命令查看映射到容器外部的端口，然后通过该端口查询。

示例

- 查询单个监控指标

2022 vesoft Inc.

查询 Graph 服务中，最近 10 分钟的请求总数。

```
$ curl -G "http://192.168.8.40:19669/stats?stats=num_queries.sum.600"
num_queries.sum.600=400
```

- 查询多个监控指标

  查询 Meta 服务中，最近 1 分钟的心跳平均延迟和最近 10 分钟 P99 心跳（1%最慢的心跳）的平均延迟。

```
$ curl -G "http://192.168.8.40:19559/stats?stats=heartbeat_latency_us.avg.60,heartbeat_latency_us.p99.600"
heartbeat_latency_us.avg.60=281
heartbeat_latency_us.p99.600=985
```

- 查询监控指标并以 JSON 格式返回

  查询 Storage 服务中，最近 10 分钟新增的点数量，并以 JSON 格式返回结果。

```
$ curl -G "http://192.168.8.40:19779/stats?stats=num_add_vertices.sum.600&format=json"
[{"value":1,"name":"num_add_vertices.sum.600"}]
```

- 查询服务器的所有监控指标

  不指定查询某个监控指标时，会返回该服务器上所有的监控指标。

```
$ curl -G "http://192.168.8.40:19559/stats"
heartbeat_latency_us.avg.5=304
heartbeat_latency_us.avg.60=308
heartbeat_latency_us.avg.600=299
heartbeat_latency_us.avg.3600=285
heartbeat_latency_us.p75.5=652
heartbeat_latency_us.p75.60=669
heartbeat_latency_us.p75.600=651
heartbeat_latency_us.p75.3600=642
heartbeat_latency_us.p95.5=930
heartbeat_latency_us.p95.60=963
heartbeat_latency_us.p95.600=933
heartbeat_latency_us.p95.3600=929
heartbeat_latency_us.p99.5=986
heartbeat_latency_us.p99.60=1409
heartbeat_latency_us.p99.600=989
heartbeat_latency_us.p99.3600=986
num_heartbeats.rate.5=0
num_heartbeats.rate.60=0
num_heartbeats.rate.600=0
num_heartbeats.rate.3600=0
num_heartbeats.sum.5=2
num_heartbeats.sum.60=40
num_heartbeats.sum.600=394
num_heartbeats.sum.3600=2364
...
```


2022 vesoft Inc.

7.1.3 监控指标说明

Graph

参数	说明
num_active_queries	当前正在执行的查询数。
num_active_sessions	当前活跃的会话数量。
num_aggregate_executors	聚合（Aggregate）算子的执行次数。
num_auth_failed_sessions_bad_username_password	因用户名密码错误导验证失败的会话数量。
num_auth_failed_sessions_out_of_max_allowed	因为超过 FLAG_OUT_OF_MAX_ALLOWED_CONNECTIONS 参数导致的验证登录的失败的 session 数量。
num_auth_failed_sessions	登录验证失败的会话数量。
num_indexscan_executors	索引扫描（IndexScan）算子的执行次数。
num_killed_queries	被终止的查询数量。
num_opened_sessions	服务端建立过的会话数量。
num_queries	查询次数。
num_query_errors_leader_changes	因查询错误而导致的 Leader 变更的次数。
num_query_errors	查询错误次数。
num_reclaimed_expired_sessions	服务端主动回收的过期的会话数量。
num_rpc_sent_to_metad_failed	Graphd 服务发给 Metad 的 RPC 请求失败的数量。
num_rpc_sent_to_metad	Graphd 服务发给 Metad 服务的 RPC 请求数量。
num_rpc_sent_to_storaged_failed	Graphd 服务发给 Storaged 服务的 RPC 请求失败的数量。
num_rpc_sent_to_storaged	Graphd 服务发给 Storaged 服务的 RPC 请求数量。
num_sentences	Graphd 服务接收的语句数。
num_slow_queries	慢查询次数。
num_sort_executors	排序（Sort）算子次数。
optimizer_latency_us	优化器阶段延迟时间。
query_latency_us	查询平均延迟时间。
slow_query_latency_us	慢查询平均延迟时间。
num_queries_hit_memory_watermark	达到内存水位线的语句的数量。

2022 vesoft Inc.

Meta

参数	说明
commit_log_latency_us	Raft 协议中 Commit 日志的延迟时间。
commit_snapshot_latency_us	Raft 协议中 Commit 快照的延迟时间。
heartbeat_latency_us	心跳延迟时间。
num_heartbeats	心跳次数。
num_raft_votes	Raft 协议中投票的次数。
transfer_leader_latency_us	Raft 协议中转移 Leader 的延迟时间。
num_agent_heartbeats	AgentHBProcessor 心跳次数。
agent_heartbeat_latency_us	AgentHBProcessor 延迟时间。
replicate_log_latency_us	Raft 复制日志至大多数节点的延迟。
num_send_snapshot	Raft 发送快照至其他节点的次数。
append_log_latency_us	Raft 复制日志到单个节点的延迟。
append_wal_latency_us	Raft 写入单条 WAL 的延迟。
num_grant_votes	Raft 投票给其他节点的次数。
num_start_elect	Raft 发起投票的次数。

2022 vesoft Inc.

Storage

2022 vesoft Inc.

参数	说明
add_edges_atomic_latency_us	添加边单次延迟。
add_edges_latency_us	添加边的平均延迟时间。
add_vertices_latency_us	添加点的平均延迟时间。
commit_log_latency_us	Raft 协议中 Commit 日志的延迟时间。
commit_snapshot_latency_us	Raft 协议中 Commit 快照的延迟时间。
delete_edges_latency_us	删除边的平均延迟时间。
delete_vertices_latency_us	删除点的平均延迟时间。
get_neighbors_latency_us	查询邻居平均延迟时间。
num_get_prop	GetPropProcessor 执行的次数。
num_get_neighbors_errors	GetNeighborsProcessor 执行出错的次数。
get_prop_latency_us	GetPropProcessor 执行的延迟时间。
num_edges_deleted	删除的边数量。
num_edges_inserted	插入的边数量。
num_raft_votes	Raft 协议中投票的次数。
num_rpc_sent_to_metad_failed	Storaged 服务发给 Metad 服务的 RPC 请求失败的数量。
num_rpc_sent_to_metad	Storaged 服务发给 Metad 服务的 RPC 请求数量。
num_tags_deleted	删除的 Tag 数量。
num_vertices_deleted	删除的点数量。
num_vertices_inserted	插入的点数量。
transfer_leader_latency_us	Raft 协议中转移 Leader 的延迟时间。
lookup_latency_us	LookupProcessor 执行的延迟时间。
num_lookup_errors	LookupProcessor 执行时出错的次数。
num_scan_vertex	ScanVertexProcessor 执行的次数。
num_scan_vertex_errors	ScanVertexProcessor 执行时出错的次数。
update_edge_latency_us	UpdateEdgeProcessor 执行的延迟时间。
num_update_vertex	UpdateVertexProcessor 执行的次数。
num_update_vertex_errors	UpdateVertexProcessor 执行时出错的次数。
kv_get_latency_us	Getprocessor 的延迟时间。
kv_put_latency_us	PutProcessor 的延迟时间。
kv_remove_latency_us	RemoveProcessor 的延迟时间。
num_kv_get_errors	GetProcessor 执行出错次数。
num_kv_get	GetProcessor 执行次数。
num_kv_put_errors	PutProcessor 执行出错次数。
num_kv_put	PutProcessor 执行次数。
num_kv_remove_errors	RemoveProcessor 执行出错次数。

2022 vesoft Inc.

参数	说明
num_kv_remove	RemoveProcessor 执行次数。
forward_tranx_latency_us	传输平均延迟时间。
scan_edge_latency_us	ScanEdgeProcessor 执行的延迟时间。
num_scan_edge_errors	ScanEdgeProcessor 执行时出错的次数。
num_scan_edge	ScanEdgeProcessor 执行的次数。
scan_vertex_latency_us	ScanVertexProcessor 执行的延迟时间。
num_add_edges	添加边的次数。
num_add_edges_errors	添加边时出错的次数。
num_add_vertices	添加点的次数。
num_start_elect	Raft 发起投票的次数
num_add_vertices_errors	添加点时出错的次数。
num_delete_vertices_errors	删除点时出错的次数。
append_log_latency_us	Raft 复制日志到单个节点的延迟。
num_grant_votes	Raft 投票给其他节点的次数。
replicate_log_latency_us	Raft 复制日志到大多数节点的延迟。
num_delete_tags	删除 Tag 的次数。
num_delete_tags_errors	删除 Tag 时出错的次数。
num_delete_edges	删除边的次数。
num_delete_edges_errors	删除边时出错的次数。
num_send_snapshot	发送快照的次数。
update_vertex_latency_us	UpdateVertexProcessor 执行的延迟时间。
append_wal_latency_us	Raft 写入单条 WAL 的延迟。
num_update_edge	UpdateEdgeProcessor 执行的次数。
delete_tags_latency_us	删除 Tag 的平均延迟时间。
num_update_edge_errors	UpdateEdgeProcessor 执行时出错的次数。
num_get_neighbors	GetNeighborsProcessor 执行的次数。
num_get_prop_errors	GetPropProcessor 执行时出错的次数。
num_delete_vertices	删除点的次数。
num_lookup	LookupProcessor 执行的次数。

2022 vesoft Inc.

图空间级别监控指标

参数	说明
num_active_queries	当前正在执行的查询数。
num_queries	查询次数。
num_sentences	Graphd 服务接收的语句数。
optimizer_latency_us	优化器阶段延迟时间。
query_latency_us	查询平均延迟时间。
num_slow_queries	慢查询次数。
num_query_errors	查询报错语句数量。
num_query_errors_leader_changes	因查询错误而导致的 Leader 变更的次数。
num_killed_queries	被终止的查询数量。
num_aggregate_executors	聚合（Aggregate）算子的执行次数。
num_sort_executors	排序（Sort）算子的执行次数。
num_indexscan_executors	索引扫描（IndexScan）算子的执行次数。
num_oom_queries	导致内存耗尽的语句数量。
num_auth_failed_sessions_bad_username_password	因用户名密码错误导验证失败的会话数量。
num_auth_failed_sessions	登录验证失败的会话数量。
num_opened_sessions	服务端建立过的会话数量。
num_queries_hit_memory_watermark	达到内存水位线的语句的数量。
num_reclaimed_expired_sessions	服务端主动回收的过期的会话数量。
num_rpc_sent_to_metad_failed	Graphd 服务发给 Metad 的 RPC 请求失败的数量。
num_rpc_sent_to_metad	Graphd 服务发给 Metad 服务的 RPC 请求数量。
num_rpc_sent_to_storaged_failed	Graphd 服务发给 Storaged 服务的 RPC 请求失败的数量。
num_rpc_sent_to_storaged	Graphd 服务发给 Storaged 服务的 RPC 请求数量。
slow_query_latency_us	慢查询平均延迟时间。

最后更新: June 20, 2023

2022 vesoft Inc.

7.2 RocksDB 统计数据

Nebula Graph 使用 RocksDB 作为底层存储，本文介绍如何收集和展示 Nebula Graph 的 RocksDB 统计信息。

7.2.1 启用 RocksDB

RocksDB 统计功能默认关闭，启动 RocksDB 统计功能，你需要：

1. 修改 `nebula-storaged.conf` 文件中 `--enable_rocksdb_statistics` 参数为 `true`。配置默认文件目录为 `/use/local/nebula/etc`。
2. 重启服务使修改生效。

7.2.2 获取 RocksDB 统计信息

用户可以使用存储服务中的内置 HTTP 服务来获取以下类型的统计信息，且支持返回 JSON 格式的结果：

- 所有统计信息。
- 指定条目的信息。

7.2.3 示例

使用以下命令获取所有 RocksDB 统计信息：

```
curl -L "http://${storage_ip}:${port}/rocksdb_stats"
```

例如：

```
curl -L "http://172.28.2.1:19779/rocksdb_stats"

rocksdb.blobdb.blob.file.bytes.read=0
rocksdb.blobdb.blob.file.bytes.written=0
rocksdb.blobdb.blob.file.bytes.synced=0
...
```

使用以下命令获取部分 RocksDB 统计信息：

```
curl -L "http://${storage_ip}:${port}/rocksdb_stats?stats=${stats_name}"
```

例如使用以下语句获取 `rocksdb.bytes.read` 和 `rocksdb.block.cache.add` 的信息。

```
curl -L "http://172.28.2.1:19779/rocksdb_stats?stats=rocksdb.bytes.read,rocksdb.block.cache.add"

rocksdb.block.cache.add=14
rocksdb.bytes.read=1632
```

使用以下命令获取部分 JSON 格式的 RocksDB 统计信息：

```
curl -L "http://${storage_ip}:${port}/rocksdb_stats?stats=${stats_name}&format=json"
```

例如使用以下语句获取 `rocksdb.bytes.read` 和 `rocksdb.block.cache.add` 的统计信息并返回 JSON 的格式数据。

```
curl -L "http://172.28.2.1:19779/rocksdb_stats?stats=rocksdb.bytes.read,rocksdb.block.cache.add&format=json"

[
  {
    "rocksdb.block.cache.add": 1
  },
  {
    "rocksdb.bytes.read": 160
  }
]
```

最后更新: June 20, 2023

2022 vesoft Inc.

8. 数据安全

8.1 验证和授权

8.1.1 身份验证

身份验证用于将会话映射到特定用户，从而实现访问控制。

当客户端连接到 Nebula Graph 时，Nebula Graph 会创建一个会话，会话中存储连接的各种信息，如果开启了身份验证，就会将会话映射到对应的用户。

> **Note**
>
> 默认情况下，身份验证功能是关闭的，用户可以使用 root 用户名和任意密码连接到 Nebula Graph。

Nebula Graph 支持两种身份验证方式：本地身份验证和 LDAP 验证。

本地身份验证

本地身份验证是指在服务器本地存储用户名、加密密码，当用户尝试访问 Nebula Graph 时，将进行身份验证。

启用本地身份验证

1. 编辑配置文件 `nebula-graphd.conf`（默认目录为 `/usr/local/nebula/etc/`），设置如下参数：
   - `--enable_authorize`：是否启用身份验证，可选值：`true`、`false`。
   - `--failed_login_attempts`：可选项，需要手动添加该参数。单个 Graph 节点允许连续输入错误密码的次数。超过该次数时，账户会被锁定。如果有多个 Graph 节点，允许的次数为 `节点数 * 次数`。
   - `--password_lock_time_in_secs`：可选项，需要手动添加该参数。多次输入错误密码后，账户被锁定的时间。单位：秒。
2. 重启 Nebula Graph 服务。

> **Note**
>
> 开启身份验证后，默认的 God 角色账号为 `root`，密码为 `nebula`。角色详情请参见内置角色权限。

OpenLDAP 验证

OpenLDAP 是轻型目录访问协议（LDAP）的开源实现，可以实现账号集中管理。

启用 OPENLDAP 验证

> **Enterpriseonly**
>
> 当前仅企业版支持集成 OpenLDAP 进行身份验证，详情请参见使用 OpenLDAP 进行身份验证。

最后更新: June 20, 2023

2022 vesoft Inc.

8.1.2 用户管理

用户管理是 Nebula Graph 访问控制中不可或缺的组成部分，本文将介绍用户管理的相关语法。

开启身份验证后，用户需要使用已创建的用户才能连接 Nebula Graph，而且连接后可以进行的操作也取决于该用户拥有的角色权限。

> **Note**

- 默认情况下，身份验证功能是关闭的，用户可以使用 root 用户名和任意密码连接到 Nebula Graph。
- 修改权限后，对应的用户需要重新登录才能生效。

创建用户（`CREATE USER`）

执行 `CREATE USER` 语句可以创建新的 Nebula Graph 用户。当前仅 **God** 角色用户（即 `root` 用户）能够执行 `CREATE USER` 语句。

- 语法

```
CREATE USER [IF NOT EXISTS] <user_name> [WITH PASSWORD '<password>'] [WITH IP WHITELIST <ip_list>];
```

- `IF NOT EXISTS`：检测待创建的用户名是否存在，只有不存在时，才会创建新用户。
- `user_name`：待创建的用户名。
- `password`：用户名对应的密码。
- `ip_list`（企业版）：IP 白名单列表。该用户只有通过列表内的 IP 才能连接 Nebula Graph。多个 IP 之间用英文逗号（,）分隔。

- 示例

```
nebula> CREATE USER user1 WITH PASSWORD 'nebula';
nebula> CREATE USER user2 WITH PASSWORD 'nebula' WITH IP WHITELIST 192.168.10.10,192.168.10.12;
nebula> SHOW USERS;
+---------+--------------------------------+
| Account | IP Whitelist                   |
+---------+--------------------------------+
| "root"  | ""                             |
| "user1" | ""                             |
| "user2" | "192.168.10.10,192.168.10.12" |
+---------+--------------------------------+
```

授权用户（`GRANT ROLE`）

执行 `GRANT ROLE` 语句可以将指定图空间的内置角色权限授予用户。当前仅 **God** 角色用户和 **Admin** 角色用户能够执行 `GRANT ROLE` 语句。角色权限的说明，请参见内置角色权限。

- 语法

```
GRANT ROLE <role_type> ON <space_name> TO <user_name>;
```

- 示例

```
nebula> GRANT ROLE USER ON basketballplayer TO user1;
```


2022 vesoft Inc.

撤销用户权限（`REVOKE ROLE`）

执行 `REVOKE ROLE` 语句可以撤销用户的指定图空间的内置角色权限。当前仅 **God** 角色用户和 **Admin** 角色用户能够执行 `REVOKE ROLE` 语句。角色权限的说明，请参见内置角色权限。

- 语法

```
REVOKE ROLE <role_type> ON <space_name> FROM <user_name>;
```

- 示例

```
nebula> REVOKE ROLE USER ON basketballplayer FROM user1;
```

查看指定用户权限（`DESCRIBE USER`）

执行 `DESCRIBE USER` 语句可以查看指定用户的角色权限信息。

- 语法

```
DESCRIBE USER <user_name>;
DESC USER <user_name>;
```

- 示例

```
nebula> DESCRIBE USER user1;
+---------+--------------------+
| role    | space              |
+---------+--------------------+
| "ADMIN" | "basketballplayer" |
+---------+--------------------+
```

查看指定空间内用户权限（`SHOW ROLES`）

执行 `SHOW ROLES` 语句可以查看指定空间内的所有用户（除 `root` 以外）和对应角色权限的信息。

- 语法

```
SHOW ROLES IN <space_name>;
```

- 示例

```
nebula> SHOW ROLES IN basketballplayer;
+---------+-----------+
| Account | Role Type |
+---------+-----------+
| "user1" | "ADMIN"   |
+---------+-----------+
```

修改用户密码（`CHANGE PASSWORD`）

执行 `CHANGE PASSWORD` 语句可以修改用户密码，修改时需要提供旧密码和新密码。

- 语法

```
CHANGE PASSWORD <user_name> FROM '<old_password>' TO '<new_password>';
```

- 示例

```
nebula> CHANGE PASSWORD user1 FROM 'nebula' TO 'nebula123';
```


2022 vesoft Inc.

修改用户密码和 IP 白名单（`ALTER USER`）

执行 `ALTER USER` 语句可以修改用户密码和 IP 白名单，修改时不需要提供旧密码。当前仅 **God** 角色用户（即 `root` 用户）能够执行 `ALTER USER` 语句。

- 语法

```
ALTER USER <user_name> WITH PASSWORD '<password>' [WITH IP WHITELIST <ip_list>];
```

- 示例

> **Enterpriseonly**
>
> 没有使用 `WITH IP WHITELIST` 时，表示取消 IP 白名单，用户使用任何 IP 都可以连接 Nebula Graph。

```
nebula> ALTER USER user2 WITH PASSWORD 'nebula';
nebula> SHOW USERS;
+---------+--------------+
| Account | IP Whitelist |
+---------+--------------+
| "root"  | ""           |
| "user1" | ""           |
| "user2" | ""           |
+---------+--------------+

nebula> ALTER USER user2 WITH PASSWORD 'nebula' WITH IP WHITELIST 192.168.10.10;
```

删除用户（`DROP USER`）

执行 `DROP USER` 语句可以删除用户。当前仅 **God** 角色用户能够执行 `DROP USER` 语句。

> **Note**
>
> 删除用户不会自动断开该用户当前会话，而且权限仍在当前会话中生效。

- 语法

```
DROP USER [IF EXISTS] <user_name>;
```

- 示例

```
nebula> DROP USER user1;
```

查看用户列表（`SHOW USERS`）

执行 `SHOW USERS` 语句可以查看用户列表。当前仅 **God** 角色用户能够执行 `SHOW USERS` 语句。

- 语法

```
SHOW USERS;
```

- 示例

```
nebula> SHOW USERS;
+---------+-----------------+
| Account | IP Whitelist    |
+---------+-----------------+
| "root"  | ""              |
| "user1" | ""              |
| "user2" | "192.168.10.10" |
+---------+-----------------+
```


2022 vesoft Inc.

最后更新: June 20, 2023

2022 vesoft Inc.

8.1.3 内置角色权限

所谓角色，就是一组相关权限的集合。用户可以把角色分配给创建的用户，从而实现访问控制。

内置角色

Nebula Graph 内置了多种角色，说明如下：

- God
- 初始最高权限角色，拥有所有操作的权限。类似于 Linux 中的 `root` 和 Windows 中的 `administrator`。
- Meta 服务初始化时，会自动创建 God 角色用户 `root`，密码为 `nebula`。

  注意

  请及时修改 `root` 用户的密码，保证数据安全。

- 一个集群只能有一个 God 角色用户，该用户可以管理集群内所有图空间。
- 不支持手动授权 God 角色，只能使用默认 God 角色用户 `root`。

- Admin
- 对权限内的图空间拥有 Schema 和 data 的读写权限。
- 可以将权限内的图空间授权给其他用户。

  注意

  只能授权低于 ADMIN 级别的角色给其他用户。

- DBA
- 对权限内的图空间拥有 Schema 和 data 的读写权限。
- 无法将权限内的图空间授权给其他用户。

- User
- 对权限内的图空间拥有 Schema 的只读权限。
- 对权限内的图空间拥有 data 的读写权限。

- Guest
- 对权限内的图空间拥有 Schema 和 data 的只读权限。

Note

- 不支持自行创建角色，只能使用默认的内置角色。
- 一个用户在一个图空间内只能拥有一个角色权限。授权用户请参见用户管理。

2022 vesoft Inc.

角色权限

各角色的执行权限如下。

权限	God	Admin	DBA	User	Guest	相关语句
Read space	Y	Y	Y	Y	Y	`USE` 、 `DESCRIBE SPACE`
Read schema	Y	Y	Y	Y	Y	`DESCRIBE TAG` 、 `DESCRIBE EDGE` 、 `DESCRIBE TAG INDEX` 、 `DESCRIBE EDGE INDEX`
Write schema	Y	Y	Y			`CREATE TAG` 、 `ALTER TAG` 、 `CREATE EDGE` 、 `ALTER EDGE` 、 `DROP TAG` 、 `DELETE TAG` 、 `DROP EDGE` 、 `CREATE TAG INDEX` 、 `CREATE EDGE INDEX` 、 `DROP TAG INDEX` 、 `DROP EDGE INDEX`
Write user	Y					`CREATE USER` 、 `DROP USER` 、 `ALTER USER`
Write role	Y	Y				`GRANT` 、 `REVOKE`
Read data	Y	Y	Y	Y	Y	`GO` 、 `SET` 、 `PIPE` 、 `MATCH` 、 `ASSIGNMENT` 、 `LOOKUP` 、 `YIELD` 、 `ORDER BY` 、 `FETCH VERTICES` 、 `Find` 、 `FETCH EDGES` 、 `FIND PATH` 、 `LIMIT` 、 `GROUP BY` 、 `RETURN`
Write data	Y	Y	Y	Y		`INSERT VERTEX` 、 `UPDATE VERTEX` 、 `INSERT EDGE` 、 `UPDATE EDGE` 、 `DELETE VERTEX` 、 `DELETE EDGES` 、 `DELETE TAG`
Show operations	Y	Y	Y	Y	Y	`SHOW` 、 `CHANGE PASSWORD`
Job	Y	Y	Y	Y		`SUBMIT JOB COMPACT` 、 `SUBMIT JOB FLUSH` 、 `SUBMIT JOB STATS` 、 `STOP JOB` 、 `RECOVER JOB` 、 `BUILD TAG INDEX` 、 `BUILD EDGE INDEX` 、 `INGEST` 、 `DOWNLOAD`
Write space	Y					`CREATE SPACE` 、 `DROP SPACE` 、 `CREATE SNAPSHOT` 、 `DROP SNAPSHOT` 、 `BALANCE` 、 `ADMIN` 、 `CONFIG`

Caution

Show operations 为特殊操作，只会在自身权限内执行。例如 `SHOW SPACES`，每个角色都可以执行，但是只会返回自身权限内的图空间。 只有 God 角色可以执行 `SHOW USERS` 和 `SHOW SNAPSHOTS` 语句。

最后更新: June 20, 2023

2022 vesoft Inc.

8.1.4 使用 OpenLDAP 进行身份验证

本文介绍如何将 Nebula Graph 连接到 OpenLDAP 服务器，使用 OpenLDAP 中定义的 DN（Distinguished Name）和密码进行身份验证。

> **Enterpriseonly**
>
> 仅企业版支持本功能。

认证方式

启用 OpenLDAP 身份验证后，输入用户的账号和密码登录 Nebula Graph 时，Nebula Graph 会在 Meta 服务中查找登录账号是否存在，如果账号存在，再根据认证方式去 OpenLDAP 中找到对应的 DN，验证密码。

OpenLDAP 支持的认证方式有两种：简单绑定认证和搜索绑定认证。

简单绑定认证（SIMPLEBINDAUTH）

简单绑定认证会根据登录账号和 Graph 服务配置信息，拼接成 OpenLDAP 可以识别的 DN，然后根据 DN 和密码，在 OpenLDAP 上进行验证。

搜索绑定认证（SEARCHBINDAUTH）

搜索绑定认证会读取 Graph 服务配置信息，查询配置信息中的`uid`和登录账号是否匹配，如果匹配，就读取这个 DN，然后用 DN 和密码，在 OpenLDAP 上进行验证。

前提条件

- 已安装 OpenLDAP。
- 已在 OpenLDAP 上导入用户的账号和密码信息。
- OpenLDAP 所在服务器已开放相应认证端口。

2022 vesoft Inc.

操作步骤

以 OpenLDAP 上已存在的账号 `test2` 、密码 `passwdtest2` 为例进行演示。

1. 连接 Nebula Graph，创建与 OpenLDAP 中对应的影子账号 `test2` 并授权。

```
nebula> CREATE USER test2 WITH PASSWORD '';
nebula> GRANT ROLE ADMIN ON basketballplayer TO test2;
```

> **Note**
>
> Nebula Graph 内创建用户时，密码可以任意设置。

2. 编辑配置文件 `nebula-graphd.conf` （默认目录为 `/usr/local/nebula/etc/` ）：

- 简单绑定认证（推荐）

```
# 是否从配置文件获取配置信息。
--local_config=true
# 是否开启身份验证
--enable_authorize=true
# 身份验证方式：password、ldap、cloud
--auth_type=ldap
# OpenLDAP 服务器地址
--ldap_server=192.168.8.211
# OpenLDAP 服务器端口
--ldap_port=389
# OpenLDAP 中的 Schema 名称
--ldap_scheme=ldap
# DN 前缀
--ldap_prefix=uid=
# DN 后缀
--ldap_suffix=,ou=it,dc=sys,dc=com
```

- 搜索绑定认证

```
# 是否从配置文件获取配置信息。
--local_config=true
# 是否开启身份验证
--enable_authorize=true
# 身份验证方式：password、ldap、cloud
--auth_type=ldap
# OpenLDAP 服务器地址
--ldap_server=192.168.8.211
# OpenLDAP 服务器端口
--ldap_port=389
# OpenLDAP 中的 Schema 名称
--ldap_scheme=ldap
# 绑定目标对象的 DN
--ldap_basedn=ou=it,dc=sys,dc=com
```

3. 重启 Nebula Graph 服务，让新配置生效。
4. 进行登录测试。

```
$ ./nebula-console --addr 127.0.0.1 --port 9669 -u test2 -p passwdtest2
2021/09/08 03:49:39 [INFO] connection pool is initialized successfully

Welcome to Nebula Graph!
```

> **Note**
>
> 使用 OpenLDAP 进行身份验证后，无法使用 Nebula Graph 内置账号密码（包括 `root` ）登录。

最后更新: June 20, 2023

2022 vesoft Inc.

8.2 SSL 加密

Nebula Graph 支持在客户端、Graph 服务、Meta 服务和 Storage 服务之间进行 SSL 加密传输，本文介绍如何设置 SSL 加密。

8.2.1 注意事项

开启 SSL 加密会轻微影响性能。

8.2.2 参数介绍

参数	默认值	说明
`cert_path`	-	PEM 证书的路径。
`key_path`	-	密钥证书的路径。
`password_path`	-	密码文件证书的路径。
`ca_path`	-	受信任 CA 证书文件的路径。
`enable_ssl`	`false`	是否开启 SSL 加密。
`enable_graph_ssl`	`false`	是否仅在 Graph 服务上开启 SSL 加密。
`enable_meta_ssl`	`false`	是否仅在 Meta 服务上开启 SSL 加密。

8.2.3 证书模式

为了使用 SSL 加密，必须有 SSL 证书。Nebula Graph 支持两种证书模式：

- 自签名证书模式

  需要自行制作签名证书。需要根据加密策略，在对应的配置文件内设置 `cert_path` 、`key_path` 和 `password_path` 。

- CA 签名证书模式

  需要在认证机构（Certificate Authority）申请签名证书。需要根据加密策略，在对应的配置文件内设置 `cert_path` 、`key_path` 和 `ca_path` 。

8.2.4 加密策略

Nebula Graph 支持三种加密策略。加密涉及的具体进程请参见详细说明。

- 对客户端、Graph 服务、Meta 服务和 Storage 服务之间的传输数据加密。

  需要修改 `nebula-graphd.conf` 、`nebula-metad.conf` 和 `nebula-storaged.conf` 配置文件，设置 `enable_ssl = true` 。

- 对客户端和 Graph 服务之间的传输数据加密。

  适用于集群设置在同一个机房内，仅对外开放 Graph 服务的端口。因为其他服务可以通过内部网络通信，无需加密。需要修改 `nebula-graphd.conf` 配置文件，设置 `enable_graph_ssl = true` 。

- 对集群中 Meta 服务相关的传输数据加密。

  适用于向 meta 服务传输需保密的信息。需要修改 `nebula-graphd.conf` 、`nebula-metad.conf` 和 `nebula-storaged.conf` 配置文件，设置 `enable_meta_ssl = true` 。

2022 vesoft Inc.

8.2.5 使用方式

1. 确认证书模式和加密策略。
2. 在对应的配置文件内增加证书配置和策略配置。

   例如使用自签名证书，并对客户端、Graph 服务、Meta 服务和 Storage 服务之间的数据传输进行加密。需要对三个配置文件都进行如下设置:

   ```
   --cert_path=xxxxxx
   --key_path=xxxxx
   --password_path=xxxxxx
   --enable_ssl=true
   ```

3. 客户端设置安全套接字并添加受信任的 CA。示例代码请参见 nebula-test-run.py。

最后更新: June 20, 2023

2022 vesoft Inc.

9. 备份与恢复

9.1 Nebula BR

9.1.1 什么是 Backup&Restore

Backup&Restore（简称 BR）是一款命令行界面（CLI）工具，可以帮助备份 NebulaGraph 的图空间数据，或者通过备份文件恢复数据。

功能

- 一键操作备份和恢复数据。
- 支持基于以下备份文件恢复数据：
- 本地磁盘（SSD 或 HDD），建议仅在测试环境使用。
- 兼容亚马逊对象存储（Amazon S3）云存储服务接口，例如：阿里云对象存储（Alibaba Cloud OSS）、MinIO、Ceph RGW 等。
- 支持备份并恢复整个 Nebula Graph 集群。
- （实验性功能）支持备份指定图空间数据。

限制

- NebulaGraph 版本需要为 v3.1.0。
- 数据备份仅支持全量备份，不支持增量备份。
- 仅支持单 metad 部署的本地文件的备份和恢复。
- Nebula Listener 暂时不支持备份，且全文索引也不支持备份。
- 如果备份数据到本地磁盘，备份的文件将会放置在每个服务器的本地路径中。也可以在本地挂载 NFS 文件系统，以便将备份数据还原到不同的主机上。
- 数据备份过程中，指定图空间中的 DDL 和 DML 语句将会阻塞，我们建议在业务低峰期进行操作，例如凌晨 2 点至 5 点。
- 备份图空间只能恢复到原集群，不能跨集群恢复，并且集群的主机数量必须相同；还原指定图空间时将清除集群中其余所有图空间。
- 数据恢复需要删除数据并重启，建议离线进行。

如何使用 **BR**

可以按照如下步骤使用 BR：

1. 安装 BR
2. 使用 BR 备份数据
3. 使用 BR 恢复数据

视频

- Nebula Graph 容灾备份工具 nebula-br 介绍（3 分 34 秒）

最后更新: June 20, 2023

2022 vesoft Inc.

9.1.2 安装 BR

本文介绍如何安装 BR。

安装说明

使用 BR 工具备份和恢复 NebulaGraph 时，需要安装 NebulaGraph Agent 服务。Agent 是集群中每台机器的一个守护进程，用于启停 NebulaGraph 服务和上传、下载备份文件。BR 工具和 Agent 插件的安装方式如下文。

版本兼容性

NebulaGraph	BR	Agent
3.0 ~ 3.2	0.6.1	0.1.0 ~ 0.2.0

安装 BR

使用二进制文件安装

1. 下载 BR。

```
wget https://github.com/vesoft-inc/nebula-br/releases/download/v0.6.1/br-0.6.1-linux-amd64
```

2. 修改文件名称为 `br`。

```
sudo mv br-0.6.1-linux-amd64 br
```

3. 授予 BR 执行权限。

```
sudo chmod +x br
```

4. 执行 `./br version` 查看 BR 版本。

```
[nebula-br]$ ./br version
Nebula Backup And Restore Utility Tool,V-0.6.1
```

使用源码安装

使用源码安装 BR 前，准备工作如下：

- 安装 Go 1.14.x 或更新版本。
- 安装 make。

1. 克隆 `nebula-br` 库至机器。

```
git clone https://github.com/vesoft-inc/nebula-br.git
```

2. 进入 `br` 目录。

```
cd nebula-br
```

3. 编译 BR。

```
make
```

用户可以在命令行输入 `bin/br version`，如果返回以下内容，则认为编译成功。

```
[nebula-br]$ bin/br version
Nebula Backup And Restore Utility Tool,V-0.6.1
   GitSha: 079e7c7
   GitRef: master
please run "help" subcommand for more infomation.
```


2022 vesoft Inc.

安装 **Agent**

NebulaGraph Agent 以二进制形式存在各个机器的安装目录中，通过 RPC 协议对 BR 工具提供服务。

在每台机器中，进行以下操作：

1. 下载 Agent。

```
wget https://github.com/vesoft-inc/nebula-agent/releases/download/v0.2.0/agent-0.2.0-linux-amd64
```

2. 修改 Agent 的名称为 `agent`。

```
sudo mv agent-0.2.0-linux-amd64 agent
```

3. 授予 Agent 可执行权限。

```
sudo chmod +x agent
```

4. 执行以下命令启动 Agent。

Note

启动 Agent 前，确保已经启动集群中的 Meta 服务，并且 Agent 有对应 NebulaGraph 服务目录和备份目录的读写权限。

```
sudo nohup ./nebula_agent --agent="<agent_node_ip>:8888" --meta="<metad_node_ip>:9559" > nebula_agent.log 2>&1 &
```

- `--agent`：Agent 所在机器的 IP 地址和访问端口。
- `--meta`：集群中任一 Meta 服务所在机器的 IP 地址和访问端口。
- `--ratelimit`：可选项（Agent 0.2.0 及以后版本支持此选项），限制文件上传和下载的速度，防止带宽被占满导致其他服务不可用。单位：Bytes。

例如：

```
sudo nohup ./nebula_agent --agent="192.168.8.129:8888" --meta="192.168.8.129:9559" --ratelimit=1048576 > nebula_agent.log 2>&1 &
```

Caution

`--agent` 配置的 IP 地址需要和配置文件中 Meta 和 Storage 服务的地址格式保持一致，即都使用真实 IP 地址或者都使用 `127.0.0.1`，否则 Agent 无法启动。

5. 连接服务并查看 Agent 的运行状态。

```
nebula> SHOW HOSTS AGENT;
+-----------------+------+----------+---------+--------------+---------+
| Host            | Port | Status   | Role    | Git Info Sha | Version |
+-----------------+------+----------+---------+--------------+---------+
| "192.168.8.129" | 8888 | "ONLINE" | "AGENT" | "96646b8"    |         |
+-----------------+------+----------+---------+--------------+---------+
```

常见问题

报错 `E_LIST_CLUSTER_NO_AGENT_FAILURE`

如果操作过程中遇见 `E_LIST_CLUSTER_NO_AGENT_FAILURE` 错误，可能是由于 Agent 服务没有启动或者 Agent 服务没有被注册至 Meta 服务。首先执行 `SHOW HOSTS AGENT` 查看集群上所有节点的 Agent 服务的状态，当时状态显示为 `OFFLINE` 时表示注册 Agent 失败， 然后检查启动 Agent 服务的命令中的 `--meta` 参数是否正确。

最后更新: June 20, 2023

2022 vesoft Inc.

9.1.3 使用 BR 备份数据

成功安装 BR 工具后，可以备份整个图空间的数据，本文介绍如何使用 BR 备份数据。

准备工作

- 安装 BR 和 Agent 并在集群中的每个主机上运行 Agent。
- 确认 Nebula Graph 服务正在运行中。
- 如果在本地保存备份文件，需要在 Meta 服务器、Storage 服务器和 BR 机器上创建绝对路径相同的目录，并记录绝对路径，同时需要保证账号对该目录有写权限。

> **Note**
>
> 在生产环境中，我们建议用户将 NFS (Network File System）存储设备挂载到 Meta 服务器、Storage 服务器和 BR 机器上进行本地备份，或者使用 Alibaba Cloud OSS、Amazon S3 进行远程备份。否则当需要通过本地文件恢复数据时，必须手动将这些备份文件移动到指定目录，会导致数据冗余和某些问题。更多信息，请参见使用 BR 恢复数据。

操作步骤

在 BR 工具的安装路径下（编译安装的默认路径为 `./bin/br`）运行以下命令对整个集群进行全量备份操作。

> **Note**
>
> 确保备份文件的路径存在。

```
$ ./br backup full --meta <ip_address> --storage <storage_path>
```

例如：

- 运行以下命令对 meta 服务的地址为 `192.168.8.129:9559` 的整个集群进行全量备份操作，并将备份文件保存到本地的 `/home/nebula/backup/` 路径下。

> **Caution**
>
> - 如果有多个 metad 地址，可以使用其中任意一个。
> - 备份至本地时，默认只备份 leader metad 的数据，因此有多个 metad 节时，需要手动将备份后的 leader metad 的目录（路径为 `<storage_path>/meta`）拷贝并覆盖其他 follower meatd 节点的对应目录。

```
$ ./br backup full --meta "192.168.8.129:9559" --storage "local:///home/nebula/backup/"
```

- 运行以下命令对 meta 服务的地址为 `192.168.8.129:9559` 的整个集群进行全量备份操作，并将备份文件保存到兼容 s3 协议的对象存储服务 `br-test` 桶下的 `backup` 中。

```
$ ./br backup full --meta "192.168.8.129:9559" --s3.endpoint "http://192.168.8.129:9000" --storage="s3://br-test/backup/" --s3.access_key=minioadmin --s3.secret_key=minioadmin --
s3.region=default
```


2022 vesoft Inc.

以下列出命令的相关参数。

参数	数据类型	是否必需	默认值	说明
`-h`,`--help`	-	否	无	查看帮助。
`--debug`	-	否	无	查看更多日志信息。
`--log`	string	否	`"br.log"`	日志路径。
`--meta`	string	是	无	meta 服务的地址和端口号。
`--spaces`	stringArray	否	无	（实验性功能）指定要备份的图空间名字，未指定将备份所有图空间。可指定多个图空间，用法为 `--spaces nba_01 --spaces nba_02`。
`--storage`	string	是	无	BR 备份数据存储位置，格式为：`<Schema>://<PATH>` Schema：可选值为 `local` 和 `s3`。选择 s3 时，需要填写 `s3.access_key`、`s3.endpoint`、`s3.region` 和 `s3.secret_key`。 PATH：存储位置的路径。
`--s3.access_key`	string	否	无	用于标识用户。
`--s3.endpoint`	string	否	无	S3 对外服务的访问域名的 URL，指定 http 或 https。
`--s3.region`	string	否	无	数据中心所在物理位置。
`--s3.secret_key`	string	否	无	用户用于加密签名字符串和用来验证签名字符串的密钥，必须保密。

下一步

备份文件生成后，可以使用 BR 将备份文件的数据恢复到 NebulaGraph 中。具体操作，请参见使用 BR 恢复数据。

最后更新: June 20, 2023

2022 vesoft Inc.

9.1.4 使用 BR 恢复数据

如果使用 BR 备份了 Nebula Graph 的数据，可以通过备份文件进行数据恢复。本文介绍如何通过备份文件恢复数据。

Caution

恢复执行成功后，目标集群上已有的数据会被删除，然后替换为备份文件中的数据。建议提前备份目标集群上的数据。

Caution

数据恢复需要离线进行。

准备工作

- 安装 BR 和 Agent 并在集群中的每个主机上运行 Agent。
- 确认没有应用程序连接到待恢复数据的 Nebula Graph 集群。
- 确认集群的拓扑结构一致，即原集群和目标集群的主机数量一致，且每个主机数据文件夹数量分布一致。

2022 vesoft Inc.

操作步骤

在 BR 工具的安装路径下（编译安装的默认路径为 `./bin/br`），完成以下操作。

2022 vesoft Inc.

1. 用户可以使用以下命令列出现有备份信息：

```
$ ./br show --storage <storage_path>
```

例如，可以使用以下命令列出在本地 `/home/nebula/backup` 路径中的备份的信息。

```
$ ./br show --storage "local:///home/nebula/backup"
+---------------------------+---------------------+-----------------------+-------------+------------+
|           NAME            |     CREATE TIME     |        SPACES         | FULL BACKUP | ALL SPACES |
+---------------------------+---------------------+-----------------------+-------------+------------+
| BACKUP_2022_02_10_07_40_41 | 2022-02-10 07:40:41 | basketballplayer       | true        | true       |
| BACKUP_2022_02_11_08_26_43 | 2022-02-11 08:26:47 | basketballplayer,foesa | true        | true       |
+---------------------------+---------------------+-----------------------+-------------+------------+
```

或使用以下命令列出在兼容 s3 协议的对象存储服务 `br-test` 桶下的 `backup` 中的备份的信息。

```
$ ./br show --s3.endpoint "http://192.168.8.129:9000" --storage="s3://br-test/backup/" --s3.access_key=minioadmin --s3.secret_key=minioadmin --s3.region=default
```

以下列出命令的相关参数。

参数	数据类型	是否必需	默认值	说明
`-h,-help`	-	否	无	查看帮助。
`--debug`	-	否	无	查看更多日志信息。
`--log`	string	否	"br.log"	日志路径。
`--storage`	string	是	无	BR 备份数据存储位置，格式为：<Schema>://<PATH> Schema：可选值为 `local` 和 `s3`。选择 s3 时，需要填写 `s3.access_key`、`s3.endpoint`、`s3.region` 和 `s3.secret_key`。 PATH：存储位置的路径。
`--s3.access_key`	string	否	无	用于标识用户。
`--s3.endpoint`	string	否	无	S3 对外服务的访问域名的 URL，指定 http 或 https。
`--s3.region`	string	否	无	数据中心所在物理位置。
`--s3.secret_key`	string	否	无	用户用于加密签名字符串和用来验证签名字符串的密钥，必须保密。

2. 用户可以使用以下命令恢复数据：

```
$ ./br restore full --meta <ip_address> --storage <storage_path> --name <backup_name>
```

例如，可以使用以下命令，将本地 `/home/nebula/backup/` 路径中的数据恢复到为 meta 地址为 `192.168.8.129:9559` 集群中：

```
$ ./br restore full --meta "192.168.8.129:9559" --storage "local:///home/nebula/backup/" --name BACKUP_2021_12_08_18_38_08
```

或者使用以下命令，将兼容 s3 协议的对象存储服务 `br-test` 桶下的 `backup` 的备份，恢复到 meta 服务的地址为 `192.168.8.129:9559` 的集群中。

```
$ ./br restore full --meta "192.168.8.129:9559" --s3.endpoint "http://192.168.8.129:9000" --storage="s3://br-test/backup/" --s3.access_key=minioadmin --s3.secret_key=minioadmin --
s3.region="default" --name BACKUP_2021_12_08_18_38_08
```

如果返回如下信息，表示数据已经恢复成功。

```
Restore succeed.
```

Caution

如果用户新集群的IP和备份集群不同，在恢复集群后需要使用 `add host` 向新集群中添加 Storage 主机。

2022 vesoft Inc.

以下列出命令的相关参数。

参数	数据类型	是否必需	默认值	说明
`-h,-help`	-	否	-	查看帮助。
`--debug`	-	否	无	查看更多日志信息。
`--log`	string	否	"br.log"	日志路径。
`--meta`	string	是	无	meta 服务的地址和端口号。
`--name`	string	是	无	备份名字。
`--storage`	string	是	无	BR 备份数据存储位置，格式为：<Schema>://<PATH> Schema：可选值为 `local` 和 `s3`。选择 s3 时，需要填写 `s3.access_key`、`s3.endpoint`、`s3.region` 和 `s3.secret_key`。 PATH：存储位置的路径。
`--s3.access_key`	string	否	无	用于标识用户。
`--s3.endpoint`	string	否	无	S3 对外服务的访问域名的 URL，指定 http 或 https。
`--s3.region`	string	否	无	数据中心所在物理位置。
`--s3.secret_key`	string	否	无	用户用于加密签名字符串和用来验证签名字符串的密钥，必须保密。

3. 如果在备份期间发现任何错误，用户可以使用以下命令清理临时文件。该命令将清理集群和外部存储中的文件，同时用户也可以使用该命令清理外部存储中的旧的备份文件。

```
$ ./br cleanup --meta <ip_address> --storage <storage_path> --name <backup_name>
```

以下列出命令的相关参数。

参数	数据类型	是否必需	默认值	说明
`-h,-help`	-	否	-	查看帮助。
`--debug`	-	否	无	查看更多日志信息。
`--log`	string	否	"br.log"	日志路径。
`--meta`	string	是	无	meta 服务的地址和端口号。
`--name`	string	是	无	备份名字。
`--storage`	string	是	无	BR 备份数据存储位置，格式为：<Schema>://<PATH> Schema：可选值为 `local` 和 `s3`。选择 s3 时，需要填写 `s3.access_key`、`s3.endpoint`、`s3.region` 和 `s3.secret_key`。 PATH：存储位置的路径。
`--s3.access_key`	string	否	无	用于标识用户。
`--s3.endpoint`	string	否	无	S3 对外服务的访问域名的 URL，指定 http 或 https。
`--s3.region`	string	否	无	数据中心所在物理位置。
`--s3.secret_key`	string	否	无	用户用于加密签名字符串和用来验证签名字符串的密钥，必须保密。

最后更新: June 20, 2023

2022 vesoft Inc.

9.2 管理快照

Nebula Graph 提供快照（snapshot）功能，用于保存集群当前时间点的数据状态，当出现数据丢失或误操作时，可以通过快照恢复数据。

9.2.1 前提条件

Nebula Graph 的身份认证功能默认是关闭的，此时任何用户都能使用快照功能。

如果身份认证开启，仅 God 角色用户可以使用快照功能。关于角色说明，请参见内置角色权限。

9.2.2 注意事项

- 系统结构发生变化后，建议立刻创建快照，例如在 `add host` 、 `drop host` 、 `create space` 、 `drop space` 、 `balance` 等操作之后。
- 不支持自动回收创建失败的快照垃圾文件，需要手动删除。
- 不支持指定快照保存路径，默认路径为 `/usr/local/nebula/data` 。

9.2.3 快照路径

Nebula Graph 创建的快照以目录的形式存储，例如 `SNAPSHOT_2021_03_09_08_43_12` ，后缀 `2021_03_09_08_43_12` 根据创建时间（UTC）自动生成。

创建快照时，快照目录会自动在 leader Meta 服务器和所有 Storage 服务器的目录 `checkpoints` 内创建。

为了快速定位快照所在路径，可以使用 Linux 命令 `find` 。例如：

```
$ find |grep 'SNAPSHOT_2021_03_09_08_43_12'
./data/meta2/nebula/0/checkpoints/SNAPSHOT_2021_03_09_08_43_12
./data/meta2/nebula/0/checkpoints/SNAPSHOT_2021_03_09_08_43_12/data
./data/meta2/nebula/0/checkpoints/SNAPSHOT_2021_03_09_08_43_12/data/000081.sst
...
```

9.2.4 创建快照

命令 `CREATE SNAPSHOT` 可以创建集群当前时间点的快照。只支持创建所有图空间的快照，不支持创建指定图空间的快照。

> **Note**

如果快照创建失败，请删除快照重新创建。

```
nebula> CREATE SNAPSHOT;
```

9.2.5 查看快照

命令 `SHOW SNAPSHOTS` 可以查看集群中的所有快照。

```
nebula> SHOW SNAPSHOTS;
+---------------------------------+---------+-----------------+
| Name                            | Status  | Hosts           |
+---------------------------------+---------+-----------------+
| "SNAPSHOT_2021_03_09_08_43_12" | "VALID" | "127.0.0.1:9779" |
| "SNAPSHOT_2021_03_09_09_10_52" | "VALID" | "127.0.0.1:9779" |
+---------------------------------+---------+-----------------+
```


2022 vesoft Inc.

参数说明如下：

参数	说明
`Name`	快照名称，前缀为 `SNAPSHOT`，表示该文件为快照文件，后缀为快照创建的时间点（UTC 时间）。
`Status`	快照状态。`VALID` 表示快照有效，`INVALID` 表示快照无效。
`Hosts`	创建快照时所有 Storage 服务器的 IP 地址和端口。

9.2.6 删除快照

命令 `DROP SNAPSHOT` 可以删除指定的快照，语法为：

```
DROP SNAPSHOT <snapshot_name>;
```

示例如下：

```
nebula> DROP SNAPSHOT SNAPSHOT_2021_03_09_08_43_12;
nebula> SHOW SNAPSHOTS;
+---------------------------------+---------+------------------+
| Name                            | Status  | Hosts            |
+---------------------------------+---------+------------------+
| "SNAPSHOT_2021_03_09_09_10_52" | "VALID" | "127.0.0.1:9779" |
+---------------------------------+---------+------------------+
```

9.2.7 恢复快照

当前未提供恢复快照命令，需要手动拷贝快照文件到对应的文件夹内，也可以通过 shell 脚本进行操作。实现逻辑如下：

1. 创建快照后，会在 leader Meta 服务器和所有 Storage 服务器的安装目录内生成 `checkpoints` 目录，保存创建的快照。以本文为例，当存在 2 个图空间时，创建的快照分别保存在 `/usr/local/nebula/data/meta/nebula/0/checkpoints` 、`/usr/local/nebula/data/storage/nebula/3/checkpoints` 和 `/usr/local/nebula/data/storage/nebula/4/checkpoints` 中。

```
$ ls /usr/local/nebula/data/meta/nebula/0/checkpoints/
SNAPSHOT_2021_03_09_09_10_52
$ ls /usr/local/nebula/data/storage/nebula/3/checkpoints/
SNAPSHOT_2021_03_09_09_10_52
$ ls /usr/local/nebula/data/storage/nebula/4/checkpoints/
SNAPSHOT_2021_03_09_09_10_52
```

2. 当数据丢失需要通过快照恢复时，用户可以找到合适的时间点快照，将内部的文件夹 `data` 和 `wal` 分别拷贝到各自的上级目录（和 `checkpoints` 平级），覆盖之前的 `data` 和 `wal`，然后重启集群即可。

Caution

需要同时覆盖所有 Meta 节点的 data 和 wal 目录，因为存在重启集群后发生 Meta 重新选举 leader 的情况，如果不覆盖所有 Meta 节点，新的 leader 使用的还是最新的 Meta 数据，导致恢复失败。

最后更新: June 20, 2023

2022 vesoft Inc.

10. 同步与迁移

10.1 BALANCE

`BALANCE` 语句可以让 Nebula Graph 的 Storage 服务实现负载均衡。更多 `BALANCE` 语句示例和 Storage 负载均衡，请参见 Storage 负载均衡。

`BALANCE` 语法说明如下。

语法	说明
`BALANCE DATA`	启动任务均衡分布当前图空间中的所有分片。该命令会返回任务 ID（`job_id`）。
`BALANCE DATA REMOVE <ip:port> [,<ip>:<port> ...]`	启动任务迁空当前图空间指定的 Storage 服务中的分片。
`BALANCE LEADER`	启动任务均衡分布当前图空间中的所有 leader。该命令会返回任务 ID（`job_id`）。

查看、停止、重启任务，请参见作业管理。

最后更新: June 20, 2023

2022 vesoft Inc.

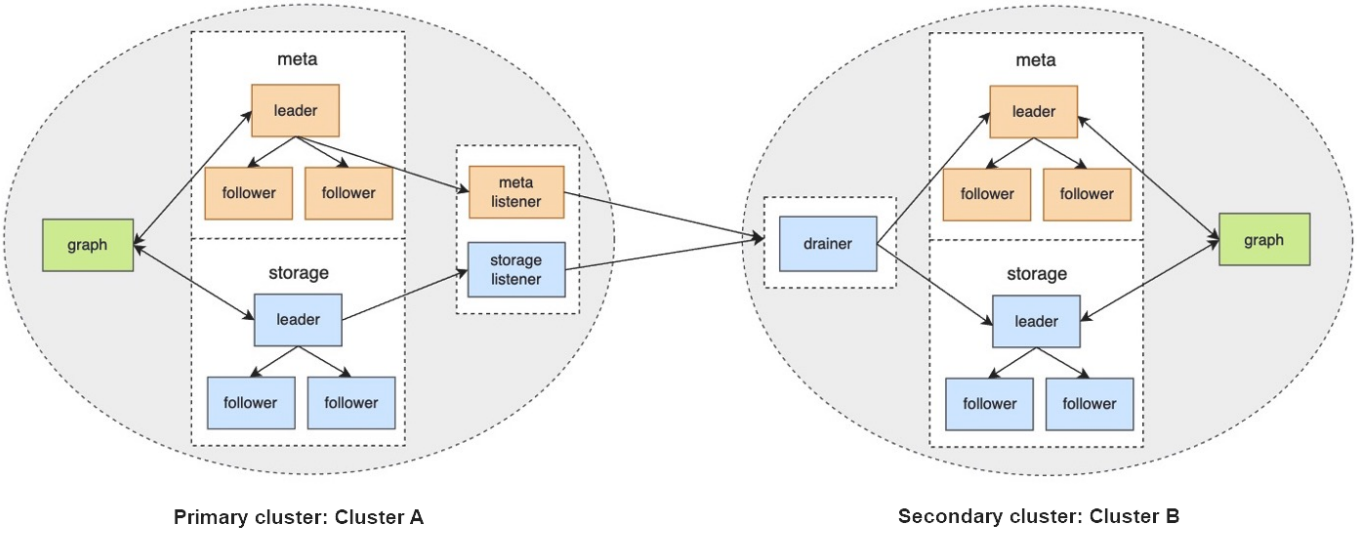
10.2 集群间数据同步

Nebula Graph 支持在集群间进行数据同步，即主集群 A 的数据可以近实时地复制到从集群 B 中，方便用户进行异地灾备或分流，降低数据丢失的风险，保证数据安全。

Enterpriseonly

仅企业版支持本功能。

10.2.1 背景



Primary cluster: Cluster A　　Secondary cluster: Cluster B

在集群间数据同步方案中，如果主集群 A 的图空间 a 和从集群 B 的图空间 b 建立了同步关系，任何向图空间 a 写入的数据，都会被发送到 Meta listener 或 Storage listener。listener 再将数据发送到 drainer。drainer 接收并存储数据，然后通过从集群的 Meta client 或 Storage client 发送数据至从集群的对应分片。

通过以上流程，最终实现集群间数据同步。

10.2.2 适用场景

- 异地灾备：通过数据同步可以实现跨机房或者跨城市的异地灾备。
- 数据迁移：通过切换主从集群的身份，可以实现不停止服务而完成迁移。
- 读写分离：通过设置主集群只写，从集群只读，实现读写分离，降低集群负载，提高稳定性和可用性。

2022 vesoft Inc.

10.2.3 注意事项

- 数据同步的基本单位是图空间，即只可以设置从一个图空间到另一个图空间的数据同步。
- 主从集群的数据同步是异步的（近实时）。
- 主从集群之间只支持1 对 1，不支持多个主集群同步到 1 个从集群，也不支持 1 个主集群同步到多个从集群，但是可以通过链式方式同步到多个从集群，例如 `主集群->从集群1->从集群2->从集群3`。
- Meta listener 监听 Meta 服务，Storage listener 监听 Storage 服务，不可以混用。
- 1 个图空间只有 1 个 Meta listener 和 1 个或多个 Storage listener，这些 listener 可以对应 1 个或多个 drainer。
- listener 服务记录来自主集群的 WAL 或快照，drainer 服务记录来自 listener 的 WAL 和写入从集群的 WAL。这些文件都保存在对应服务的本地。
- 从集群中数据如果不为空，数据同步时可能会导致数据冲突或者数据不一致。建议保持从集群数据为空。

10.2.4 操作步骤

准备工作

- 准备至少 2 台部署服务的机器。主从集群需要分开部署，listener 和 drainer 可以单独部署，也可以分别部署在主从集群所在机器上，但是会增加集群负载。
- 准备企业版 License 文件。

示例环境

主集群A：机器 IP 地址为 `192.168.10.101`，只启动 Graph、Meta、Storage 服务。

从集群B：机器 IP 地址为 `192.168.10.102`，只启动 Graph、Meta、Storage 服务。

listener：机器 IP 地址为 `192.168.10.103`，只启动 Meta listener、Storage listener 服务。

drainer：机器 IP 地址为 `192.168.10.104`，只启动 drainer 服务。

1.搭建主从集群、**listener** 和 **drainer** 服务

1. 在所有机器上安装 Nebula Graph，修改配置文件：

- 主、从集群修改：`nebula-graphd.conf`、`nebula-metad.conf`、`nebula-storaged.conf`。
- listener 修改：`nebula-metad-listener.conf`、`nebula-storaged-listener.conf`。
- drainer 修改：`nebula-drainerd.conf`。

> **Note**
>
> 修改配置文件时需要注意：
>
> - 将配置文件的后缀 `.default` 或 `.production` 删除。
> - 所有配置文件里都需要用真实的机器 IP 地址替换 `local_ip` 的 `127.0.0.1`。
> - 所有 `nebula-graphd.conf` 配置文件里设置 `enable_authorize=true`。
> - 主从集群填写各自集群的 `meta_server_addrs`，注意不要错填其他集群的地址。
> - listener 的配置文件里 `meta_server_addrs` 填写主集群的机器 IP，`meta_sync_listener` 填写 listener 机器的 IP。
> - drainer 的配置文件里 `meta_server_addrs` 填写从集群的机器 IP。
>
> 更多配置说明，请参见配置管理。

2. 在主从集群和 listener 服务的机器上放置 License 文件，路径为安装目录的 `share/resources/` 内。

2022 vesoft Inc.

3. 在所有机器的 Nebula Graph 安装目录内启动对应的服务：

- 主、从集群启动命令：`sudo scripts/nebula.service start all` 。
- listener 启动命令：
- Meta listener：`sudo bin/nebula-metad --flagfile etc/nebula-metad-listener.conf` 。
- Storage listener：`sudo bin/nebula-storaged --flagfile etc/nebula-storaged-listener.conf` 。
- drainer 启动命令：`sudo scripts/nebula-drainerd.service start` 。

4. 登录主集群增加 Storage 主机，检查 listener 服务状态。

```
nebula> ADD HOSTS 192.168.10.101:9779;
nebula> SHOW HOSTS STORAGE;
+-----------------+------+----------+-----------+--------------+----------------------+
| Host            | Port | Status   | Role      | Git Info Sha | Version              |
+-----------------+------+----------+-----------+--------------+----------------------+
| "192.168.10.101" | 9779 | "ONLINE" | "STORAGE" | "xxxxxxx"    | "ent-3.1.0"          |
+-----------------+------+----------+-----------+--------------+----------------------+

nebula> SHOW HOSTS STORAGE LISTENER;
+-----------------+------+----------+--------------------+--------------+----------------------+
| Host            | Port | Status   | Role               | Git Info Sha | Version              |
+-----------------+------+----------+--------------------+--------------+----------------------+
| "192.168.10.103" | 9789 | "ONLINE" | "STORAGE_LISTENER" | "xxxxxxx"    | "ent-3.1.0"          |
+-----------------+------+----------+--------------------+--------------+----------------------+

nebula> SHOW HOSTS META LISTENER;
+-----------------+------+----------+-----------------+--------------+----------------------+
| Host            | Port | Status   | Role            | Git Info Sha | Version              |
+-----------------+------+----------+-----------------+--------------+----------------------+
| "192.168.10.103" | 9569 | "ONLINE" | "META_LISTENER" | "xxxxxxx"    |  "ent-3.1.0"         |
+-----------------+------+----------+-----------------+--------------+----------------------+
```

5. 登录从集群增加 Storage 主机，检查 drainer 服务状态。

```
nebula> ADD HOSTS 192.168.10.102:9779;
nebula> SHOW HOSTS STORAGE;
+-----------------+------+----------+-----------+--------------+----------------------+
| Host            | Port | Status   | Role      | Git Info Sha | Version              |
+-----------------+------+----------+-----------+--------------+----------------------+
| "192.168.10.102" | 9779 | "ONLINE" | "STORAGE" | "xxxxxxx"    | "ent-3.1.0"          |
+-----------------+------+----------+-----------+--------------+----------------------+

nebula> SHOW HOSTS DRAINER;
+-----------------+------+----------+-----------+--------------+----------------------+
| Host            | Port | Status   | Role      | Git Info Sha | Version              |
+-----------------+------+----------+-----------+--------------+----------------------+
| "192.168.10.104" | 9889 | "ONLINE" | "DRAINER" | "xxxxxxx"    | "ent-3.1.0"          |
+-----------------+------+----------+-----------+--------------+----------------------+
```

2.设置服务

1. 登录主集群，创建图空间 `basketballplayer` 。

```
nebula> CREATE SPACE basketballplayer(partition_num=15, replica_factor=1, vid_type=fixed_string(30));
```

2. 进入图空间 `basketballplayer` ，注册 drainer 服务。

```
nebula> USE basketballplayer;
//注册 drainer 服务。
nebula> SIGN IN DRAINER SERVICE(192.168.10.104:9889);
//检查是否注册成功。
nebula> SHOW DRAINER CLIENTS;
+-----------+-----------------+------+
| Type      | Host            | Port |
+-----------+-----------------+------+
| "DRAINER" | "192.168.10.104" | 9889 |
+-----------+-----------------+------+
```

3. 设置 listener 服务。

```
//设置 listener 服务，待同步的图空间名称为replication_basketballplayer（下文将在从集群中创建）。
nebula> ADD LISTENER SYNC META 192.168.10.103:9569 STORAGE 192.168.10.103:9789 TO SPACE replication_basketballplayer;
//查看 listener 状态。
nebula> SHOW LISTENER SYNC;
+--------+--------+-------------------------+--------------------------------+----------+
| PartId | Type   | Host                    | SpaceName                      | Status   |
+--------+--------+-------------------------+--------------------------------+----------+
```


2022 vesoft Inc.

```
| 0      | "SYNC" | ""192.168.10.103":9569" | "replication_basketballplayer" | "ONLINE" |
| 1      | "SYNC" | ""192.168.10.103":9789" | "replication_basketballplayer" | "ONLINE" |
| 2      | "SYNC" | ""192.168.10.103":9789" | "replication_basketballplayer" | "ONLINE" |
| 3      | "SYNC" | ""192.168.10.103":9789" | "replication_basketballplayer" | "ONLINE" |
| 4      | "SYNC" | ""192.168.10.103":9789" | "replication_basketballplayer" | "ONLINE" |
| 5      | "SYNC" | ""192.168.10.103":9789" | "replication_basketballplayer" | "ONLINE" |
| 6      | "SYNC" | ""192.168.10.103":9789" | "replication_basketballplayer" | "ONLINE" |
| 7      | "SYNC" | ""192.168.10.103":9789" | "replication_basketballplayer" | "ONLINE" |
| 8      | "SYNC" | ""192.168.10.103":9789" | "replication_basketballplayer" | "ONLINE" |
| 9      | "SYNC" | ""192.168.10.103":9789" | "replication_basketballplayer" | "ONLINE" |
| 10     | "SYNC" | ""192.168.10.103":9789" | "replication_basketballplayer" | "ONLINE" |
| 11     | "SYNC" | ""192.168.10.103":9789" | "replication_basketballplayer" | "ONLINE" |
| 12     | "SYNC" | ""192.168.10.103":9789" | "replication_basketballplayer" | "ONLINE" |
| 13     | "SYNC" | ""192.168.10.103":9789" | "replication_basketballplayer" | "ONLINE" |
| 14     | "SYNC" | ""192.168.10.103":9789" | "replication_basketballplayer" | "ONLINE" |
| 15     | "SYNC" | ""192.168.10.103":9789" | "replication_basketballplayer" | "ONLINE" |
+--------+--------+-------------------------+--------------------------------+----------+
```

4. 登录从集群，创建图空间 `replication_basketballplayer` 。

```
nebula> CREATE SPACE replication_basketballplayer(partition_num=15, replica_factor=1, vid_type=fixed_string(30));
```

5. 进入图空间 `replication_basketballplayer` ，设置 drainer 服务。

```
nebula> USE replication_basketballplayer;
//设置 drainer 服务。
nebula> ADD DRAINER 192.168.10.104:9889;
//查看 drainer 状态。
nebula> SHOW DRAINERS;
+-------------------------+----------+
| Host                    | Status   |
+-------------------------+----------+
| ""192.168.10.104":9889" | "ONLINE" |
+-------------------------+----------+
```

6. 修改图空间 `replication_basketballplayer` 为只读。

> **Note**
>
> 修改为只读是防止误操作导致数据不一致。只影响该图空间，其他图空间仍然可以读写。

```
//设置当前图空间为只读。
nebula> SET VARIABLES read_only=true;
//查看当前图空间的读写属性。
nebula> GET VARIABLES read_only;
+-------------+--------+-------+
| name        | type   | value |
+-------------+--------+-------+
| "read_only" | "bool" | true  |
+-------------+--------+-------+
```

3.验证数据

1. 登录主集群，创建 Schema，插入数据。

```
nebula> USE basketballplayer;
nebula> CREATE TAG player(name string, age int);
nebula> CREATE EDGE follow(degree int);
nebula> INSERT VERTEX player(name, age) VALUES "player100":("Tim Duncan", 42);
nebula> INSERT VERTEX player(name, age) VALUES "player101":("Tony Parker", 36);
nebula> INSERT EDGE follow(degree) VALUES "player101" -> "player100":(95);
```

2. 登录从集群，检查数据。

```
nebula> USE replication_basketballplayer;
nebula> SUBMIT JOB STATS;
nebula> SHOW STATS;
+---------+------------+-------+
| Type    | Name       | Count |
+---------+------------+-------+
| "Tag"   | "player"   | 2     |
| "Edge"  | "follow"   | 1     |
| "Space" | "vertices" | 2     |
| "Space" | "edges"    | 1     |
+---------+------------+-------+

nebula> FETCH PROP ON player "player100" YIELD properties(vertex);
+------------------------------+
| properties(VERTEX)           |
```


2022 vesoft Inc.

```
+--------------------------------+
| {age: 42, name: "Tim Duncan"} |
+--------------------------------+

nebula> GO FROM "player101" OVER follow YIELD dst(edge);
+-------------+
| dst(EDGE)   |
+-------------+
| "player100" |
+-------------+
```

10.2.5 停止/重启数据同步

数据同步时，listener 会持续发送 WAL 给 drainer。

如果需要停止数据同步，可以使用 `stop sync` 命令。此时 listener 会停止向 drainer 发送 WAL。

如果需要重启数据同步，可以使用 `restart sync` 命令。此时 listener 会向 drainer 发送停止期间堆积的 WAL。如果 listener 上的 WAL 丢失，listener 会从主集群拉取快照重新进行同步。

10.2.6 切换主从集群

如果因为业务需要进行数据迁移，或者灾备恢复后需要切换主从集群，需要手动进行切换。

> **Note**
>
> 在切换主从之前需要为新的主集群搭建并启动 listener 服务（示例 IP 为 `192.168.10.105`），为新的从集群搭建并启动 drainer 服务（示例 IP 为 `192.168.10.106`）。

1. 登录主集群，取消 drainer 和 listener 服务。

```
nebula> USE basketballplayer;
nebula> SIGN OUT DRAINER SERVICE;
nebula> REMOVE LISTENER SYNC;
```

2. 设置图空间为只读，防止有新的数据写入主集群，导致数据不一致。

```
nebula> SET VARIABLES read_only=true;
```

3. 登录从集群，设置图空间为可读写，取消 drainer。

```
nebula> USE replication_basketballplayer;
nebula> SET VARIABLES read_only=false;
nebula> REMOVE DRAINER;
```

4. 将从集群更改为主集群。

```
nebula> SIGN IN DRAINER SERVICE(192.168.10.106:9889);
nebula> ADD LISTENER SYNC META 192.168.10.105:9569 STORAGE 192.168.10.105:9789 TO SPACE basketballplayer;
nebula> REMOVE DRAINER;
```

5. 登录之前的主集群，将其更改为从集群。

```
nebula> USE basketballplayer;
//修改图空间为可读写，否则无法设置 drainer 服务。
nebula> SET VARIABLES read_only=false;
nebula> ADD DRAINER 192.168.10.106:9889;
nebula> SET VARIABLES read_only=true;
```

10.2.7 常见问题

主集群中已经有 **data** 了，从集群可以同步到之前的存量 **data** 吗?

可以。对于主集群中的存量 data，主集群的 listener 会从各个分片的 leader 节点拉取快照，然后以 WAL 的形式发送给 drainer。存量 data 相关的 WAL 发送完毕后，开始发送主集群的增量 data 相关的 WAL 给 drainer。

2022 vesoft Inc.

从集群中已经有 **data** 了，数据同步会有影响吗?

仍然会进行全量数据同步。如果从集群中的数据是主集群数据的子集，最终会数据一致；如果不是主集群数据的子集，从集群不会进行反向同步，而且这部分数据可能会受到影响，请确保主从集群数据不会冲突；建议保持从集群数据为空。

从集群中已经有 **Schema** 了，数据同步会有影响吗?

数据同步时主集群中的 Schema 会覆盖从集群中的 Schema，因此可能导致被覆盖的 Schema 对应的数据失效或者数据不一致。请确保从集群中的 Schema 和主集群中的 Schema 没有冲突。

修改主集群的 **Schema** 会影响数据同步吗?

可能会增加数据同步延迟。因为 Schema 数据和 data 数据是分开处理的（Meta listener 和 Storage listener），data 数据同步时，drainer 会检查自身的 Schema 版本，如果版本大于当前存储的版本，说明 Schema 有更新，这时候会暂缓更新，等待从集群中的 Schema 数据先更新完成。

主从集群的机器数量、副本数量、分片数量需要相同吗?

不需要。因为是以图空间为基本单位，主集群不需要知道从集群的架构信息，只需要 listener 将 WAL 发送给 drainer 即可。

如果同步时出现故障，如何修复?

可以根据故障节点，进行如下处理:

- 主集群故障：会导致同步暂停，重启主集群服务即可。
- listener/drainer/从集群故障：服务恢复后，会收到前一节点发送的故障期间的 WAL。例如 drainer 故障恢复后，会收到 listener 发送来的故障期间的 WAL。如果用新的节点替换故障的 drainer 节点或从集群，需要将原节点的数据复制到新节点，否则相当于重新同步全量数据。

如何判断数据同步进度?

暂无工具进行直接判断。

最后更新: June 20, 2023

2022 vesoft Inc.

11. 最佳实践

11.1 Compaction

本文介绍 Compaction 的相关信息。

Nebula Graph 中，`Compaction` 是最重要的后台操作，对性能有极其重要的影响。

`Compaction` 操作会读取硬盘上的数据，然后重组数据结构和索引，然后再写回硬盘，可以成倍提升读取性能。将大量数据写入 Nebula Graph 后，为了提高读取性能，需要手动触发 `Compaction` 操作（全量 `Compaction`）。

> **Note**
>
> `Compaction` 操作会长时间占用硬盘的 IO，建议在业务低峰期（例如凌晨）执行该操作。

Nebula Graph 有两种类型的 `Compaction` 操作：自动 `Compaction` 和全量 `Compaction`。

11.1.1 自动 `Compaction`

自动 `Compaction` 是在系统读取数据、写入数据或系统重启时自动触发 `Compaction` 操作，提升短时间内的读取性能。默认情况下，自动 `Compaction` 是开启状态，可能在业务高峰期触发，导致意外抢占 IO 影响业务。

11.1.2 全量 `Compaction`

全量 `Compaction` 可以对图空间进行大规模后台操作，例如合并文件、删除 TTL 过期数据等，该操作需要手动发起。使用如下语句执行全量 `Compaction` 操作：

> **Note**
>
> 建议在业务低峰期（例如凌晨）执行该操作，避免大量占用硬盘 IO 影响业务。

```
nebula> USE <your_graph_space>;
nebula> SUBMIT JOB COMPACT;
```

上述命令会返回作业的 ID，用户可以使用如下命令查看 `Compaction` 状态：

```
nebula> SHOW JOB <job_id>;
```

11.1.3 操作建议

为保证 Nebula Graph 的性能，请参考如下操作建议：

- 数据导入完成后，请执行 `SUBMIT JOB COMPACT`。
- 业务低峰期（例如凌晨）执行 `SUBMIT JOB COMPACT`。
- 为控制 `Compaction` 的写入速率，请在配置文件 `nebula-storaged.conf` 中设置如下参数（注：此参数限制全部写入，包括正常写入和 Compaction）：

```
# 写入速度限制为 20MB/S。
--rocksdb_rate_limit=20 (in MB/s)
```


2022 vesoft Inc.

11.1.4 FAQ

Compaction 相关的日志在哪?

默认情况下，`/usr/local/nebula/data/storage/nebula/{1}/data/` 目录下的文件名为 `LOG` 文件，或者类似 `LOG.old.1625797988509303`，找到如下的部分。

```
** Compaction Stats [default] **
Level    Files   Size     Score Read(GB)  Rn(GB) Rnp1(GB) Write(GB) Wnew(GB) Moved(GB) W-Amp Rd(MB/s) Wr(MB/s) Comp(sec) CompMergeCPU(sec) Comp(cnt) Avg(sec) KeyIn KeyDrop
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
  L0      2/0    2.46 KB   0.5      0.0     0.0      0.0       0.0      0.0       0.0   1.0      0.0      0.0      0.53              0.51         2    0.264       0      0
 Sum      2/0    2.46 KB   0.0      0.0     0.0      0.0       0.0      0.0       0.0   1.0      0.0      0.0      0.53              0.51         2    0.264       0      0
 Int      0/0    0.00 KB   0.0      0.0     0.0      0.0       0.0      0.0       0.0   0.0      0.0      0.0      0.00              0.00         0    0.000       0      0
```

如果当前的 `L0` 文件数量较多，对读性能影响较大，可以触发 compaction。

可以同时在多个图空间执行全量 **`Compaction`** 操作吗?

可以，但是此时的硬盘 IO 会很高，可能会影响效率。

全量 **`Compaction`** 操作会耗费多长时间?

如果已经设置读写速率限制，例如 `rocksdb_rate_limit` 限制为 20MB/S 时，用户可以通过 `硬盘使用量/rocksdb_rate_limit` 预估需要耗费的时间。 如果没有设置读写速率限制，根据经验，速率大约为 50MB/S。

可以动态调整 **`rocksdb_rate_limit`** 吗?

不可以。

全量 **`Compaction`** 操作开始后可以停止吗?

不可以停止，必须等待操作完成。这是 RocksDB 的限制。

最后更新: June 20, 2023

2022 vesoft Inc.

11.2 Storage 负载均衡

用户可以使用 `BALANCE` 语句平衡分片和 Raft leader 的分布，或者清空某些 Storage 服务器方便进行维护。详情请参见 BALANCE。

Danger

`BALANCE` 命令通过创建和执行一组子任务来迁移数据和均衡分片分布，禁止停止集群中的任何机器或改变机器的 IP 地址，直到所有子任务完成，否则后续子任务会失败。

11.2.1 均衡分片分布

Enterpriseonly

仅企业版支持均衡分片分布。

Note

如果当前图空间已经有失败的均衡分片分布作业，无法开始新的均衡分片分布作业，只能恢复之前失败的作业。如果作业一直执行失败，可以先停止作业，再开始新的均衡分片分布作业。

`BALANCE DATA` 语句会开始一个作业，将当前图空间的分片平均分配到所有 Storage 服务器。通过创建和执行一组子任务来迁移数据和均衡分片分布。

2022 vesoft Inc.

示例

以横向扩容 Nebula Graph 为例，集群中增加新的 Storage 主机后，新主机上没有分片。

1. 执行命令 `SHOW HOSTS` 检查分片的分布。

```
nebual> SHOW HOSTS;
+-----------------+------+-----------+----------+--------------+----------------------+-----------------------+-------------+
| Host            | Port | HTTP port | Status   | Leader count | Leader distribution   | Partition distribution | Version     |
+-----------------+------+-----------+----------+--------------+----------------------+-----------------------+-------------+
| "192.168.8.101" | 9779 | 19669     | "ONLINE" | 0            | "No valid partition"  | "No valid partition"   | "3.1.0-ent" |
| "192.168.8.100" | 9779 | 19669     | "ONLINE" | 15           | "basketballplayer:15" | "basketballplayer:15"  | "3.1.0-ent" |
+-----------------+------+-----------+----------+--------------+----------------------+-----------------------+-------------+
```

2. 进入图空间 `basketballplayer`，然后执行命令 `BALANCE DATA` 将所有分片均衡分布。

```
nebula> USE basketballplayer;
nebula> BALANCE DATA;
+------------+
| New Job Id |
+------------+
| 2          |
+------------+
```

3. 根据返回的任务ID，执行命令 `SHOW JOB <job_id>` 检查任务状态。

```
nebula> SHOW JOB 2;
+------------------------+-----------------------------------------+-------------+-------------------------------+-------------------------------+-------------+
| Job Id(spaceId:partId) | Command(src->dst)                       | Status      | Start Time                    | Stop Time                     | Error Code  |
+------------------------+-----------------------------------------+-------------+-------------------------------+-------------------------------+-------------+
| 2                      | "DATA_BALANCE"                          | "FINISHED"  | "2022-04-12T03:41:43.000000000" | "2022-04-12T03:41:53.000000000" | "SUCCEEDED" |
| "2, 1:1"               | "192.168.8.100:9779->192.168.8.101:9779" | "SUCCEEDED" | 2022-04-12T03:41:43.000000      | 2022-04-12T03:41:53.000000      | "SUCCEEDED" |
| "2, 1:2"               | "192.168.8.100:9779->192.168.8.101:9779" | "SUCCEEDED" | 2022-04-12T03:41:43.000000      | 2022-04-12T03:41:53.000000      | "SUCCEEDED" |
| "2, 1:3"               | "192.168.8.100:9779->192.168.8.101:9779" | "SUCCEEDED" | 2022-04-12T03:41:43.000000      | 2022-04-12T03:41:53.000000      | "SUCCEEDED" |
| "2, 1:4"               | "192.168.8.100:9779->192.168.8.101:9779" | "SUCCEEDED" | 2022-04-12T03:41:43.000000      | 2022-04-12T03:41:53.000000      | "SUCCEEDED" |
| "2, 1:5"               | "192.168.8.100:9779->192.168.8.101:9779" | "SUCCEEDED" | 2022-04-12T03:41:43.000000      | 2022-04-12T03:41:53.000000      | "SUCCEEDED" |
| "2, 1:6"               | "192.168.8.100:9779->192.168.8.101:9779" | "SUCCEEDED" | 2022-04-12T03:41:43.000000      | 2022-04-12T03:41:43.000000      | "SUCCEEDED" |
| "2, 1:7"               | "192.168.8.100:9779->192.168.8.101:9779" | "SUCCEEDED" | 2022-04-12T03:41:43.000000      | 2022-04-12T03:41:53.000000      | "SUCCEEDED" |
| "Total:7"              | "Succeeded:7"                           | "Failed:0"  | "In Progress:0"                 | "Invalid:0"                   | ""          |
+------------------------+-----------------------------------------+-------------+-------------------------------+-------------------------------+-------------+
```

4. 等待所有子任务完成，负载均衡进程结束，执行命令 `SHOW HOSTS` 确认分片已经均衡分布。

Note

`BALANCE DATA` 不会均衡 leader 的分布。均衡 leader 请参见均衡 leader 分布。

```
nebula> SHOW HOSTS;
+-----------------+------+-----------+----------+--------------+----------------------+-----------------------+-------------+
| Host            | Port | HTTP port | Status   | Leader count | Leader distribution  | Partition distribution | Version     |
+-----------------+------+-----------+----------+--------------+----------------------+-----------------------+-------------+
| "192.168.8.101" | 9779 | 19669     | "ONLINE" | 7            | "basketballplayer:7" | "basketballplayer:7"   | "3.1.0-ent" |
| "192.168.8.100" | 9779 | 19669     | "ONLINE" | 8            | "basketballplayer:8" | "basketballplayer:8"   | "3.1.0-ent" |
+-----------------+------+-----------+----------+--------------+----------------------+-----------------------+-------------+
```

如果有子任务失败，请执行 `RECOVER JOB <job_id>`。如果重做负载均衡仍然不能解决问题，请到Nebula Graph社区寻求帮助。

停止负载均衡作业

停止负载均衡作业，请执行命令 `STOP JOB <job_id>`。

- 如果没有正在执行的负载均衡作业，会返回错误。
- 如果有正在执行的负载均衡作业，会返回 `Job stopped`。

Note

`STOP JOB <job_id>` 不会停止正在执行的子任务，而是取消所有后续子任务，状态会置为 `INVALID`，然后等待正在执行的子任执行完毕根据结果置为 `SUCCEEDED` 或 `FAILED`。用户可以执行命令 `SHOW JOB <job_id>` 检查停止的作业状态。

2022 vesoft Inc.

恢复负载均衡作业

恢复负载均衡作业，请执行命令 `RECOVER JOB <job_id>` 。

> **Note**

- 可以恢复执行失败的作业。
- 对于停止的作业，Nebula Graph 会判断该作业的开始时间（start time）之后是否有相同类型的失败作业（failed job）或完成作业（finished job），如果有的话，无法恢复停止的作业。例如当有 `stopped job1 -> finished job2 -> stopped job3` 时，只能恢复 job3，无法恢复 job1。

迁移分片

迁移指定的 Storage 主机中的分片来缩小集群规模，可以使用命令 `BALANCE DATA REMOVE <ip:port> [,<ip>:<port> ...]` 。

例如需要迁移 `192.168.8.100:9779` 中的分片，请执行如下命令：

```
nebula> BALANCE DATA REMOVE 192.168.8.100:9779;
nebula> SHOW HOSTS;
+-----------------+------+-----------+----------+--------------+----------------------+------------------------+------------+
| Host            | Port | HTTP port | Status   | Leader count | Leader distribution  | Partition distribution | Version    |
+-----------------+------+-----------+----------+--------------+----------------------+------------------------+------------+
| "192.168.8.101" | 9779 | 19669     | "ONLINE" | 15           | "basketballplayer:15" | "basketballplayer:15"  | "3.1.0-ent" |
| "192.168.8.100" | 9779 | 19669     | "ONLINE" | 0            | "No valid partition"  | "No valid partition"   | "3.1.0-ent" |
+-----------------+------+-----------+----------+--------------+----------------------+------------------------+------------+
```

> **Note**

该命令仅迁移分片，不会将 Storage 主机从集群中删除。删除 Storage 主机请参见管理 Storage 主机。

11.2.2 均衡 leader 分布

用户可以使用命令 `BALANCE LEADER` 均衡 leader 分布。

示例

```
nebula> BALANCE LEADER;
```

用户可以执行 `SHOW HOSTS` 检查结果。

```
nebula> SHOW HOSTS;
+-----------------+------+-----------+----------+--------------+----------------------------------+------------------------+---------+
| Host            | Port | HTTP port | Status   | Leader count | Leader distribution              | Partition distribution | Version |
+-----------------+------+-----------+----------+--------------+----------------------------------+------------------------+---------+
| "192.168.10.100" | 9779 | 19669     | "ONLINE" | 4            | "basketballplayer:3"             | "basketballplayer:8"   | "3.1.0" |
| "192.168.10.101" | 9779 | 19669     | "ONLINE" | 8            | "basketballplayer:3"             | "basketballplayer:8"   | "3.1.0" |
| "192.168.10.102" | 9779 | 19669     | "ONLINE" | 3            | "basketballplayer:3"             | "basketballplayer:8"   | "3.1.0" |
| "192.168.10.103" | 9779 | 19669     | "ONLINE" | 0            | "basketballplayer:2"             | "basketballplayer:7"   | "3.1.0" |
| "192.168.10.104" | 9779 | 19669     | "ONLINE" | 0            | "basketballplayer:2"             | "basketballplayer:7"   | "3.1.0" |
| "192.168.10.105" | 9779 | 19669     | "ONLINE" | 0            | "basketballplayer:2"             | "basketballplayer:7"   | "3.1.0" |
+-----------------+------+-----------+----------+--------------+----------------------------------+------------------------+---------+
```

> **Caution**

在 Nebula Graph 3.1.0 中，Leader 切换会导致短时的大量请求错误（Storage Error `E_RPC_FAILURE` ），处理方法见 FAQ。

最后更新: June 20, 2023

2022 vesoft Inc.

11.3 图建模设计

本文介绍在 Nebula Graph 项目中成功应用的一些图建模和系统设计的通用建议。

Note

本文建议是通用的，在特定领域有例外，请结合实际业务情况进行图建模。

11.3.1 以性能为第一目标进行建模

目前 Nebula Graph 没有完美的建模方法，如何建模取决于想从数据中挖掘的内容。分析数据并根据业务模型创建方便直观的数据模型，测试模型并优化，逐渐适应业务。为了更好的性能，用户可以多次更改或重新设计模型。

设计和评估最重要的查询语句

在测试环节中，通常会验证各种各样的查询语句，以全面评估系统能力。但在大多数生产场景下，每个集群被频繁调用的查询语句的类型并不会太多；根据 20-80 原则，针对重要的查询语句进行建模优化。

避免全量扫描

通过属性索引或者 VID 来先定位到某个（些）点或者边，然后开始图遍历；对于有些查询，它们只给定了一个子图或者路径的（正则）模式，但无法通过属性索引或者 VID 定位到遍历起始的点边，而期望找到库中全部满足该模式的子图，这样的查询是通过全量扫描实现的，这样的性能会很差。Nebula Graph 没有实现对于子图或者路径的图结构的索引。

Tag 与 **Edge type** 之间没有绑定关系

任何 Tag 可以与任何 Edge type 相关联，完全交由应用程序控制。不需要在 Nebula Graph 中预先定义，也没有命令获取哪些 Tag 与哪些 Edge type 相关联。

Tag/Edge type 预先定义了一组属性

建立 Tag（或者 Edge type）时，需要指定对应的属性。通常称为 Schema。

区分“经常改变的部分”和“不经常改变的部分”

改变指的是业务模型和数据模型上的改变（元信息），不是数据自身的改变。

一些图数据库产品是 schema-free 的设计，所以在数据模型上，不论是图拓扑结构还是属性，都可以非常自由。属性可以建模转变为图拓扑，反之亦然。这类系统通常对于图拓扑的访问有特别的优化。

而 Nebula Graph 3.1.0 是强 Schema 的（行存型）系统，这意味着业务数据模型中的部分是“不应该经常改变的”，例如属性 Schema 应该避免改变。类似于 MySQL 中 ALTER TABLE 是应该尽量避免的操作。

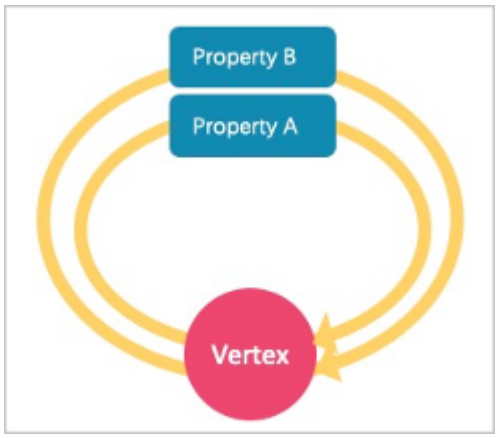
而点及邻边可以非常低成本的增删，因此可以将业务模型中“经常改变的部分”建模成点或边（关系），而不是属性 Schema。

例如，在一个业务模型中，人的属性是相对固定的，例如“年龄”，“性别”，“姓名”。而“通信好友”，“出入场所”，“交易账号”，“登录设备”等是相对容易改变的。前者适合建模为属性，后者适合建模为点或边。

自环

Nebula Graph是强Schema类型系统，使用 `ALTER TAG` 的开销很大，而且也不支持List类型属性，当用户需要为点添加一些临时属性或者List类型的属性时，可以先创建包含所需属性的边类型，然后为点插入一条或多条指向自身的边。查询时只需要查询指向自己的边属性。如下图所示。

2022 vesoft Inc.



示例：

如果要检索点的临时属性，请从自环边中获取。例如：

```
//创建边类型并插入自环属性。
nebula> CREATE EDGE IF NOT EXISTS temp(tmp int);
nebula> INSERT EDGE temp(tmp) VALUES "player100"->"player100"@1:(1);
nebula> INSERT EDGE temp(tmp) VALUES "player100"->"player100"@2:(2);
nebula> INSERT EDGE temp(tmp) VALUES "player100"->"player100"@3:(3);

//插入数据后，可以通过查询语句查询，例如：
nebula> GO FROM "player100" OVER temp YIELD properties(edge).tmp;
+----------------------+
| properties(EDGE).tmp |
+----------------------+
| 1                    |
| 2                    |
| 3                    |
+----------------------+

//如果需要返回结果为List，可以通过函数实现，例如：
nebula> MATCH (v1:player)-[e:temp]->() return collect(e.tmp);
+---------------+
| collect(e.tmp) |
+---------------+
| [1, 2, 3]      |
+---------------+
```

对于自环的操作没有封装任何的语法糖，操作方式与普通的边无异。

- Nebula Graph 自环小科普（2 分 54 秒）

悬挂边

悬挂边 (Dangling edge) 是指一条起点或者终点不存在于数据库中的边。

在 Nebula Graph 3.1.0中，有两种情况可能会出现悬挂边。

第一种情况：在起点和终点插入之前，用 INSERT EDGE 语句插入一条边。

第二种情况：使用 DELETE VERTEX 语句删除点的时候，没有使用 `WITH EDGE` 选项。此时系统默认不删除该点关联的出边和入边，这些边将变成悬挂边。

Nebula Graph 3.1.0 的数据模型中，由于设计允许图中存在“悬挂边”；没有 openCypher 中的 MERGE 语句。对于悬挂边的保证完全依赖应用层面。用户可以使用 GO 和 LOOKUP 语句查询到悬挂边，但无法使用 MATCH 语句查询到悬挂边。

示例：

```
// 插入起点为“11”，终点为“13”并且都不存在于数据库中的悬挂边
nebula> CREATE EDGE IF NOT EXISTS e1 (name string, age int);
nebula> INSERT EDGE e1 (name, age) VALUES "11"->"13":("n1", 1);

// 使用 GO 语句查询
nebula> GO FROM "11" over e1 YIELD properties(edge);
+----------------------+
| properties(EDGE)     |
```


2022 vesoft Inc.

```
+----------------------+
| {age: 1, name: "n1"} |
+----------------------+

// 使用 LOOKUP 语句查询
nebula> LOOKUP ON e1 YIELD EDGE AS r;
+-------------------------------------------------------+
| r                                                     |
+-------------------------------------------------------+
| [:e2 "11"->"13" @0 {age: 1, name: "n1"}]              |
+-------------------------------------------------------+

// 使用 MATCH 查询，不能查询到悬挂边
nebula> MATCH ()-[e:e1]->() RETURN e LIMIT 100;
+---+
| e |
+---+
+---+
Empty set (time spent 3153/3573 us)
```

- Nebula Graph 的悬挂边小科普（2 分 28 秒）

广度优先大于深度优先

- Nebula Graph 基于图拓扑结构进行深度图遍历的性能较低，广度优先遍历以及获取属性的性能较好。例如，模型 a 包括姓名、年龄、眼睛颜色三种属性，建议创建一个 Tag `person`，然后为它添加姓名、年龄、眼睛颜色的属性。如果创建一个包含眼睛颜色的 Tag 和一个 Edge type `has`，然后创建一个边用来表示人拥有的眼睛颜色，这种建模方法会降低遍历性能。
- “通过边属性获取边”的性能与“通过点属性获取点”的性能是接近的。在一些数据库中，会建议将边上的属性重新建模为中间节点的属性：例如 `(src)-[edge {P1, P2}]->(dst)`，`edge` 上有属性 `P1, P2`，会建议建模为 `(src)-[edge1]->(i_node {P1, P2})-[edge2]->(dst)`。在 Nebula Graph 3.1.0 中可以直接使用 `(src)-[edge {P1, P2}]->(dst)`，减少遍历深度有助于性能。

边的方向

查询时，如果需要使用边的逆向查询，可以用如下语法：

`(dst)<-[edge]-(src)` 或者 `GO FROM dst REVERSELY;`

如果不关心边的方向，可以使用如下语法：

`(src)-[edge]-(dst)` 或者 `GO FROM src BIDIRECT;`

因此，通常同一条边没有必要反向再冗余插入一次。

合理设置 **Tag** 属性

在图建模中，请将一组类似的平级属性放入同一个 Tag，即按不同概念进行分组。

正确使用索引

使用属性索引可以通过属性查找到 VID。但是索引会导致写性能下降 90%甚至更多，只有在根据点或边的属性定位点或边时才使用索引。

合理设计 **VID**

参考点 VID 一节。

长文本

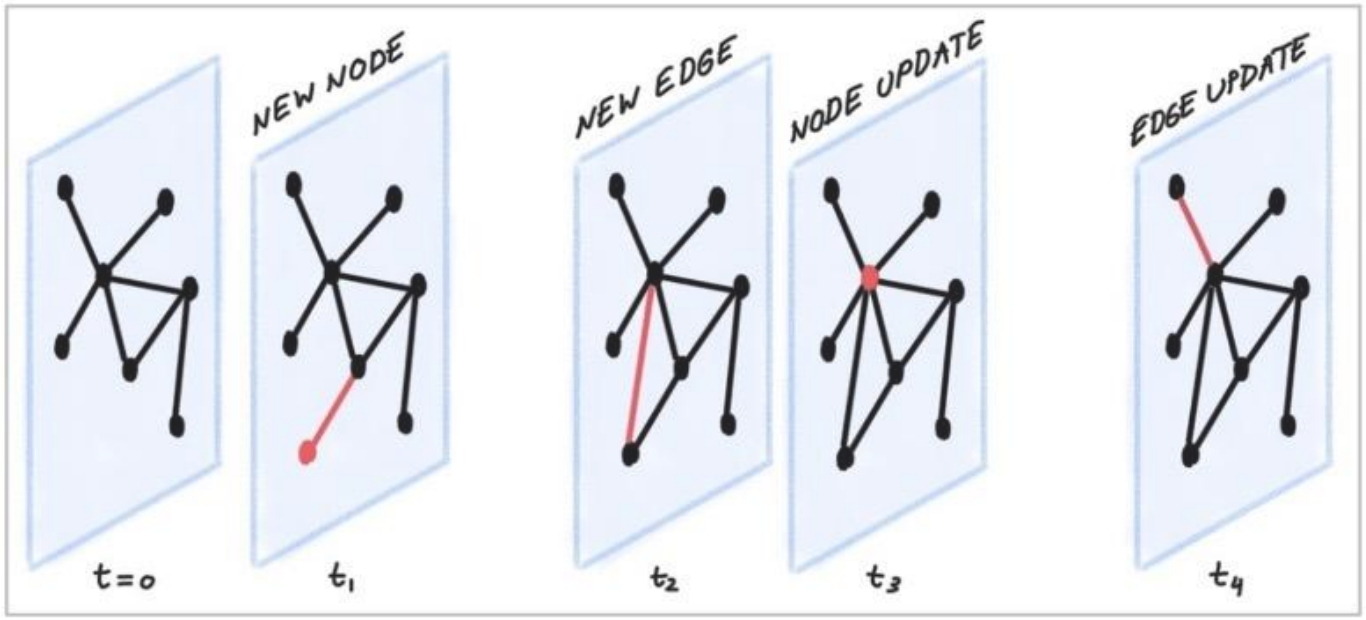
为边创建属性时请勿使用长文本：这些属性会被存储 2 份，导致写入放大问题（write amplification）。此时建议将长文本放在 HBase/ES 中，将其地址存放在 Nebula Graph 中。

11.3.2 关于支持动态图（时序图）

在某些场景下，图需要同时带有时序信息，以描述整个图的结构随着时间变化的情况 [1]。

2022 vesoft Inc.

Nebula Graph 3.1.0 的边可以使用 Rank 字段存放时间信息 (int64)，但是点上没有字段可以存放时间信息（存放在属性会被新写入覆盖）。一个折中的办法是在点上设计自己指向自己的自环，并将时间信息放置在自环的 Rank 上。



1. https://blog.twitter.com/engineering/en_us/topics/insights/2021/temporal-graph-networks ↩

最后更新: June 20, 2023

2022 vesoft Inc.

11.4 系统设计建议

11.4.1 选择 QPS 优先或时延优先

- Nebula Graph 3.1.0 更擅长处理（互联网式的）有大量并发的小请求。也即：虽然全图很大（万亿点边），但是每个请求要访问到的子图本身并不大（几百万个点边）——单个请求时延不大；但这类请求的并发数量特别多——QPS 大。
- 但对于一些交互分析型的场景，并发请求的数量不多，而每个请求要访问的子图本身特别大（亿以上）。为降低时延，可以在应用程序中将一个大的请求，拆分为多个小请求，并发发送给多个 graphd。这样可以降低单个大请求的时延，降低单个 graphd 的内存占用。另外，也可以使用图计算功能 Nebula Algorithm。

11.4.2 数据传输与优化

- 读写平衡。Nebula Graph 适合读写平衡性的在线场景，也即 OLTP 型的的“并发的发生写入与读取”；而非数仓 OLAP 型的“一次写入多次读取”。
- 选择不同的写入方式。大批量的数据写入可以使用 sst 加载的方式；小批量的写入使用 `INSERT` 语句。
- 选择合适的时间运行 COMPACTION 和 BALANCE，来分别优化数据格式和存储分布。
- Nebula Graph 3.1.0 不支持关系型数据库意义上的事务和隔离性，更接近 NoSQL。

11.4.3 查询预热与数据预热

应用端进行预热：

- Graphd 不支持预编译查询及相应生成查询计划，也不支持缓存之前的查询结果；
- Storaged 不支持预热数据，只有 RocksDB 自身的 LSM-tree 和 BloomFilter 会启动时加载到内存中。
- 点和边被访问过后，会各自缓存在 Storaged 的两种 (LRU) Cache 中。

最后更新: June 20, 2023

2022 vesoft Inc.

11.5 执行计划

Nebula Graph 3.1.0 实现了基于规则的执行计划。用户无法改变执行计划，无法进行查询的预编译（及相应的计划缓存），无法通过指定索引来加速查询。

要查看执行计划及执行概要，请参考 EXPLAIN 和 PROFILE。

最后更新: June 20, 2023

2022 vesoft Inc.

11.6 超级顶点（稠密点）处理

11.6.1 原理介绍

在图论中，超级顶点（稠密点）是指一个点有着极多的相邻边。相邻边可以是出边（我指向谁）或者是入边（谁指向我）。

由于幂律分布的特点，超级顶点现象非常普遍。 例如社交网络中的影响力领袖（网红大 V）、证券市场中的热门股票、银行系统中的四大行、交通网络中的枢纽站、互联网中的高流量网站等、电商网络中的爆款产品。

在 Nebula Graph 3.1.0 中，一个 `点` 和其 `属性` 是一个 `Key-Value`（以该点的 `VID` 以及其他元信息作为 `Key`），其 `Out-Edge Key-Value` 和 `In-Edge Key-Value` 都存储在同一个 partition 中（具体原理详见存储架构，并且以 LSM-tree 的形式组织存放在硬盘（和缓存）中。

因此不论是 `从该点出发的有向遍历`，或者 `以该点为终点的有向遍历`，都会涉及到大量的 `顺序 IO 扫描`（最理想情况，当完成 Compact 操作之后），或者大量的 `随机 IO`（有关于 `该点` 和其 `出入边` 频繁的写入）。

经验上说，当一个点的出入度超过 1 万时，就可以视为是稠密点。需要考虑一些特殊的设计和处理。

> **Note**
>
> Nebula Graph 中没有专用的字段来记录每个点的出度和入度，也没有内置任务来进行统计，因此无法预知哪些点会是超级节点。一个折中的办法是使用 Spark 周期性地计算和统计。

重复属性索引

在属性图中，除了网络拓扑结构中的超级顶点，还有一类情况类似于超级顶点————某属性有极高重复率，也即"相同的 `点类型 Tag`，不同的 `顶点 VID`，同一属性字段，拥有相同属性值"。

Nebula Graph 3.1.0 属性索引的设计复用了存储模块 RocksDB 的功能，这种情况下的索引会被建模为 `前缀相同的 Key`。对于该属性的查找，（如果未能命中缓存，）会对应为硬盘上的“一次随机查找 + 一次前缀顺序扫描”，以找到对应的 `点 VID`（此后，通常会从该顶点开始图遍历，这样又会发生该点对应 Key-Value 的一次随机读+顺序扫描）。当重复率越高，扫描范围就越大。

关于属性索引的原理详细介绍在博客《分布式图数据库 Nebula Graph 的 Index 实践》。

经验上说，当重复属性值超过 1 万时，也需要特殊的设计和处理。

建议的办法

数据库端的常见办法

1. 截断: 只访问一定阈值的边，超过该阈值的其他边则不返回。
2. Compact：重新组织 RocksDB 中数据的排列方式，减少随机读，增加顺序读。

2022 vesoft Inc.

应用端的常见办法

根据业务意义，将一些超级顶点拆分：

- 删除多条边，合并为一条

  例如，一个转账场景： `（账户 A)-[转账]-> (账户 B)`。每次 `转账` 建模为 `一条 AB 之间的边`，那么 `（账户 A)` 和 `（账户 B)` 之间会有着数万十次转账的场景。

  按日、周、或者月为粒度，合并陈旧的转账明细。也就是批量删除陈旧的边，改为少量的边“月总额”和“次数。而保留最近月的转账明细。

- 拆分相同类型的边，变为多种不同类型的边

  例如，`（机场）<-[depart]-（航班）` 场景，每个架次航班的离港，都建模为一条航班和机场之间的边。那么大型机场的离港航班会极多。

  根据不同的 `航空公司` 将 `depart` 这个 Edge type 拆分更细的 Edge type，如 `depart_ceair`，`depart_csair` 等。在查询（图遍历）时，指定离港的航空公司。

- 切分顶点本身

  例如，对于 `（人）-[借款]->（银行）` 的借款网络，某大型银行 A 的借款次数和借款人会非常的多。

  可以将该大行节点 A 拆分为多个相关联的子节点 A1、A2、A3，

  ```
  (人 1)-[借款]-> (银行 A1), (人 2)-[借款]-> (银行 A2), (人 2)-[借款]-> (银行 A3);
  (银行 A1)-[属于]-> (银行 A), (银行 A2)-[属于]-> (银行 A), (银行 A3)-[属于]-> (银行 A).
  ```

  这里的 A1、A2、A3 既可以是 A 真实的三个分行（例如北京、上海、浙江），也可以是三个按某种规则设立的虚拟分行，例如按借款金额划分 `A1: 1-1000, A2: 1001-10000, A3: 10000+`。这样，查询时对于 A 的任何操作，都转变为为对于 A1、A2、A3 的三次单独操作。

最后更新: June 20, 2023

2022 vesoft Inc.

11.7 实践案例

Nebula Graph 在各行各业都有应用，本文介绍部分实践案例。更多实践分享内容请参见博客。

11.7.1 业务场景

- 案例
- 经验
- 三方评测

11.7.2 内核

- MATCH 中变长 Pattern 的实现
- 如何向 Nebula Graph 增加一个测试用例
- 基于 BDD 理论的 Nebula 集成测试框架重构（上）
- 基于 BDD 理论的 Nebula 集成测试框架重构（下）
- 解析 Nebula Graph 子图设计及实践
- 基于全文搜索引擎的文本搜索
- 实操 | LDBC 数据导入及 nGQL 实践

11.7.3 周边工具

- 基于 Nebula Importer 批量导入工具性能验证方案总结
- 详解 Nebula 3.0 性能报告
- Nebula Graph 支持 JDBC 协议
- Nebula·利器 | Norm 知乎开源的 ORM 工具
- 基于 Nebula Graph 的 Betweenness Centrality 算法
- 无依赖单机尝鲜 Nebula Exchange 的 SST 导入
- logrotate 在 Nebula Graph 的日志滚动实践

最后更新: June 20, 2023

2022 vesoft Inc.

12. 客户端

12.1 客户端介绍

Nebula Graph 提供多种类型客户端，便于用户连接、管理 Nebula Graph 图数据库。

- Nebula Console：原生 CLI 客户端
- Nebula CPP：C++ 客户端
- Nebula Java：Java 客户端
- Nebula Python：Python 客户端
- Nebula Go：Go 客户端

Note

目前仅 Nebula Java 客户端支持线程安全（thread-safe）。

Caution

其他客户端工具（Nebula PHP、Nebula Node、Nebula .net、Nebula JDBC、NORM - Nebula Graph 的 Golang ORM、Graph-Ocean - Nebula Graph 的 Java ORM）也可用于连接和管理 Nebula Graph。他们由非常酷的社区用户提供和维护，欢迎大家参与测试和贡献更多社区项目。

最后更新: June 20, 2023

2022 vesoft Inc.

12.2 Nebula Console

Nebula Console 是 Nebula Graph 的原生命令行客户端，用于连接 Nebula Graph 集群并执行查询，同时支持管理参数、导出命令的执行结果、导入测试数据集等功能。

12.2.1 获取 Nebula Console

Nebula Console 的获取方式如下：

- 从 GitHub 发布页下载二进制文件。
- 编译源码获取二进制文件。编译方法参见 Install from source code。

12.2.2 功能说明

连接 **Nebula Graph**

运行二进制文件 `nebula-console` 连接 Nebula Graph 的命令语法如下：

```
<path_of_console> -addr <ip> -port <port> -u <username> -p <password>
```

`path_of_console` 是 Nebula Console 二进制文件的存储路径。

常用参数的说明如下。

参数	说明
`-h/-help`	显示帮助菜单。
`-addr/-address`	设置要连接的 Graph 服务的 IP 地址。默认地址为 127.0.0.1。
`-P/-port`	设置要连接的 Graph 服务的端口。默认端口为 9669。
`-u/-user`	设置 Nebula Graph 账号的用户名。未启用身份认证时，可以使用任意已存在的用户名（默认为 `root`）。
`-p/-password`	设置用户名对应的密码。未启用身份认证时，密码可以填写任意字符。
`-t/-timeout`	设置整数类型的连接超时时间。单位为秒，默认值为 120。
`-e/-eval`	设置字符串类型的 nGQL 语句。连接成功后会执行一次该语句并返回结果，然后自动断开连接。
`-f/-file`	设置存储 nGQL 语句的文件的路径。连接成功后会执行该文件内的 nGQL 语句并返回结果，执行完毕后自动断开连接。
`-enable_ssl`	连接 Nebula Graph 时使用 SSL 加密。
`-ssl_root_ca_path`	指定 CA 证书的存储路径。
`-ssl_cert_path`	指定 CRT 证书的存储路径。
`-ssl_private_key_path`	指定私钥文件的存储路径。

更多参数参见项目仓库。

例如，要连接到部署在 192.168.10.8 上的 Graph 服务，运行以下命令。

```
./nebula-console -addr 192.168.10.8 -port 9669 -u Joe -p Joespassword
```

管理参数

Nebula Console 可以保存参数，用于参数化查询。

2022 vesoft Inc.

Note

- VID 不支持参数化查询。
- `SAMPLE` 子句中不支持参数化查询。
- 会话释放后，参数不会保留。

- 保存参数命令如下：

```
nebula> :param <param_name> => <param_value>;
```

示例：

```
nebula> :param p1 => "Tim Duncan";
nebula> MATCH (v:player{name:$p1})-[:follow]->(n)  RETURN v,n;
+-------------------------------------------------+-------------------------------------------------+
| v                                               | n                                               |
+-------------------------------------------------+-------------------------------------------------+
| ("player100" :player{age: 42, name: "Tim Duncan"}) | ("player125" :player{age: 41, name: "Manu Ginobili"}) |
| ("player100" :player{age: 42, name: "Tim Duncan"}) | ("player101" :player{age: 36, name: "Tony Parker"})   |
+-------------------------------------------------+-------------------------------------------------+

nebula> :param p2 => {"a":3,"b":false,"c":"Tim Duncan"};
nebula> RETURN $p2.b AS b;
+-------+
| b     |
+-------+
| false |
+-------+
```

- 查看当前保存的所有参数，命令如下：

```
nebula> :params;
```

- 查看指定参数，命令如下：

```
nebula> :params <param_name>;
```

- 删除指定参数，命令如下：

```
nebula> :param <param_name> =>;
```

导出执行结果

导出命令执行的返回结果，可以保存为 CSV 文件或 DOT 文件。

2022 vesoft Inc.

Note

- 文件保存在当前工作目录中，即 Linux 命令`pwd`显示的目录。
- 命令只对下一条查询语句生效。
- DOT 文件的内容可以复制后在 GraphvizOnline 网页中粘贴，生成可视化的执行计划图。

- 导出 CSV 文件命令如下：

```
nebula> :CSV <file_name.csv>;
```

- 导出 DOT 文件命令如下：

```
nebula> :dot <file_name.dot>;
```

示例：

```
nebula> :dot a.dot;
nebula> PROFILE FORMAT="dot" GO FROM "player100" OVER follow;
```

加载测试数据集

测试数据集名称为 basketballplayer，详细 Schema 信息和数据信息请使用相关`SHOW`命令查看。

加载测试数据集命令如下：

```
nebula> :play basketballplayer;
```

重复执行语句

重复执行下一个命令 N 次，然后打印平均执行时间。命令如下：

```
nebula> :repeat N;
```

示例：

```
nebula> :repeat 3;
nebula> GO FROM "player100" OVER follow YIELD dst(edge);
+-------------+
| dst(EDGE)   |
+-------------+
| "player101" |
| "player125" |
+-------------+
Got 2 rows (time spent 2602/3214 us)

Fri, 20 Aug 2021 06:36:05 UTC

+-------------+
| dst(EDGE)   |
+-------------+
| "player101" |
| "player125" |
+-------------+
Got 2 rows (time spent 583/849 us)

Fri, 20 Aug 2021 06:36:05 UTC

+-------------+
| dst(EDGE)   |
+-------------+
| "player101" |
| "player125" |
+-------------+
Got 2 rows (time spent 496/671 us)

Fri, 20 Aug 2021 06:36:05 UTC

Executed 3 times, (total time spent 3681/4734 us), (average time spent 1227/1578 us)
```


2022 vesoft Inc.

睡眠

睡眠 N 秒。常用于修改 Schema 的操作中，因为修改 Schema 是异步实现的，需要在下一个心跳周期才同步数据。命令如下：

```
nebula> :sleep N;
```

断开连接

用户可以使用 `:EXIT` 或者 `:QUIT` 从 Nebula Graph 断开连接。为方便使用，Nebula Console 支持使用不带冒号（:）的小写命令，例如 `quit`。

示例：

```
nebula> :QUIT;

Bye root!
```

最后更新: June 20, 2023

2022 vesoft Inc.

12.3 Nebula CPP

Nebula CPP 是一款 C++ 语言的客户端，可以连接、管理 Nebula Graph 图数据库。

12.3.1 使用限制

请确保已安装 C++ 且 GCC 版本为 4.8 及以上。

12.3.2 版本对照表

Nebula Graph 版本	**Nebula CPP** 版本
3.1.0	3.0.2
2.6.x	2.5.0
2.5.x	2.5.0
2.0.x	2.0.0

12.3.3 安装 Nebula CPP

本文介绍通过编译方式安装 Nebula CPP。

前提条件

- 准备正确的编译环境，详情请参见软硬件要求和安装三方库依赖包。
- 确保已安装 C++ 且 GCC 版本为：{10.1.0 | 9.3.0 | 9.2.0 | 9.1.0 | 8.3.0 | 7.5.0 | 7.1.0}。详情请参见 gcc_preset_versions 参数。

安装步骤

1. 克隆 Nebula CPP 源码到机器。

- （推荐）如果需要安装指定版本的 Nebula CPP，请使用选项 `--branch` 指定分支。例如安装 v3.0.2发布版本，请执行如下命令：

```
$ git clone --branch v3.0.2 https://github.com/vesoft-inc/nebula-cpp.git
```

- 如果需要安装日常开发版本，请执行如下命令下载 `master` 分支的源码：

```
$ git clone https://github.com/vesoft-inc/nebula-cpp.git
```

2. 进入目录 nebula-cpp。

```
$ cd nebula-cpp
```

3. 创建目录 build 并进入该目录。

```
$ mkdir build && cd build
```

4. 使用 CMake 生成 makefile 文件。

> **Note**
>
> 默认安装路径为 `/usr/local/nebula`，如果需要修改路径，请在下方命令内增加参数 `-DCMAKE_INSTALL_PREFIX=<installation_path>`。

```
$ cmake -DCMAKE_BUILD_TYPE=Release ..
```


2022 vesoft Inc.

> **Note**
>
> 如果 g++ 不支持 C++11，请添加选项 `-DDISABLE_CXX11_ABI=ON` 。

5. 编译 Nebula CPP。

   为了适当地加快编译速度，可以使用选项 `-j` 并行编译。并行数量 `N` 建议为\(\min(\text{CPU}核数，\frac{内存 (GB)}{2})\)。

   ```
   $ make -j{N}
   ```

6. 安装 Nebula CPP。

   ```
   $ sudo make install
   ```

7. 更新动态链接库。

   ```
   $ sudo ldconfig
   ```

12.3.4 使用方法

将 CPP 文件编译为可执行文件即可。接下来以 `SessionExample.cpp` 为例，介绍如何操作。

1. 使用示例代码创建 `SessionExample.cpp` 文件。
2. 编译文件，命令如下：

   ```
   $ LIBRARY_PATH=<library_folder_path>:$LIBRARY_PATH g++ -std=c++11 SessionExample.cpp -I<include_folder_path> -lnebula_graph_client -o session_example
   ```

- `library_folder_path`： Nebula Graph 动态库文件存储路径，默认为 `/usr/local/nebula/lib64` 。
- `include_folder_path`： Nebula Graph 头文件存储路径，默认为 `/usr/local/nebula/include` 。

  示例：

  ```
  $ LIBRARY_PATH=/usr/local/nebula/lib64:$LIBRARY_PATH g++ -std=c++11 SessionExample.cpp -I/usr/local/nebula/include -lnebula_graph_client -o session_example
  ```

12.3.5 核心代码

详细示例请参见 SessionExample。

最后更新: June 20, 2023

2022 vesoft Inc.

12.4 Nebula Java

Nebula Java 是一款 Java 语言的客户端，可以连接、管理 Nebula Graph 图数据库。

12.4.1 前提条件

已安装 Java，版本为 8.0 及以上。

12.4.2 版本对照表

Nebula Graph 版本	**Nebula Java** 版本
3.1.0	3.0.0
2.6.x	2.6.1
2.0.x	2.0.0
2.0.0-rc1	2.0.0-rc1

12.4.3 下载 Nebula Java

- （推荐）如果需要使用指定版本的 Nebula Java，请使用选项 `--branch` 指定分支。例如使用 v3.0.0发布版本，请执行如下命令：

```
$ git clone --branch v3.0.0 https://github.com/vesoft-inc/nebula-java.git
```

- 如果需要安装日常开发版本，请执行如下命令下载 `master` 分支的源码：

```
$ git clone https://github.com/vesoft-inc/nebula-java.git
```

12.4.4 使用方法

> **Note**
>
> 建议一个线程使用一个会话，如果多个线程使用同一个会话，会降低效率。

使用 IDEA 等工具导入 Maven 项目，请在 `pom.xml` 中添加如下依赖：

> **Note**
>
> `3.0.0-SNAPSHOT` 为日常研发版本，可能存在未知问题，建议使用 release 版本号替换 `3.0.0-SNAPSHOT` 。

```
<dependency>
  <groupId>com.vesoft</groupId>
  <artifactId>client</artifactId>
  <version>3.0.0-SNAPSHOT</version>
</dependency>
```

如果无法下载日常研发版本的依赖，请在 `pom.xml` 中添加如下内容（release 版本不需要添加）：

```
<repositories>
  <repository>
    <id>snapshots</id>
    <url>https://oss.sonatype.org/content/repositories/snapshots/</url>
  </repository>
</repositories>
```


2022 vesoft Inc.

如果没有 Maven 管理项目，请手动下载 JAR 包进行安装。

核心代码

详细示例请参见 GraphClientExample。

12.4.5 Java 相关库

以下由非常酷的社区用户提供和维护，欢迎大家参与测试和贡献。

- java-jdbc
- java-orm
- java-springboot demo

最后更新: June 20, 2023

2022 vesoft Inc.

12.5 Nebula Python

Nebula Python 是一款 Python 语言的客户端，可以连接、管理 Nebula Graph 图数据库。

12.5.1 前提条件

已安装 Python，版本为 3.6 及以上。

12.5.2 版本对照表

Nebula Graph 版本	**Nebula Python** 版本
3.1.0	3.1.0
2.6.x	2.6.0
2.0.x	2.0.0
2.0.0-rc1	2.0.0rc1

12.5.3 安装 Nebula Python

pip 安装

```
$ pip install nebula3-python==<version>
```

克隆源码安装

1. 克隆 Nebula Python 源码到机器。

- （推荐）如果需要安装指定版本的 Nebula Python，请使用选项 `--branch` 指定分支。例如安装 v3.1.0发布版本，请执行如下命令：

```
$ git clone --branch release-3.1 https://github.com/vesoft-inc/nebula-python.git
```

- 如果需要安装日常开发版本，请执行如下命令下载 `master` 分支的源码：

```
$ git clone https://github.com/vesoft-inc/nebula-python.git
```

2. 进入目录 nebula-python。

```
$ cd nebula-python
```

3. 执行如下命令安装。

```
$ pip install .
```

12.5.4 核心代码

详细示例请参见 Example。

最后更新: June 20, 2023

2022 vesoft Inc.

12.6 Nebula Go

Nebula Go 是一款 Go 语言的客户端，可以连接、管理 Nebula Graph 图数据库。

12.6.1 前提条件

已安装 Go，版本为 1.13 及以上。

12.6.2 版本对照表

Nebula Graph 版本	**Nebula Go** 版本
3.1.0	3.1.0
2.6.x	2.6.0
2.0.x	2.0.0-GA

12.6.3 下载 Nebula Go

- （推荐）如果需要下载指定版本的 Nebula Go，请使用选项 `--branch` 指定分支。例如安装 v3.1.0发布版本，请执行如下命令：

```
$ git clone --branch release-3.1 https://github.com/vesoft-inc/nebula-go.git
```

- 如果需要安装日常开发版本，请执行如下命令下载 `master` 分支的源码：

```
$ git clone https://github.com/vesoft-inc/nebula-go.git
```

12.6.4 安装或更新

安装或更新的命令如下：

```
$ go get -u -v github.com/vesoft-inc/nebula-go/v3@v3.1.0
```

12.6.5 核心代码

详细示例请参见 graph_client_basic_example 和 graph_client_goroutines_example。

最后更新: June 20, 2023

2022 vesoft Inc.

13. Nebula Graph Cloud

13.1 什么是 NebulaGraph Cloud

NebulaGraph Cloud（简称 Cloud）是一套集成了 NebulaGraph 数据库和数据服务的云上服务，支持一键部署 NebulaGraph 和相关可视化产品。用户可以在几分钟内创建一个图数据库，并快速扩展计算、存储等资源。

Caution

NebulaGraph Cloud使用的 NebulaGraph 内核已升级到 3.1.3 版本，请阅读最新版文档。

- 基于阿里云的自管云服务，即 NebulaGraph Cloud 阿里云版。
- 基于 AWS 的自管云服务，即 NebulaGraph Cloud AWS 版。
- 基于 Azure 的自管云服务，即 NebulaGraph Cloud Azure 版。

13.1.1 内核版本

Cloud 支持的最新 NebulaGraph 内核版本为 3.1.3。 -->

13.1.2 产品功能

- 即买即用。用户可以根据自己的业务发展随时创建或停止 NebulaGraph 实例，业务提供充分的灵活性。
- 集成可视化图数据库管理工具 Nebula Dashboard、Nebula Explorer。可以使用它们导入图数据集、执行 nGQL 语句查询、探索图数据、监控数据等。即使没有图数据库操作经验，也可以快速成为图专家。
- 用户可直接使用 Nebula Console 等客户端连接到云上数据库，而不需要先访问云平台。
- 权限控制。用户可以为指定图空间增加不同角色权限的数据库用户，保证业务数据安全。
- 官方支持。用户可以快速创建工单，咨询使用过程中遇到的问题。

13.1.3 产品优点

- 灵活与弹性：Cloud 最大的优势在于灵活与弹性，无需自购软硬件即可快速（5-10分钟）搭建 NebulaGraph 数据库。
- 高可用性：Cloud 上创建的 NebulaGraph 集群采用 Raft 协议，结合云厂商提供的同城多机房实现数据冗余备份，保证云计算基础框架的高可用性、数据的可靠性以及云服务器的高可用性。
- 操作易用：Cloud 提供 Web 页面对 NebulaGraph 集群进行在线管理，同时提供配套的可视化软件，降低用户使用门槛，快速上手。

13.1.4 视频

用户也可以观看视频了解 NebulaGraph Cloud。

NebulaGraph Cloud 阿里云版介绍（05 分 37 秒）

最后更新: June 20, 2023

2022 vesoft Inc.

13.2 Nebula Graph Cloud 阿里云版

13.2.1 创建服务实例

Caution

NebulaGraph Cloud 阿里云版使用的 NebulaGraph 内核已升级到 3.1.3 版本，请阅读最新版文档。

在使用 NebulaGraph Cloud 阿里云版服务前，要先在阿里云控制台创建服务实例。

Caution

通过 NebulaGraph Cloud 阿里云版创建的 NebulaGraph、Nebula Dashboard 和 Nebula Explorer 等服务都是企业版，试用期 14 天。如需长期使用，发送邮件申请正式许可证，之后续期许可证。

账号与权限

- 准备阿里云账号。
- 如果使用 RAM 用户创建实例，需为其添加以下权限：
- AliyunECSFullAccess
- AliyunVPCFullAccess
- AliyunROSFullAccess
- AliyunCloudMonitorFullAccess
- AliyunComputeNestUserFullAccess

资源与费用

NebulaGraph Cloud 阿里云版支持免费试用和付费使用。两种使用方式有不同的资源归属和费用消耗。

免费试用版服务实例使用的资源属于阿里云，资源费用由阿里云承担，试用期间用户无需承担云资源的费用和 **NebulaGraph** 产品的许可证费用。实例内 NebulaGraph 相关服务的许可证到期后，其中的数据仍然保留在试用的阿里云资源中，直到资源过期或被用户删除。资源的试用时间为 30 天，到期后资源内保存的数据不保留。

付费版服务实例创建在用户的阿里云账号下，云资源费用由用户承担。实例内 NebulaGraph 相关服务的许可证到期后，其中的数据仍然保留在用户的阿里云资源中，直到资源被用户删除。

Note

基础版套餐将所有 NebulaGraph 服务节点部署在同一台 ECS 服务器上。

创建付费版服务实例

1. 登录阿里云计算巢推荐服务页面，搜索 **NebulaGraph**。
2. 在 **NebulaGraph** 集群版卡片上，单击云市场购买。
3. 在云市场的 **NebulaGraph Enterprise** 产品页，选择套餐版本和购买时长（即 NebulaGraph 服务的许可证有效期），并单击立即购买。

2022 vesoft Inc.

Caution

当前云市场显示的部分价格并非实际售价，购买时需通过右侧的钉钉客服确认实际价格。

4. 在确认订单页面，选择要创建实例的地域。
5. 在付费模式设置区域，指定 ECS 服务器的付费方式。默认为按量付费。

- 按量付费：按照计费周期计费，在每个结算周期生成账单并从账户中扣除相应费用。详情参见按量付费。
- 包年包月：先付费后使用。详情参见包年包月。选择包年包月模式需要指定购买时长周期和购买时长。购买时长周期当前仅支持 **Month**，即按月购买。

6. 在套餐选择区域，根据需要选择套餐版本。
7. 在 **NebulaGraph** 配置区域，完成数据盘和 ECS 服务器密码设置。
8. 在基础设施配置区域，完成以下设置。

a. 选择专有网络 **VPC** 实例 **ID**。

如果下拉列表为空，先单击其右侧的新建专有网络，完成专有网络创建。

b. 选择交换机可用区。

c. 选择业务网络交换机的实例 **ID**。

如果下拉列表为空，先单击其右侧的新建交换机，完成交换机创建。

9. 在温馨提示区域，选中我同意告知服务商我的联系方式，以便购买后发放许可证。
10. 在页面底部，单击我已接受协议，前往支付。
11. 在支付页面，选择支付方式，之后单击页面底部的支付。
12. 在支付完成页面，单击管理控制台。
13. 在已购买的服务页面，单击要创建的实例右侧的立即进入计算巢部署或查看部署详情。
14. 在计算巢的服务实例详情页面，单击基本信息区域状态右侧的部署，并在弹出的对话框中单击确定部署。
15. 监控实例状态，直到部署完成。平均部署耗时为 10 分钟。

2022 vesoft Inc.

创建免费版试用版服务实例

Danger

免费试用实例不可在生产环境中使用。

1. 登录阿里云计算巢推荐服务页面，搜索 **NebulaGraph**。
2. 在 **NebulaGraph** 集群版卡片上，单击免费试用。
3. 系统会为首次创建 NebulaGraph 服务实例的用户弹出申请对话框。在申请权限对话框中填写申请人信息。带有红色星号（*）的为必填项。

Note

申请通过后需从第 1 步重新开始。

4. 设置服务实例名称。
5. 选择要创建实例的地域。
6. 在套餐选择区域，根据需要选择套餐版本。
7. 在 **NebulaGraph** 配置区域，完成数据盘相关设置。
8. 在基础设施配置区域，选择交换机可用区。
9. 根据需要配置标签和资源组。详情参见什么是资源管理。
10. 完成权限确认，并选中我同意授权服务商（杭州悦数科技有限公司）获取上述权限以提供代运维服务。
11. 在页面底部，选中我已阅读并同意《计算巢服务协议》，并单击开始免费试用。
12. 在创建对话框，完成信息确认并单击确定。
13. 在提交成功页面，单击去列表查看。
14. 在实例列表中查看目标实例的状态，确保状态为部署中。

    部署的平均耗时为 10 分钟。完成后实例的状态变为已部署。

常见问题

Q：服务实例的状态显示为部署失败怎么处理?

1. 如果使用 RAM 账号创建的实例，确认为该账号授予了本文前提条件中指定的权限。
2. 如果权限符合要求，删除创建失败的实例，尝试重新创建。
3. 如果仍然创建失败，到 NebulaGraph 论坛寻求帮助。

最后更新: June 20, 2023

2022 vesoft Inc.

13.2.2 使用服务实例

创建好服务实例后，即可查看服务信息，并开始使用 NebulaGraph 服务。

试用期与续期

NebulaGraph Cloud 阿里云版中的 NebulaGraph、Nebula Dashboard 和 Nebula Explorer 服务都是企业版，需要有效的许可证才能正常运行。创建实例时默认赠送有效期为 3 天的试用版许可证。

试用版许可证到期时，服务无法使用。如需继续使用，发送邮件申请正式许可证。

获取有效的许可证后，使用新的许可证替换旧许可证，即可继续使用服务。

> **Note**
>
> 替换许可证时需要连接 Meta 服务所在的阿里云 ECS 实例，方法参见连接方式概述。

正式版许可证的续期方法与试用版相同。

查看服务实例

在计算巢控制台可以查看服务实例的详细信息，详情参见查看服务实例。

查看连接地址

连接服务前，需要在阿里云计算巢的实例详情中查看各服务的连接地址。查看方式如下：

1. 登录服务实例管理页面。
2. 在实例列表中，单击目标实例的服务实例**ID**，或其右侧操作列的详情。
3. 在概览页的基本信息区域，可以查看 Graph 服务、Meta 服务、Storage 服务，以及 Explorer 和 Dashboard 的连接地址。

常用的连接地址有：

- **nebula_private_ip** 与 **nebula_public_ip**，即 Graph 服务的私网与公网 IP 地址，可用于连接 NebulaGraph。
- **explorer_portal**，即 Nebula Explorer 的连接地址，单击地址即可在浏览器中打开 Nebula Explorer。
- **dashboard_portal**，即 Nebula Dashboard 的连接地址，单击地址即可在浏览器中打开 Nebula Dashboard。

连接 NebulaGraph

NebulaGraph Cloud 阿里云版提供多种连接方式。

使用客户端直连 NEBULAGRAPH

获取到连接地址后，即可使用 NebulaGraph 的各类客户端连接服务。

连接方式：

直连 **graph_private_ip** 或 **graph_public_ip** 地址。详细连接方式参见客户端文档。

使用 EXPLORER 连接 NEBULAGRAPH

使用图探索工具可以快速连接 NebulaGraph，与图数据进行可视化交互。

2022 vesoft Inc.

连接方式:

1. 在浏览器中使用 **explorer_portal** 地址打开 Nebula Explorer。
2. 使用 Explorer 连接 NebulaGraph。详情参见连接数据库。

监控 NebulaGraph

在 NebulaGraph Cloud 阿里云版中部署 Dashboard 后，可以使用 Dashboard 实时监控 NebulaGraph 服务的状态。详情参见 Dashboard 文档。

最后更新: June 20, 2023

2022 vesoft Inc.

13.2.3 扩缩容服务

NebulaGraph Cloud 阿里云版的弹性扩缩容功能支持根据需求扩缩容服务节点，灵活调整集群规模。

使用限制

- 弹性扩缩容功能支持 Storage 扩容，不支持 Storage 缩容。
- 使用弹性扩缩容功能缩容 Graph 服务时，需先使用集群中 NebulaGraph Dashboard 服务的扩缩容功能删除不再需要的 Graph 服务节点。扩容 Graph 服务时无此限制。

操作步骤

1. 登录阿里云计算巢的服务实例管理页面。
2. 在服务实例列表中，单击目标实例的ID或其右侧操作列的详情。

> **Note**
>
> 免费试用版服务实例列表在试用服务页签中。

3. 在服务实例详情页面，单击运维管理页签。
4. 在应用运维区域，单击弹性扩缩容。
5. 在弹性扩缩容对话框，完成如下操作。

a. 确认将要调整的资源类型为 **ESS**。

b. 选择伸缩组 **ID**。

- 如需扩缩容 Storage 服务，选择名称以 **ng-storage** 开头的伸缩组。
- 如需扩缩容 Graph 服务，选择名称以 **ng-graph** 开头的伸缩组。

c. 选择伸缩规则的调整方式。

- 如果选择扩容或缩容，下方的伸缩组 **ECS** 实例调整的数量指定的是要增加或减少的节点数量。
- 如果选择调整至指定数量，下方的伸缩组 **ECS** 实例调整的数量指定的是节点调整之后剩余的数量，而不是增加或减少的数量。

d. 输入伸缩组 **ECS** 实例调整的数量。

e. 单击下一步。

f. 检查任务信息，确认无误后单击创建。

6. 等候数分钟，直到运维管理页签中的执行状态显示成功。
7. 单击概览页签，查看服务资源区域的 ECS 实例数量，如果符合预期的调整效果，则扩缩容完成。

最后更新: June 20, 2023

2022 vesoft Inc.

13.2.4 删除服务实例

如果不再需要某个实例，可手动删除该实例。

删除方式参见删除服务实例。

最后更新: June 20, 2023

2022 vesoft Inc.

13.3 Nebula Graph Cloud AWS 版

13.3.1 NebulaGraph Cloud AWS 版概览

NebulaGraph 提供了基于 AWS EC2 的安全、可靠的云服务。用户可以使用 CloudFormation 模板快速部署一整套企业版 NebulaGraph 云服务，包含相应的网络、计算、存储、可视化等服务。

费用

用户需承担部署和运行 NebulaGraph 服务使用的 AWS 资源相应的费用。

在创建 NebulaGraph 服务集群时，部分配置会影响费用，例如实例规格、存储介质类型、存储空间等。页面下方会显示当前配置的预估费用。

部署 NebulaGraph 服务集群后，用户可以创建成本和使用率报告，AWS会根据报告的设置将计费指标数据发送到指定的AWS S3 存储桶中。这些报告会根据每个月的使用情况提供成本估算，并在月末汇总数据。

除了 AWS 资源，用户还需要为 NebulaGraph 服务申请许可证，并保持许可证有效。申请许可证需要一定的费用。

许可证

NebulaGraph Cloud AWS 版使用 NebulaGraph 企业版服务，默认提供有效期 3 天的试用版许可证。许可证到期后服务不可用，数据依然保留在 AWS 资源中。

申请和设置许可证的说明参见设置内核许可证和设置 Explorer 许可证。

内核版本

当前支持的 NebulaGraph 企业版内核版本为 3.1.2。

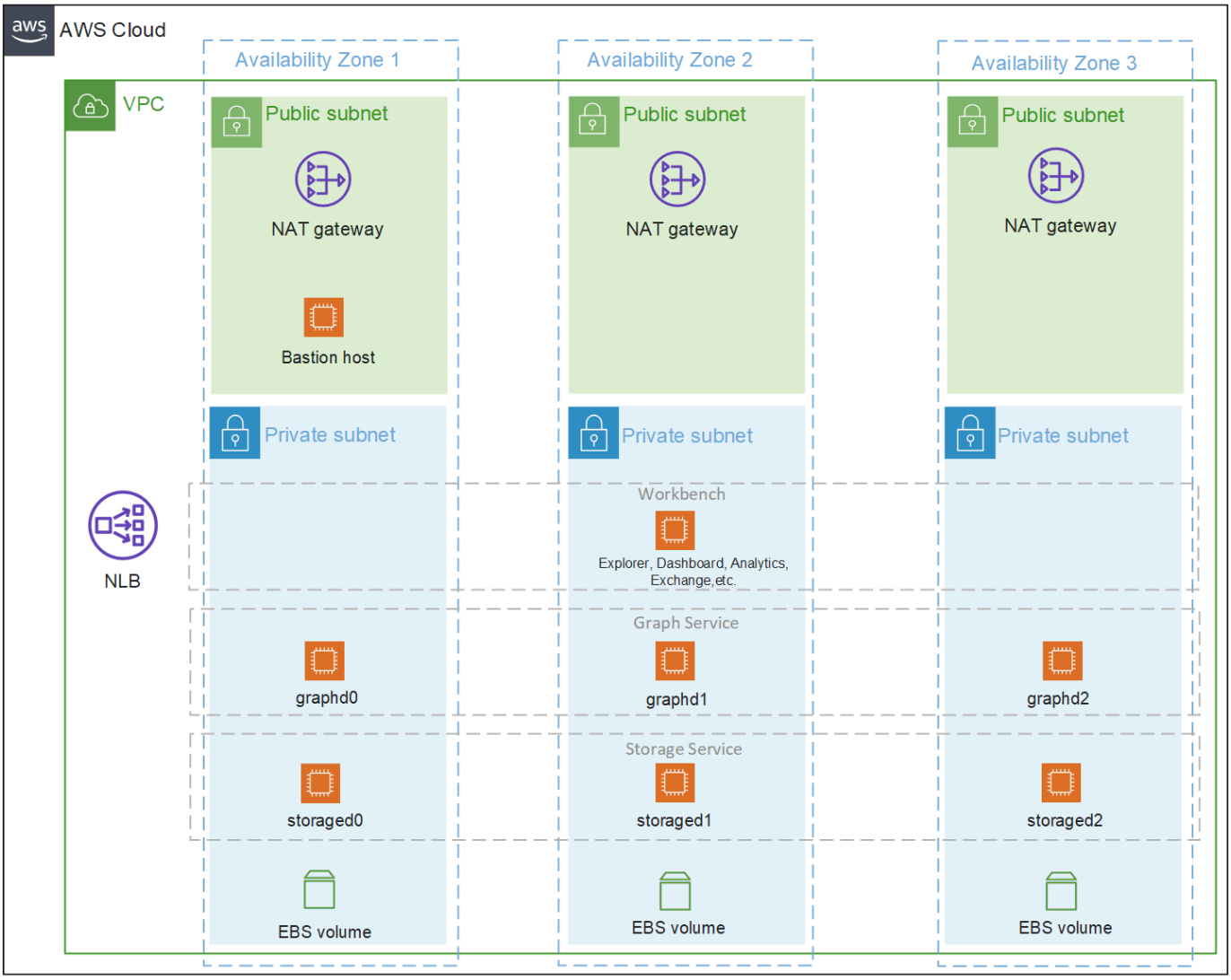
最后更新: June 20, 2023

2022 vesoft Inc.

13.3.2 部署架构

使用 CloudFormation 模板创建 NebulaGraph 服务集群时可以按需调节集群组件。本文介绍生产环境中推荐的部署架构。

生产环境中典型的集群架构如下图所示。



上图中的集群包含：

- 一套跨越三个可用区域的高可用架构。
- 一个配置了公共子网和私有子网的 VPC（virtual private cloud）网络。
- 一个用于在多个主机之间均衡流量的网络负载均衡器（Network Load Balancer）。

公共子网中包含：

- 允许堡垒机等服务连接互联网的 NAT 网关。
- 一台 Linux 系统的堡垒机，用于支持通过 SSH 方式访问公共子网和私有子网。

2022 vesoft Inc.

私有子网中包含：

- 一台 Linux 系统的 Workbench 机器，混合部署了 Explorer、Dashboard、Analytics、Exchange 等 NebulaGraph 生态工具。
- 运行在 graphd 进程中的 NebulaGraph Graph 服务。共有三个 graphd 进程，每个私有网络中一个进程。
- 运行在 storaged 进程中的 NebulaGraph Storage 服务。共有三个 storaged 进程，每个私有网络中一个进程。
- 每个子网一个 EBS（Elastic Block Store）卷。

最后更新: June 20, 2023

2022 vesoft Inc.

13.3.3 部署准备

开始部署 NebulaGraph Cloud AWS 版的服务集群前，用户需要具备相关的 AWS 和 NebulaGraph 知识、准备好 AWS 账号、确认资源情况和选择部署方式。

知识准备

用户需要熟悉 AWS 的常用服务，尤其是 EC2、VPC 和 EBS。AWS 新用户可以通过 Getting Started Resource Center、AWS Training and Certification 学习相应的知识。

账户与权限

用户需准备 AWS 账户。如果没有，在 AWS 主页申请。

部署集群使用的 AWS 账户必须具有使用使用相应资源的 IAM 权限。

资源额度

部署前需要在 Service Quotas console 检查 AWS 资源额度，防止资源额度不足导致部署失败。部署一个 NebulaGraph 服务集群需要的资源见下表。

资源	部署需求数
VPC	1 个
弹性 IP 地址	1 个（堡垒机）
安全组	2 个
IAM 角色	2 个
负载均衡器	1 个
EC2 实例（堡垒机）	1 个
EC2 实例（Workbench）	1 个
EC2 实例（graphd）	1 到 3 个
EC2 实例（storaged）	1 或 3 或 5 或 7 个

EC2 实例的数量由实际部署架构决定。

如果所需资源的额度不足，先申请提升资源额度再部署集群。

AWS 地域

如需在默认地域（us-east-1）以外部署集群，建议先在 AWS Regional Services 页面确认目标地域支持部署 NebulaGraph 服务集群需要的所有服务。

如果部署时返回错误信息 `Unrecognized resource type`，则该地域不支持部署。

EC2 密钥对

确保目标地域存在至少一对 EC2 密钥对。记住需要使用的密钥对名称，部署集群时要通过名称选择密钥对。

如果部署集群的目的是测试或验证概念，建议创建新的密钥对，而不是使用生产实例已经使用的密钥对。

2022 vesoft Inc.

部署方式

AWS CloudFormation 提供了两种部署方式:

- 在新的 VPC 中部署 NebulaGraph。该方式支持构建一个新的 AWS 环境，包括 VPC、子网、NAT 网关、安全组、堡垒机等基础设施组件，NebulaGraph 将被部署在新的 VPC 中。
- 在现有 VPC 中部署 NebulaGraph。如果决定使用该方式，需确保选定的 VPC 在每个目标可用区都有两个互不共享的私有子网。

用户需在开始部署前决定部署方式，并在部署步骤中指定相应的模板。

最后更新: June 20, 2023

2022 vesoft Inc.

13.3.4 部署步骤

完成部署准备后即可开始部署 NebulaGraph 服务集群。部署过程耗时约 15 到 20 分钟。

1. 使用拥有必要 IAM 权限的 AWS 账号登录 AWS Console。
2. 根据选定的部署方式，单击在新的 VPC 中部署 NebulaGraph 或在现有 VPC 中部署 NebulaGraph 以指定模板。
3. 检查页面右上角的地域，如有需要则修改地域。默认地域为 us-east-1。
4. 在 **Create stack** 页面，单击右下角的 **Next**。
5. 在 **Specify stack details** 页面，设置 **Stack name**，即服务集群的名称。
6. 检查页面中的模板参数，根据业务需要和参数下方的详细说明设置或修改相关参数，之后单击页面右下角的 **Next**。

> **Note**
>
> 如需从外网访问 NebulaGraph 服务集群，需将 **Permitted IP range** 设置为客户端的外网 IP 地址。建议从 VPC 内网访问，防止网络安全问题。

7. 在 **Configure stack options** 页面，根据需要设置 **Tags**、**Permissions**、**Stack failure options** 和 **Advanced options**，之后单击页面右下角的 **Next**。

   各选项的详细说明参见 Setting AWS CLoudFormation stack options。
8. 在 **Review** 页面，检查模板的设置。在 **Capacities** 区域，确认须知内容后选中两个复选框。
9. 单击页面右下角的 **Create stack**，开始部署服务集群。
10. 在 **Events** 页签中关注 stack 的 **Status**，当其变为 `CREATE_COMPLETE` 时，NebulaGraph 服务集群的创建成功。

最后更新: June 20, 2023

2022 vesoft Inc.

13.3.5 连接服务

完成 NebulaGraph 服务集群部署后，用户可以在 Stack 详情页的 **Outputs** 页签下查看各服务的连接信息。

连接信息包括:

- 堡垒机的弹性 IP 地址。
- Dashboard 服务的登录入口链接以及默认登录方式。
- Explorer 服务的登录入口链接。
- Graph 服务的私网 IP 地址与端口号。

使用 Nebula Dashboard 连接 NebulaGraph 的说明参见连接 Dashboard。

使用 Nebula Explorer 连接 NebulaGraph 的说明参见连接数据库。

使用客户端连接 NebulaGraph 的说明参见客户端文档。

最后更新: June 20, 2023

2022 vesoft Inc.

13.4 Nebula Graph Cloud Azure 版

13.4.1 NebulaGraph Cloud Azure 版概览

NebulaGraph 提供了基于 Azure 的安全、可靠的云服务。用户可以使用 Azure Marketplace 快速部署一整套企业版 NebulaGraph 企业版服务，包含相应的网络、计算、存储、可视化等服务。

费用

用户需承担部署和运行 NebulaGraph 企业版服务使用的 Azure 资源相应的费用。

在创建 NebulaGraph 企业版集群时，部分配置会影响费用，例如虚拟机规格、存储介质类型、存储空间等。

部署 NebulaGraph 服务集群后，用户可以使用 Microsoft Cost Management 分析成本并制定预算。

除了 Azure 资源，用户还需要为 NebulaGraph 企业版服务申请许可证，并保持许可证有效。申请许可证需要一定的费用。

许可证

NebulaGraph Cloud Azure 版使用 NebulaGraph 企业版服务，默认提供有效期 3 天的试用版许可证。许可证到期后服务不可用，数据依然保留在 Azure 资源中。

申请和设置许可证的说明参见设置内核许可证、设置 Dashboard 许可证和设置 Explorer 许可证。

内核版本

当前支持的 NebulaGraph 企业版内核版本为 3.1.2。

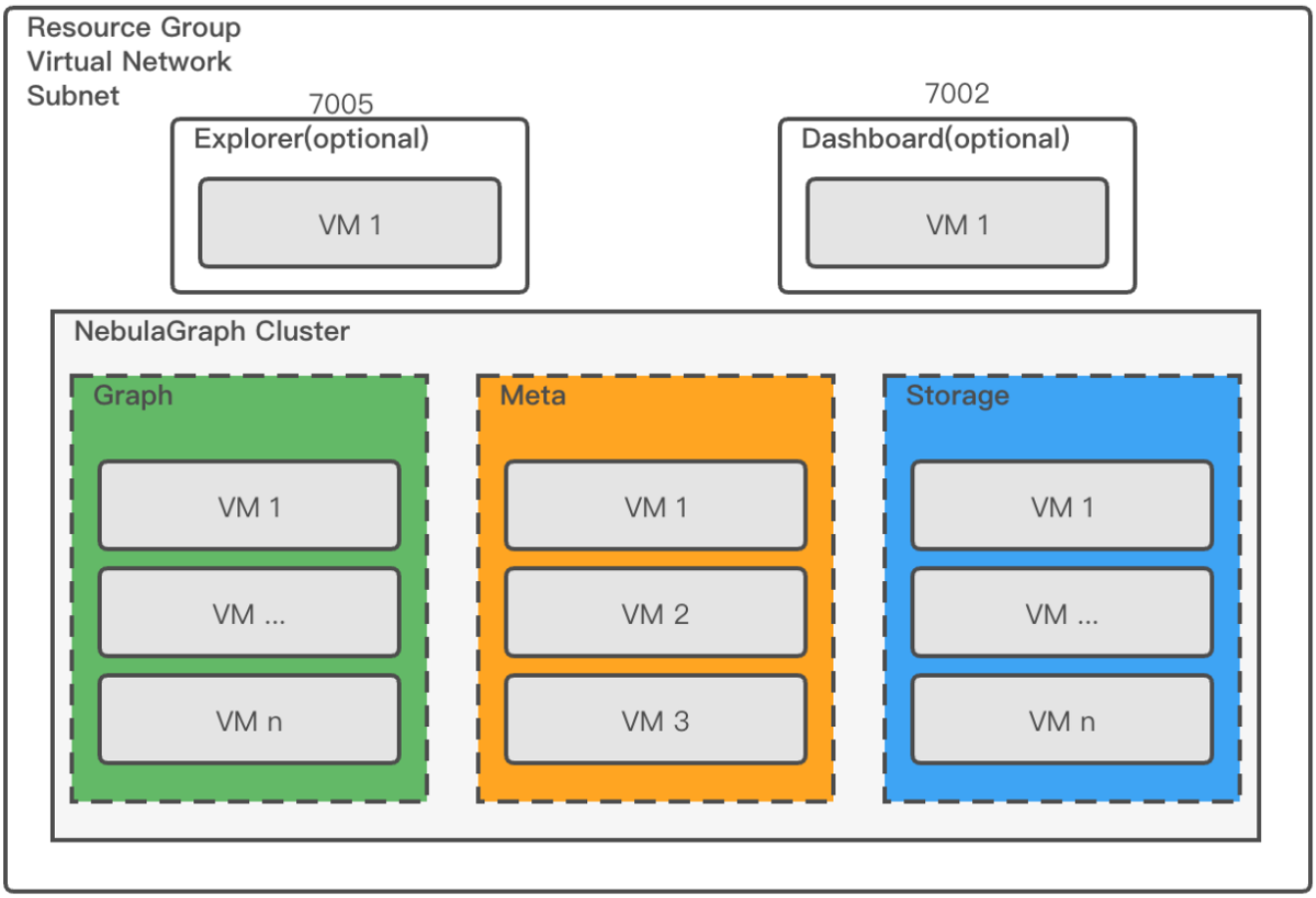
最后更新: June 20, 2023

2022 vesoft Inc.

13.4.2 部署架构

在 Azure Marketplace 创建 NebulaGraph 企业版集群时可以按需调节集群组件。本文介绍生产环境中推荐的部署架构。

生产环境中典型的集群架构如下图所示。



上图中的集群包含：

- 一个用于管理 NebulaGraph 企业版集群所需资源的 Azure 资源组。
- 一个 Azure 虚拟网络中的子网，用于部署 NebulaGraph 企业版集群的各组件。
- 分别由 metad、graphd、storaged 进程支持的 NebulaGraph Meta 服务、Graph 服务和 Storage 服务。每个进程运行在一台 Azure 虚拟机中。
- 一台 NebulaGraph Explorer 服务器，用于支持可视化图探索。
- 一台 NebulaGraph Dashboard 服务器，用于监控和管理 NebulaGraph 服务。

最后更新: June 20, 2023

2022 vesoft Inc.

13.4.3 部署准备

开始部署 NebulaGraph Cloud Azure 版的服务集群前，用户需要具备相关的 Azure 和 NebulaGraph 知识、准备好 Azure 账号以及部署所需的资源。

知识准备

用户需要熟悉 Azure 的常用服务，尤其是资源组、虚拟网络和虚拟机。Azure 新用户可以通过 自主学习路径 了解相应的知识。

账户与权限

用户需准备 Azure 账户。如果没有，根据 Azure 文档的说明申请新账户。

部署集群使用的 Azure 账户必须具有使用使用相应资源的权限。

资源额度

部署前需要确认 Azure 资源的额度，防止资源额度不足导致部署失败。部署一个 NebulaGraph 企业版集群需要的资源见下表。

资源	部署需求数
虚拟网络	1
网络接口	与虚拟机数量相同
网络安全组	3 to 5 （每个 NebulaGraph 服务一个安全组）
可用性集	3
虚拟机 （部署 NebulaGraph Dashboard）	1
虚拟机 （部署 NebulaGraph Explorer）	1
虚拟机 （部署 graphd 进程）	1 到 10
虚拟机 （部署 metad 进程）	1 或 3
虚拟机 （部署 storaged 进程）	1 到 100

网络接口、网络安全组和虚拟机的数量由实际部署架构决定。

如果所需资源的额度不足，先申请提升资源额度再部署集群。

最后更新: June 20, 2023

2022 vesoft Inc.

13.4.4 部署步骤

完成准备后即可开始部署 NebulaGraph 企业版集群。

1. 登录 Azure Marketplace。
2. 搜索 **NebulaGraph Enterprise**，之后单击搜索到的 **NebulaGraph Enterprise** 卡片跳转到产品概览页。
3. 选择默认的 **Plan** 并单击 **Create**。
4. 在 **Basics** 标签页, 完成下方配置，之后单击页面底部的 **Next: Cluster Settings** 按钮。

- Project details

字段	描述
Subscription	选择一个 Subscription。一个 Azure subscription 中的所有资源是统一计费的。
Resource group	选择一个资源组。仅支持使用空的资源组。如果没有空资源组，单击下拉列表下方的 **Create new** 创建新的资源组。

- Instance details

字段	描述
Region	选择一个部署 NebulaGraph 的 Azure 区域。
Username	指定一个用户名，用于登录 NebulaGraph 企业版集群中的虚拟机。
Authentication type	选择访问虚拟机的认证类型。如果选择 **Password**，在下方的字段中设置并确认登录虚拟机的密码。如果选择 **SSH Public Key**，在下方的字段中设置 SSH key 信息。

5. 在 **Cluster Settings** 标签页，完成下方配置，之后单击页面底部的 **Next: Nodes Configuration** 按钮。

字段	描述
NebulaGraph version	选择 NebulaGraph 企业版内核的版本。
Cluster name	设置 NebulaGraph 企业版集群的名称。
Virtual network	选择一个用于部署 NebulaGraph 企业版集群的 Azure 虚拟网络。
NebulaGraph nodes subnet	在虚拟网络中选择一个子网，用于部署 NebulaGraph 企业版集群。

6. 在 **Nodes Configuration** 标签页，为 Graph、Meta 和 Storage 服务设置虚拟机规格、磁盘类型、磁盘容量和节点数量，之后单击页面底部的 **Next: Workbench** 按钮。
7. 在 **Workbench** 标签页，根据需要开启或关闭 Explorer 和 Dashboard 服务，并为要开启的服务选择虚拟机规格，之后单击页面底部的 **Next: Certificates** 按钮。
8. 在 **Certificates** 标签页，根据需要为 Explorer 和 Dashboard 设置 SSL/TLS 证书，之后单击页面底部的 **Next: Review + create** 按钮。
9. 在 **Review + create** 标签页，确认集群配置信息并单击页面底部的 **Create** 按钮开始部署。

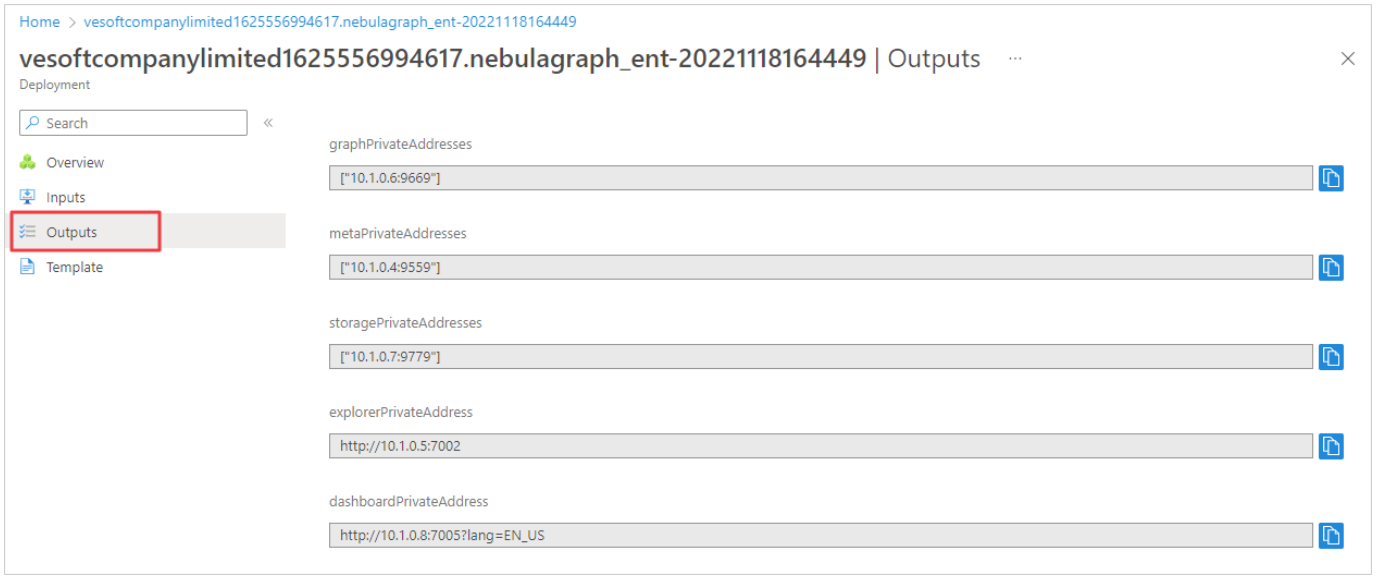
   单击 **Create** 按钮后将跳转到 **Overview** 页面，此时该页显示 **Deployment is in progress**，表示部署正在进行中。
10. 监控 **Overview** 页面，直到页面上显示 **Your deployment is complete**。

最后更新: June 20, 2023

2022 vesoft Inc.

13.4.5 连接服务

完成 NebulaGraph 服务集群部署后，用户可以在部署详情页的 **Outputs** 页面中查看各服务的连接地址，也可以在虚拟机的详细信息中查看。



使用 Nebula Explorer 连接 NebulaGraph 的说明参见连接数据库。

使用客户端连接 NebulaGraph 的说明参见客户端文档。

用户可以使用 Dashboard 监控 NebulaGraph 企业版集群。详情参见 Dashboard 文档。

最后更新: June 20, 2023

2022 vesoft Inc.

14. Nebula Studio

14.1 Studio 版本更新说明

14.1.1 v3.3.2（2022.05.19）

- 修复
- 修复 path 只有起点时渲染出错的问题。

14.1.2 v3.3.1（2022.05.07）

- 修复
- 修复导入日志偶尔为空的问题。

14.1.3 v3.3.0（2022.04.25）

- 功能增强:
- 优化整体 UI 风格。
- Schema
- 支持克隆图空间。
- 支持重建索引。
- 支持图空间数据统计。
- 导入
- 支持多任务异步导入，可随时查看进度，日志等。
- 支持 Nebula Importer 模板快速导入。
- 控制台
- 支持收藏夹功能。
- 支持历史查询结果展示。
- 新增图探索模块至控制台模块内。

最后更新: June 20, 2023

2022 vesoft Inc.

14.2 认识 Nebula Studio

14.2.1 什么是 Nebula Studio

Nebula Studio（简称 Studio）是一款可以通过 Web 访问的开源图数据库可视化工具，搭配 Nebula Graph 内核使用，提供构图、数据导入、编写 nGQL 查询等一站式服务。用户可以在 Nebula Graph GitHub 仓库中查看最新源码，详情参见 nebula-studio。

> **Note**
>
> 用户也可以在线试用 Studio 部分功能。

发行版本

可以使用以下方式安装部署 Studio：

- 用户可以使用 Docker 服务、RPM 服务、tar 包和 DEB 包安装部署 Studio，并连接到 Nebula Graph 数据库。详细信息参考 部署 Studio。
- 使用 Helm 安装部署：在 Kubernetes 集群里使用 Helm 安装并部署 Studio，并连接到 Nebula Graph 数据库。详细信息参考使用 Helm 部署 Nebula Studio。

几种部署方式功能基本相同，在使用 Studio 时可能会受到限制。详细信息，参考使用限制。

产品功能

Studio 可以方便管理 Nebula Graph 图数据，具备以下功能：

- 使用 **Schema** 管理功能，用户可以使用图形界面完成图空间、Tag（标签）、Edge Type（边类型）、索引的创建，查看图空间的统计数据，快速上手 Nebula Graph。
- 使用导入功能，通过简单的配置，用户即能批量导入点和边数据，并能实时查看数据导入日志。
- 使用控制台功能，用户可以使用 nGQL 语句创建 Schema，并对数据执行增删改查操作。

适用场景

如果有以下任一需求，都可以使用 Studio：

- 已经安装部署了 Nebula Graph 数据库，想使用 GUI 工具创建 Schema、导入数据、执行 nGQL 语句查询。
- 刚开始学习 nGQL（Nebula Graph Query Language），但是不习惯用命令行工具，更希望使用 GUI 工具查看语句输出的结果。

身份验证

因为 Nebula Graph 默认不启用身份验证，所以用户可以使用 `root` 账号和任意密码登录 Studio。

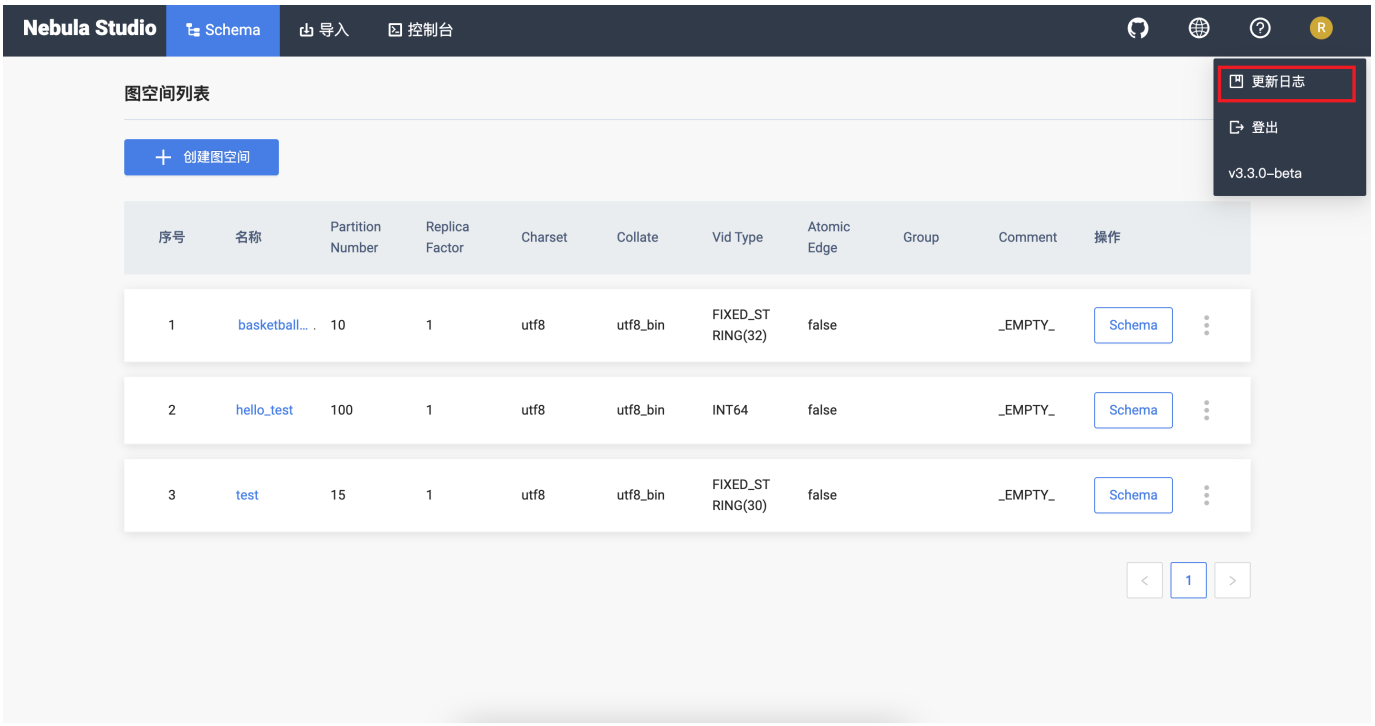
当 Nebula Graph 启用了身份验证后，用户只能使用指定的账号和密码登录 Studio。关于 Nebula Graph 的身份验证功能，参考 身份验证。

版本更新

Studio 处于持续开发状态中。用户可以通过 Studio 版本更新说明查看最新发布的功能。

成功连接 Studio 后，用户可以在页面右上角点击用户头像，再点击 更新日志，查看 Studio 的版本更新说明。

2022 vesoft Inc.



最后更新: June 20, 2023

2022 vesoft Inc.

14.2.2 使用限制

本文描述使用 Studio 的限制。

Nebula Graph 版本支持

> **Note**
>
> Studio 版本发布节奏独立于 Nebula Graph 内核，其命名方式也不参照内核命名规则，两者兼容对应关系如下表。

Nebula Graph 版本	**Studio** 版本
1.x	1.x
2.0 & 2.0.1	2.x
2.5.x	3.0.0
2.6.x	3.1.x
3.0.0	3.2.x
3.1.0	3.3.2
3.0.0 ～ 3.1.0	3.4.0

系统架构

Studio 目前仅支持 x86_64 架构。

数据上传

Studio 上传数据仅支持上传无表头的 CSV 文件，但是，单个文件大小及保存时间不受限制，而且数据总量以本地存储容量为准。

数据备份

目前仅支持在 控制台 上以 CSV 格式导出查询结果，不支持其他数据备份方式。

nGQL 支持

除以下内容外，用户可以在 控制台 上执行所有 nGQL 语句：

- `USE <space_name>`：只能在 **Space** 下拉列表中选择图空间，不能运行这个语句选择图空间。
- 控制台 上使用 nGQL 语句时，用户可以直接回车换行，不能使用换行符。

关于语句的详细信息，参考用户管理。

浏览器支持

建议使用最新版本的 Chrome 访问 Studio。

最后更新: June 20, 2023

2022 vesoft Inc.

14.3 安装与登录

14.3.1 部署 Studio

本文介绍如何在本地通过 RPM、DEB、tar 包和 Docker 部署 Studio。

RPM 部署 Studio

前提条件

在部署 RPM 版 Studio 之前，用户需要确认以下信息：

- Nebula Graph 服务已经部署并启动。详细信息，参考 Nebula Graph 安装部署。
- 使用的 Linux 发行版为 CentOS ，已安装 lsof。
- 确保以下端口未被占用。

端口号	说明
7001	Studio 提供 web 服务使用。

安装

1. 根据需要选择并下载 RPM 包，建议选择最新版本。常用下载链接如下：

安装包	检验和	适用 Nebula 版本
nebula-graph-studio-3.3.2.x86_64.rpm	nebula-graph-studio-3.3.2.x86_64.rpm.sha256	3.1.0

2. 使用 `sudo rpm -i <rpm_name>` 命令安装 RPM 包。

   例如，安装 Studio 3.3.2 版本需要运行以下命令，默认安装路径为 `/usr/local/nebula-graph-studio` ：

   ```
   $ sudo rpm -i nebula-graph-studio-3.3.2.x86_64.rpm
   ```

   也可以使用以下命令安装到指定路径：

   ```
   $ sudo rpm -i nebula-graph-studio-3.3.2.x86_64.rpm --prefix=<path>
   ```

   当屏幕返回以下信息时，表示 PRM 版 Studio 已经成功启动。

   ```
   Start installing Nebula Studio now...
   Nebula Studio has been installed.
   Nebula Studio started automatically.
   ```

3. 启动成功后，在浏览器地址栏输入 `http://<ip address>:7001` 。

   如果在浏览器窗口中能看到以下登录界面，表示已经成功部署并启动 Studio。

2022 vesoft Inc.



卸载

用户可以使用以下的命令卸载 Studio。

```
$ sudo rpm -e nebula-graph-studio-3.3.2.x86_64
```

当屏幕返回以下信息时，表示 PRM 版 Studio 已经卸载。

```
Nebula Studio removed, bye~
```

异常处理

如果在安装过程中自动启动失败或是需要手动启动或停止服务，请使用以下命令：

- 手动启动服务

  ```
  $ bash /usr/local/nebula-graph-studio/scripts/rpm/start.sh
  ```

- 手动停止服务

  ```
  $ bash /usr/local/nebula-graph-studio/scripts/rpm/stop.sh
  ```

如果启动服务时遇到报错报错 `ERROR: bind EADDRINUSE 0.0.0.0:7001`，用户可以通过以下命令查看端口 7001 是否被占用。

```
$ lsof -i:7001
```

如果端口被占用，且无法结束该端口上进程，用户可以通过以下命令修改 Studio 服务启动端口，并重新启动服务。

```
//修改 studio 服务启动端口
$ vi config/example-config.yaml

//修改
web:
#  task_id_path:
#  upload_dir:
#  tasks_dir:
#  sqlitedb_file_path:
#  ip:
  port: 7001 // 修改这个端口号，改成任意一个当前可用的即可
```


2022 vesoft Inc.

```
//重启服务
$ systemctl restart nebula-graph-studio.service
```

DEB 部署 Studio

前提条件

在通过 DEB 部署安装 Studio 之前，用户需要确认以下信息：

- Nebula Graph 服务已经部署并启动。详细信息，参考 Nebula Graph 安装部署。
- 使用的 Linux 发行版为 Ubuntu。
- 确保以下端口未被占用。

端口号	说明
7001	Studio 提供的 web 服务

- 确保系统中存在 `/usr/lib/systemd/system` 目录。如没有该目录，需手动创建。

安装

1. 根据需要选择并下载 DEB 包，建议选择最新版本。常用下载链接如下：

安装包	检验和	适用 **Nebula** 版本
nebula-graph-studio-3.3.2.x86_64.deb	nebula-graph-studio-3.3.2.x86_64.deb.sha256	3.1.0

2. 使用 `sudo dpkg -i <deb_name>` 命令安装 DEB 包。

   例如，安装 Studio 3.3.2 版本需要运行以下命令：

```
$ sudo dpkg -i nebula-graph-studio-3.3.2.x86_64.deb
```

3. 启动成功后，在浏览器地址栏输入 `http://<ip address>:7001` 。

   如果在浏览器窗口中能看到以下登录界面，表示已经成功部署并启动 Studio。



2022 vesoft Inc.

卸载

用户可以使用以下的命令卸载 Studio。

```
$ sudo dpkg -r nebula-graph-studio
```

tar 包部署 Studio

前提条件

在部署 tar 包安装的 Studio 之前，用户需要确认以下信息：

- Nebula Graph 服务已经部署并启动。详细信息，参考 Nebula Graph 安装部署。
- 确保以下端口未被占用。

端口号	说明
7001	Studio 提供的 web 服务

安装部署

1. 根据需要下载 tar 包，建议选择最新版本。

安装包	**Studio** 版本	适用 **Nebula** 版本
nebula-graph-studio-3.3.2.x86_64.tar.gz	3.3.2	3.1.0

2. 解压 tar 包。

```
tar -xvf nebula-graph-studio-3.3.2.x86_64.tar.gz
```

3. 部署 nebula-graph-studio 并启动。

```
$ cd nebula-graph-studio
$ ./server
```

4. 启动成功后，在浏览器地址栏输入 `http://<ip address>:7001`。

   如果在浏览器窗口中能看到以下登录界面，表示已经成功部署并启动 Studio。



2022 vesoft Inc.

停止服务

用户可以采用 `kill <pid>` 的方式来关停服务：

```
$ kill $(lsof -t -i :7001) # stop nebula-graph-studio
```

Docker 部署 Studio

前提条件

在部署 Docker 版 Studio 之前，用户需要确认以下信息：

- Nebula Graph 服务已经部署并启动。详细信息，参考 Nebula Graph 安装部署。
- 在即将运行 Docker 版 Studio 的机器上安装并启动 Docker Compose。详细信息参考 Docker Compose 文档。
- 确保以下端口未被占用。

端口号	说明
7001	Studio 提供的 web 服务

- （可选）在中国大陆从 Docker Hub 拉取 Docker 镜像的速度可能比较慢，用户可以使用 `registry-mirrors` 参数配置加速镜像。例如，如果要使用 Docker 中国区官方镜像、网易镜像和中国科技大学的镜像，则按以下格式配置 `registry-mirrors` 参数：

  ```
  {
  "registry-mirrors": [
    "https://registry.docker-cn.com",
    "http://hub-mirror.c.163.com",
    "https://docker.mirrors.ustc.edu.cn"
    ]
  }
  ```

  配置文件的路径和方法因操作系统和/或 Docker Desktop 版本而异。详细信息参考 Docker Daemon 配置文档。

2022 vesoft Inc.

操作步骤

在命令行工具中按以下步骤依次运行命令，部署并启动 Docker 版 Studio，这里我们用 Nebula Graph 版本为 3.1.0 的进行演示：

1. 下载 Studio 的部署配置文件。

安装包	适用 **Nebula** 版本
nebula-graph-studio-3.3.2.tar.gz	3.1.0

2. 创建 `nebula-graph-studio-3.3.2` 目录，并将安装包解压至目录中。

```
mkdir nebula-graph-studio-3.3.2 && tar -zxvf nebula-graph-studio-3.3.2.tar.gz -C nebula-graph-studio-3.3.2
```

3. 解压后进入 `nebula-graph-studio-3.3.2` 目录。

```
cd nebula-graph-studio-3.3.2
```

4. 拉取 Studio 的 Docker 镜像。

```
docker-compose pull
```

5. 构建并启动 Studio 服务。其中，`-d` 表示在后台运行服务容器。

```
docker-compose up -d
```

当屏幕返回以下信息时，表示 Docker 版 Studio 已经成功启动。

```
Creating docker_web_1     ... done
```

6. 启动成功后，在浏览器地址栏输入 `http://<ip address>:7001` 。

Note

在运行 Docker 版 Studio 的机器上，用户可以运行 `ifconfig` 或者 `ipconfig` 获取本机 IP 地址。如果使用这台机器访问 Studio，可以在浏览器地址栏里输入 `http://localhost:7001` 。

如果在浏览器窗口中能看到以下登录界面，表示已经成功部署并启动 Studio。



2022 vesoft Inc.

后续操作

进入 Studio 登录界面后，用户需要连接 Nebula Graph。详细信息，参考连接数据库。

最后更新: June 20, 2023

2022 vesoft Inc.

14.3.2 使用 Helm 部署 Studio

本文介绍如何在 Kubernetes 中使用 Helm 部署并启动 Studio。

前提条件

安装 Studio 前，用户需要安装以下软件并确保安装版本的正确性：

软件	版本要求
Kubernetes	>= 1.14
Helm	>= 3.2.0

安装

1. 克隆 Studio 的源代码到主机。

```
$ git clone https://github.com/vesoft-inc/nebula-studio.git
```

2. 进入 `nebula-studio` 目录。

```
$ cd nebula-studio
```

3. 更新并安装 Nebula Studio chart，命名为 `my-studio`。

```
$ helm upgrade --install my-studio --set service.type=NodePort --set service.port={30070} deployment/helm
```

4. 启动成功后，在浏览器地址栏输入 `http://<node_address>:30070`。 如果在浏览器窗口中能看到以下登录界面，表示已经成功部署并启动 Studio。



卸载

```
$ helm uninstall my-studio
```


2022 vesoft Inc.

后续操作

进入 Studio 登录界面后，用户需要连接 Nebula Graph。详细信息，参考连接数据库。

Nebula Studio chart 配置参数说明

参数	默认值	描述
replicaCount	0	Deployment 的副本数。
image.nebulaStudio.name	vesoft/nebula-graph-studio	nebula-graph-studio 镜像的仓库地址。
image.nebulaStudio.version	v3.2.0	nebula-graph-studio 的版本。
service.type	ClusterIP	服务类型，必须为 `NodePort`， `ClusterIP` 或 `LoadBalancer` 其中之一。
service.port	7001	nebula-graph-studio 中 web 服务的端口。
service.nodePort	32701	Kubernetes 集群外部访问 nebula-studio 的代理端口。
resources.nebulaStudio	{}	nebula-studio 的资源限制/请求。
persistent.storageClassName	""	storageClass 名称，如果不指定就使用默认值。
persistent.size	5Gi	存储盘大小。

最后更新: June 20, 2023

2022 vesoft Inc.

14.3.3 连接数据库

在成功启动 Studio 后，用户需要配置连接 Nebula Graph。本文主要描述 Studio 如何连接 Nebula Graph 数据库。

前提条件

在连接 Nebula Graph 数据库前，用户需要确认以下信息：

- Studio 已经启动。详细信息参考部署 Studio。
- Nebula Graph 的 Graph 服务本机 IP 地址以及服务所用端口。默认端口为 `9669`。
- Nebula Graph 数据库登录账号信息，包括用户名和密码。

> **Note**
>
> 如果 Nebula Graph 已经启用了身份验证，并且已经创建了不同角色的用户，用户只能使用被分配到的账号和密码登录数据库。如果未启用身份验证，用户可以使用 `root` 用户名和任意密码登录数据库。关于启用身份验证，参考 Nebula Graph 用户手册。

2022 vesoft Inc.

操作步骤

按以下步骤连接 Nebula Graph 数据库:

2022 vesoft Inc.

1. 在 Studio 的 配置数据库 页面上，输入以下信息：

- **Host**：填写 Nebula Graph 的 Graph 服务本机 IP 地址及端口。格式为 `ip:port`。默认端口为 `9669`。

> **Note**
>
> 即使 Nebula Graph 数据库与 Studio 部署在同一台机器上，用户也必须在 **Host** 字段填写这台机器的本机 IP 地址，而不是 `127.0.0.1` 或者 `localhost`。

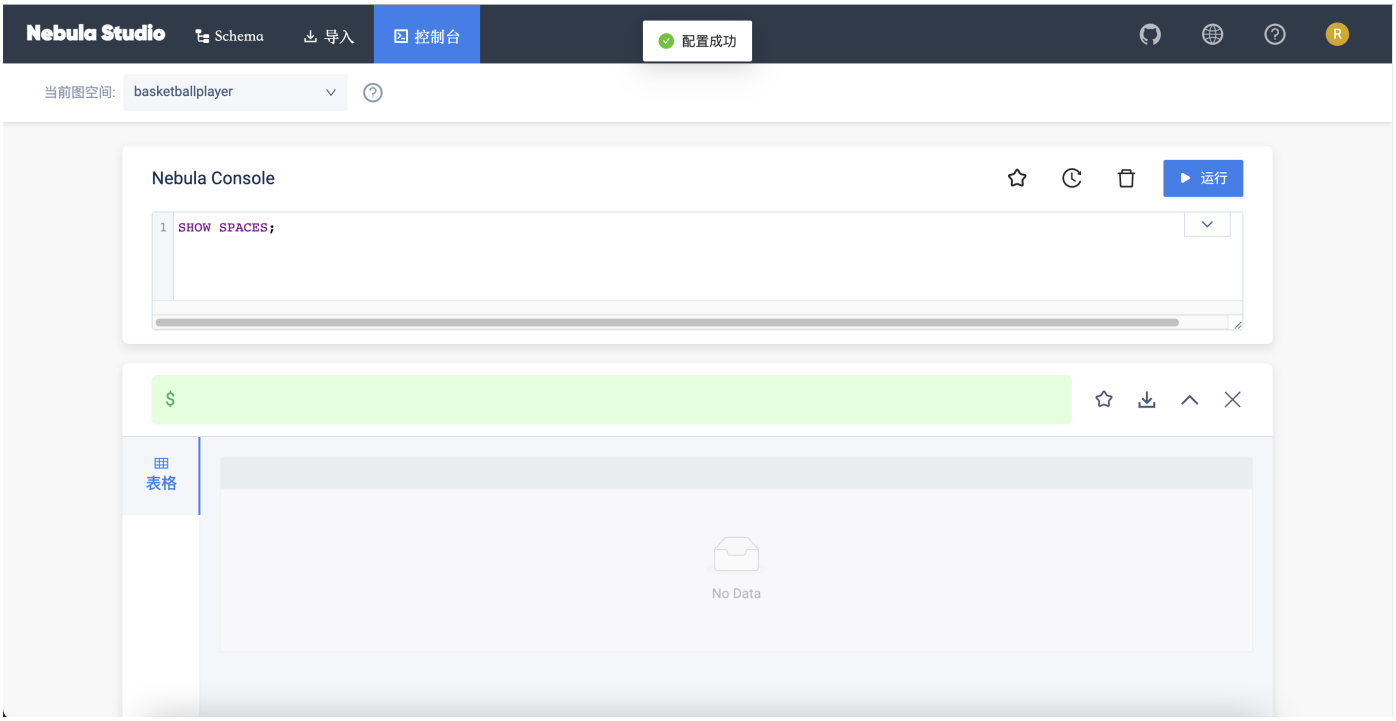
- 用户名 和 密码：根据 Nebula Graph 的身份验证设置填写登录账号和密码。
- 如果未启用身份验证，可以填写默认用户名 `root` 和任意密码。
- 如果已启用身份验证，但是未创建账号信息，用户只能以 GOD 角色登录，必须填写 `root` 及对应的密码 `nebula`。
- 如果已启用身份验证，同时又创建了不同的用户并分配了角色，不同角色的用户使用自己的账号和密码登录。



2. 完成设置后，点击 连接 按钮。

   如果能看到如下图所示的界面，表示已经成功连接到 Nebula Graph 数据库。

2022 vesoft Inc.



一次连接会话持续 30 分钟。如果超过 30 分钟没有操作，会话即断开，用户需要重新登录数据库。

后续操作

成功连接 Nebula Graph 数据库后，根据账号的权限，用户可以选择执行以下操作：

- 拥有 GOD 或者 ADMIN 权限的账号，可以使用控制台或者 **Schema** 页面管理 Schema、批量导入数据、在控制台页面上执行 nGQL 语句。
- 拥有 DBA 或者 USER 权限的账号，可以批量导入数据、在控制台页面上执行 nGQL 语句。
- 拥有 GUEST 权限的账号，仅可以在 控制台 页面上执行 nGQL 语句查询数据。

> **Note**
>
> 账号权限参见 内置角色权限。

登出

如果需要重新连接 Nebula Graph 数据库，可以登出后重新配置数据库。

在页面右上角单击用户头像，单击 登出，如果浏览器上显示 配置数据库 页面，表示 Studio 已经成功断开了与 Nebula Graph 数据库的连接。

2022 vesoft Inc.



最后更新: June 20, 2023

2022 vesoft Inc.

14.4 快速开始

14.4.1 规划 Schema

在使用 Studio 之前，用户需要先根据 Nebula Graph 数据库的要求规划 Schema（模式）。

Schema 至少要包含以下要素：

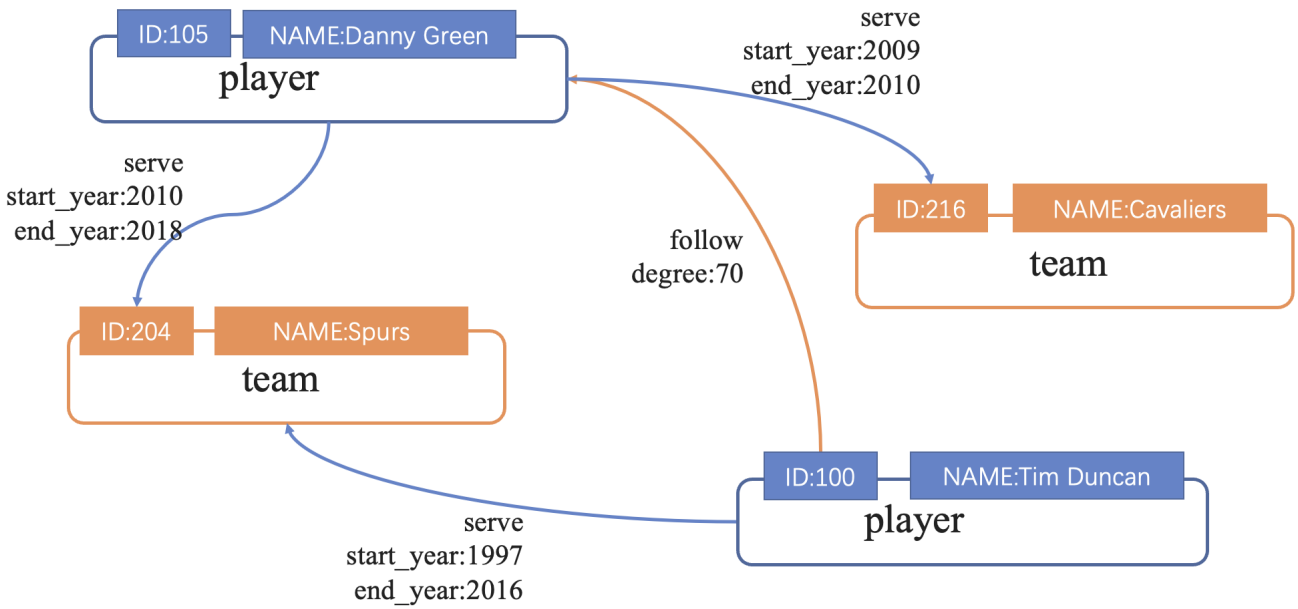
- Tag，以及每种 Tag 的属性。
- Edge type，以及每种 Edge type 的属性。

用户可以下载 Nebula Graph 示例数据集 basketballplayer ，本文将通过该示例说明如何规划 Schema。

下表列出了 Schema 要素。

类型	名称	属性名（数据类型）	说明
Tag	**player**	- `name` （`string`） - `age` （`int`）	表示球员。
Tag	**team**	- `name` （`string`）	表示球队。
Edge type	**serve**	- `start_year` （`int`） - `end_year` （`int`）	表示球员的行为。 该行为将球员和球队联系起来，方向是从球员到球队。
Edge type	**follow**	- `degree` （`int`）	表示球员的行为。 该行为将两个球员联系起来，方向是从一个球员到另一个球员。

下图说明示例中 **player** 类点与 **team** 类点之间如何发生关系（**serve/follow**）。



最后更新: June 20, 2023

2022 vesoft Inc.

14.4.2 创建 Schema

在 Nebula Graph 中，用户必须先有 Schema，才能向其中写入点数据和边数据。本文描述如何使用 Nebula Graph 的控制台或 **Schema** 功能创建 Schema。

> **Note**
>
> 用户也可以使用 nebula-console 创建 Schema。详细信息，参考 Nebula Graph 使用手册和 Nebula Graph 快速开始。

前提条件

在 Studio 上创建 Schema 之前，用户需要确认以下信息：

- Studio 已经连接到 Nebula Graph 数据库。
- 账号拥有 GOD、ADMIN 或 DBA 权限。详细信息，参考 Nebula Graph 内置角色。
- 已经规划 Schema 的要素。
- 已经创建图空间。

> **Note**
>
> 本示例已经创建图空间。如果账号拥有 GOD 权限，也可以在 控制台 或 **Schema** 上创建一个图空间。

使用 **Schema** 管理功能创建 **Schema**

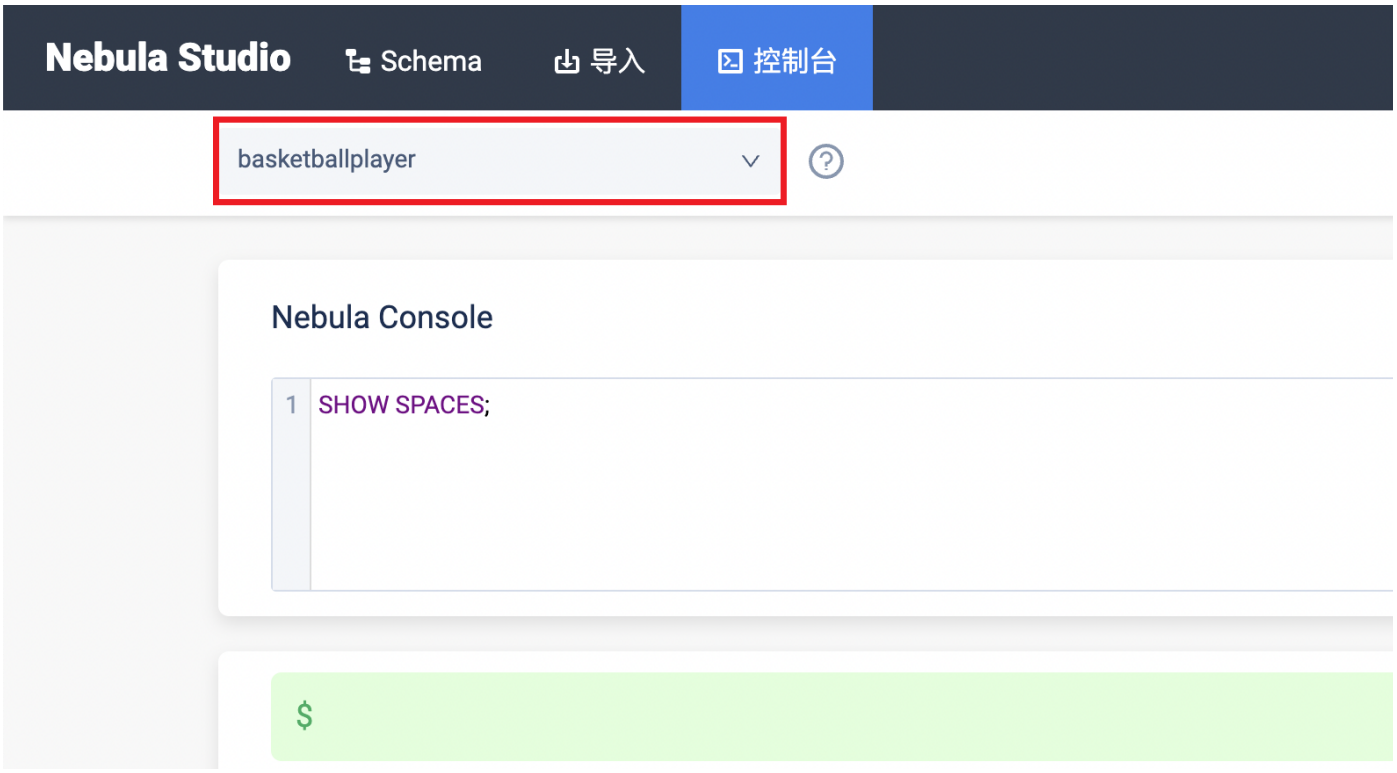
按以下步骤使用 **Schema** 管理功能创建 Schema：

1. 创建 Tag。详细信息，参考操作 Tag。
2. 创建 Edge type。详细信息，参考操作 Edge type。

使用控制台创建 **Schema**

1. 在顶部导航栏里，点击 控制台 页签。
2. 在 当前 **Space** 中选择一个图空间。在本示例中，选择 **basketballplayer**。

2022 vesoft Inc.



3. 在命令行中，依次输入以下语句，并点击右侧的运行按钮。

```
// 创建 Tag player，带有 2 个属性
CREATE TAG player(name string, age int);

// 创建 Tag team，带有 1 个属性
CREATE TAG team(name string);

// 创建 Edge type follow，带有 1 个属性
CREATE EDGE follow(degree int);

// 创建 Edge type serve，带有 2 个属性
CREATE EDGE serve(start_year int, end_year int);
```

至此，用户已经完成了 Schema 创建。用户可以运行以下语句查看 Tag 与 Edge type 的定义是否正确、完整。

```
// 列出当前图空间中所有 Tag
SHOW TAGS;

// 列出当前图空间中所有 Edge type
SHOW EDGES;

// 查看每种 Tag 和 Edge type 的结构是否正确
DESCRIBE TAG player;
DESCRIBE TAG team;
DESCRIBE EDGE follow;
DESCRIBE EDGE serve;
```

后续操作

创建 Schema 后，用户可以开始导入数据。

最后更新: June 20, 2023

2022 vesoft Inc.

14.4.3 导入数据

准备好 CSV 文件，创建了 Schema 后，用户可以使用 导入 功能将所有点和边数据上传到 Studio，用于数据查询和数据分析。

前提条件

导入数据之前，需要确认以下信息：

- Studio 已经连接到 Nebula Graph 数据库。
- Nebula Graph 数据库里已经创建 Schema。
- CSV 文件符合 Schema 要求。
- 账号拥有 GOD、ADMIN、DBA 或者 USER 的权限，能往图空间中写入数据。

操作步骤

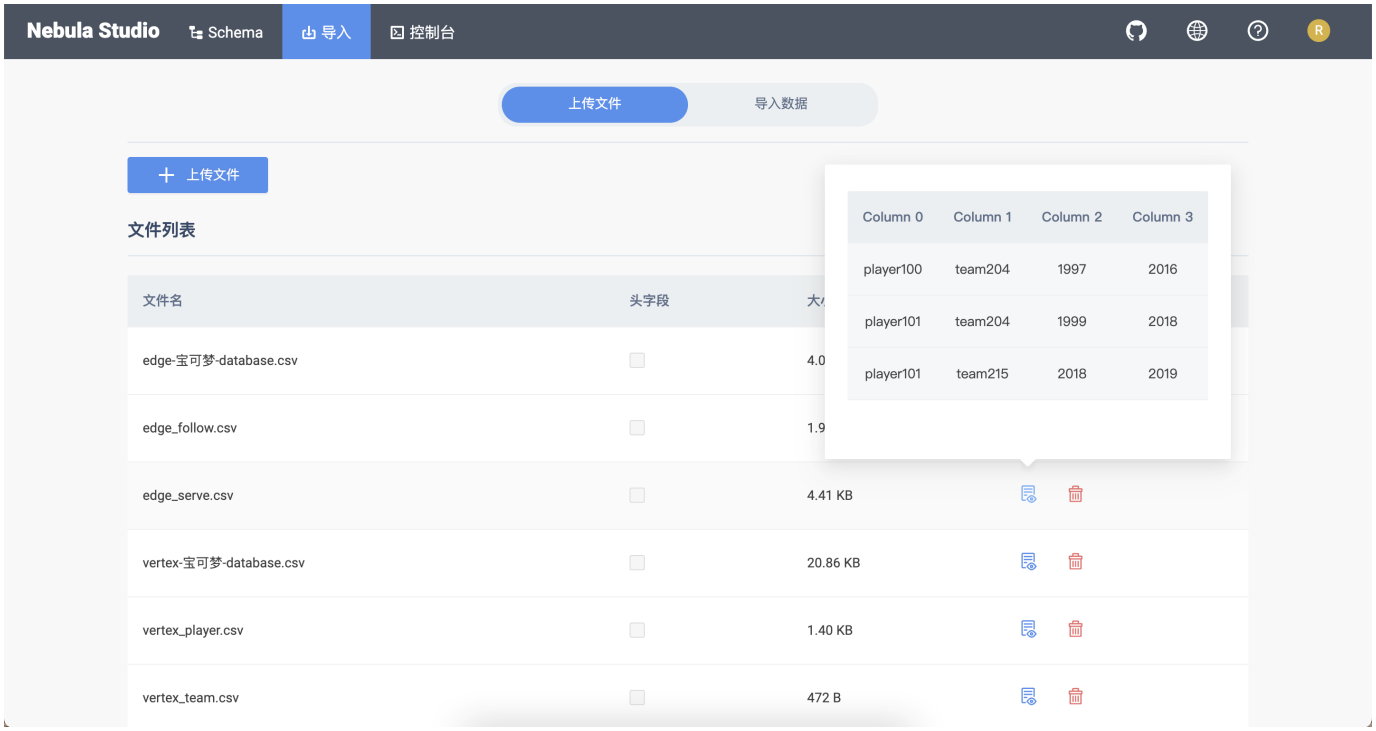
在导入数据之前，用户需要先上传文件后再创建导入任务。

上传文件

1. 在顶部导航栏里，点击 导入 页签。
2. 在 上传文件 页面，点击 上传文件 按钮，并选择需要的 CSV 文件。本示例中，选择 `edge_serve.csv`、`edge_follow.csv`、`vertex_player.csv` 和 `vertex_team.csv` 文件。

> **Note**
>
> 一次可以选择多个 CSV 文件，本文使用的 CSV 文件可以在规划 Schema 中下载。

3. 上传结束后，可以在文件列表的 操作 列，点击  图标预览文件内容，或点击  图标删除上传的文件。



2022 vesoft Inc.

导入数据

按照以下步骤导入数据:

2022 vesoft Inc.

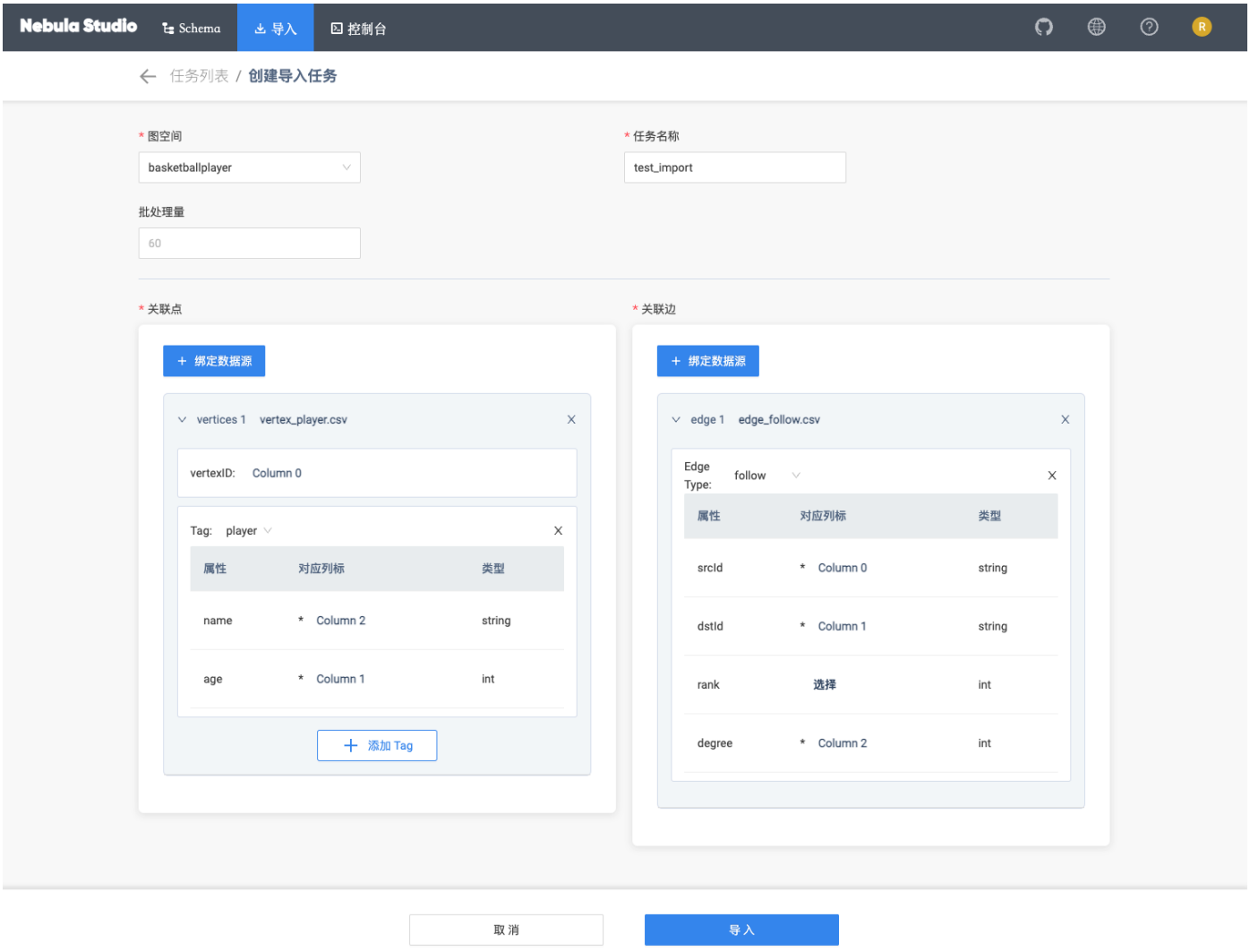
1. 在顶部导航栏里，点击 导入 页签。
2. 在标签页内点击 导入数据 按钮。
3. 在 导入数据 页面，点击 + 创建导入任务 按钮，完成以下任务：

2022 vesoft Inc.

Caution

用户也可以点击 导入模版 ，下载示例配置文件 `example.yaml`，配置后再上传配置文件。配置方式与 Nebula Importer 大致相同，但是所有文件路径仅保留文件名。并且请确保在导入配置文件之前已上传所有 CSV 数据文件。

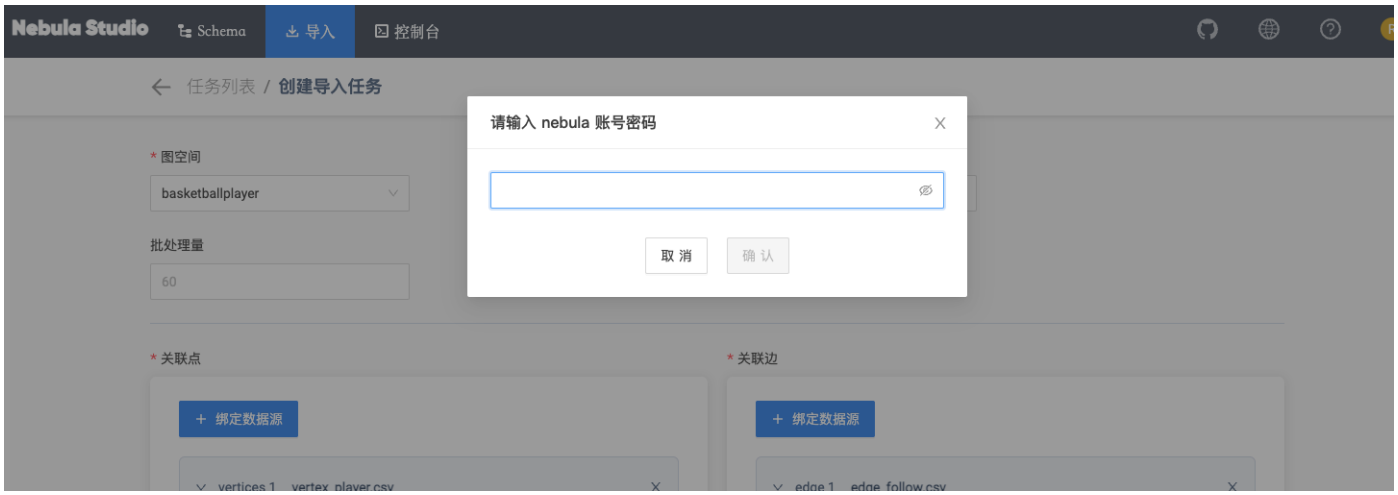
- 选择图空间。
- 填写任务名称。
- （可选）填写批处理量。
- 在 关联点 页签里，点击 **+** 绑定数据源 按钮，在对话框中选择绑定文件，并点击 确认 按钮。如本示例中的 `vertex_player.csv` 文件。
- 在 **vertices 1** 页签下的 `verteID` 项中，点击 **Select CSV Index**，在弹出的对话框内选择 vertexID 所在的列。
- 点击 **+** 添加**Tag** 按钮，点击右方的 图标，在显示的属性列表中，为 Tag 属性绑定源数据。在本示例中，`player` 标签的 `name` 属性对应文件中的 **Column 2** 列，类型为 `string`，`age` 属性对应文件中的 **Column 1** 列，类型 为 `int`。
- 在 关联边 页签里，点击 **+** 绑定数据源 按钮，在对话框中选择绑定文件，并点击 确认 按钮。如本示例中的 `edge_follow.csv` 文件。
- 在 **edge 1** 页签下，点击右方的 图标，在显示的属性列表中，选择 Edge Type。
- 根据 Edge type 的属性，从 `edge_follow.csv` 文件中选择相应的数据列。其中，**srcId** 和 **dstId** 分别表示边的起点与终点，所选择的数据类型必须与 Schema 中的 VID 类型保持一致。本示例中，**srcId** 对应的是表示起点球员的 VID，**dstId** 对应的是表示终点球员的 VID。rank 为选填项，可以忽略。



4. 完成设置后，点击 导入 按钮。

2022 vesoft Inc.

5. 用户输入 Nebula 账号的密码后方可导入数据。



6. 导入数据后可以在 导入数据 页签内查看日志、下载日志、下载配置文件、删除任务等操作。



后续操作

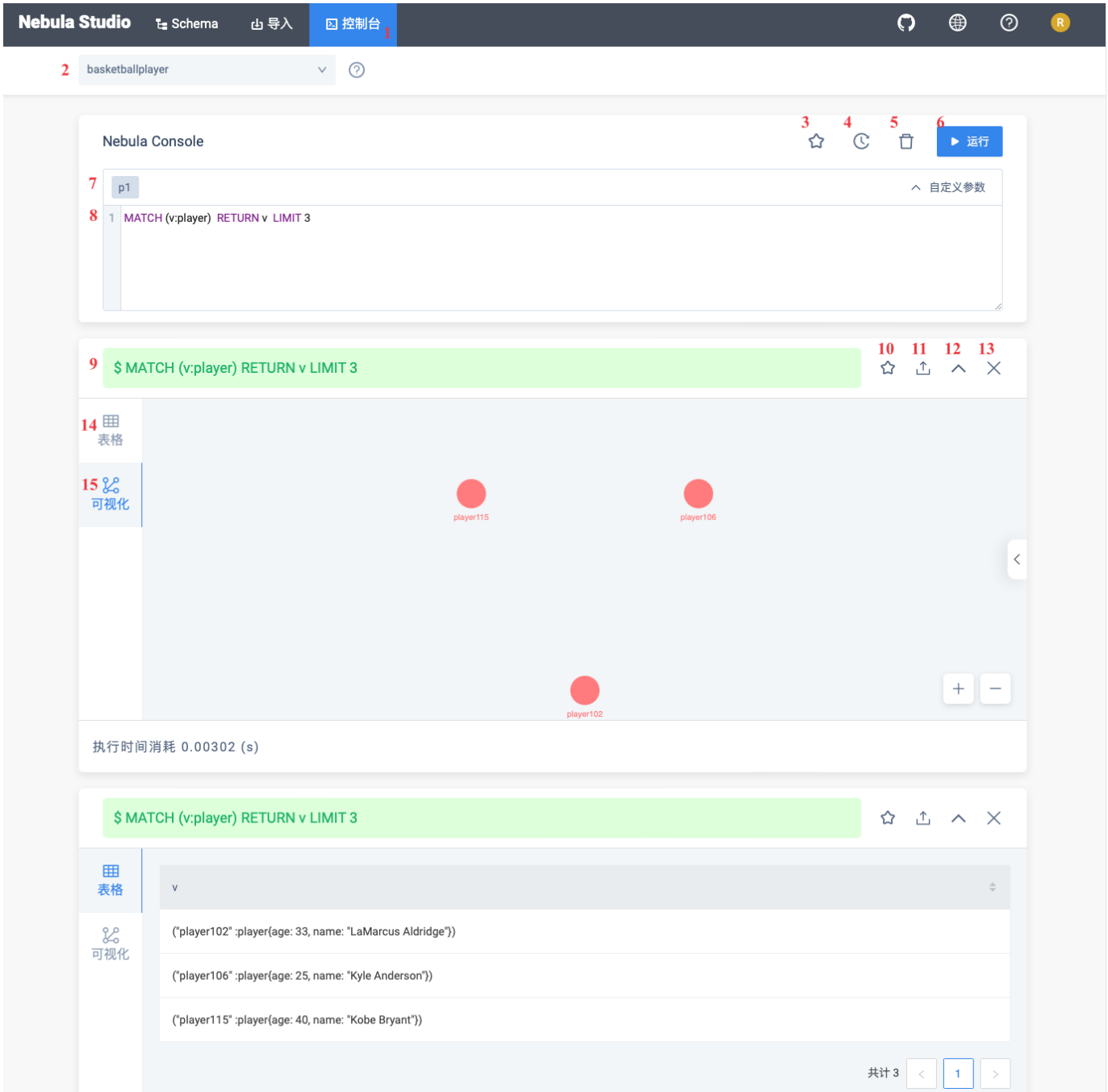
完成数据导入后，用户可以进入控制台页面。

最后更新: June 20, 2023

2022 vesoft Inc.

14.4.4 控制台界面

本文介绍 Studio 的控制台界面。



2022 vesoft Inc.

下表列出了控制台界面上的各种功能。

编号	功能	说明
1	工具栏	点击 控制台 页签进入控制台页面。
2	选择图空间	在 当前图空间 列表中选择一个图空间。 说明：Studio 不支持直接在输入框中运行 `USE <space_name>` 语句。
3	收藏夹	点击  按钮，展开收藏夹，点击其中一个语句，输入框中即自动输入该语句。
4	历史清单	点击  按钮，在语句运行记录列表里，点击其中一个语句，输入框中即自动输入该语句。列表里提供最近 15 次语句运行记录。
5	清空输入框	点击  按钮，清空输入框中已经输入的内容。
6	运行	在输入框中输入 nGQL 语句后，点击  按钮即开始运行语句。
7	自定义参数展示	点击  按钮可展开自定义参数，用于参数化查询，详情信息可见管理参数。
8	输入框	在输入框中输入 nGQL 语句后，点击  按钮运行语句。用户可以同时输入多个语句同时运行，语句之间以 `;` 分隔。
9	语句运行状态	运行 nGQL 语句后，这里显示语句运行状态。如果语句运行成功，语句以绿色显示。如果语句运行失败，语句以红色显示。
10	添加到收藏夹	点击  按钮，将语句存入收藏夹中，已收藏的语句该按钮以黄色展示。
11	导出 CSV 文件或 PNG 格式图片	运行 nGQL 语句返回结果后，返回结果为表格形式时，点击  按钮即能将结果以 CSV 文件的形式导出。切换到可视化窗口，点击  按钮即能将结果以 CSV 文件或 PNG 图片的形式导出。
12	展开/隐藏执行结果	点击  按钮，隐藏此条 nGQL 语句返回的结果或点击  按钮，展开此条 nGQL 语句返回的结果.
13	关闭执行结果	点击  按钮，关闭此条 nGQL 语句返回的结果。
14	表格窗口	显示语句运行结果。如果语句会返回结果，窗口会以表格形式呈现返回的结果。
15	可视化窗口	显示语句运行结果。如果语句会返回完整的点边结果，窗口会以可视化形式呈现返回的结果。点击右方  按钮，展开数据概览面板。

最后更新: June 20, 2023

2022 vesoft Inc.

14.4.5 管理Schema

操作图空间

Studio 连接到 Nebula Graph 数据库后，用户可以创建或删除图空间。用户可以使用 控制台 或者 **Schema** 操作图空间。本文仅说明如何使用 **Schema** 操作图空间。

前提条件

操作图空间之前，用户需要确保以下信息：

- Studio 已经连接到 Nebula Graph 数据库。
- 当前登录的账号拥有创建或删除图空间的权限，即：
- 如果 Nebula Graph 未开启身份验证，用户以默认用户名 `user` 账号和任意密码登录。
- 如果 Nebula Graph 已开启身份验证，用户以 `root` 账号及其密码登录。

创建图空间

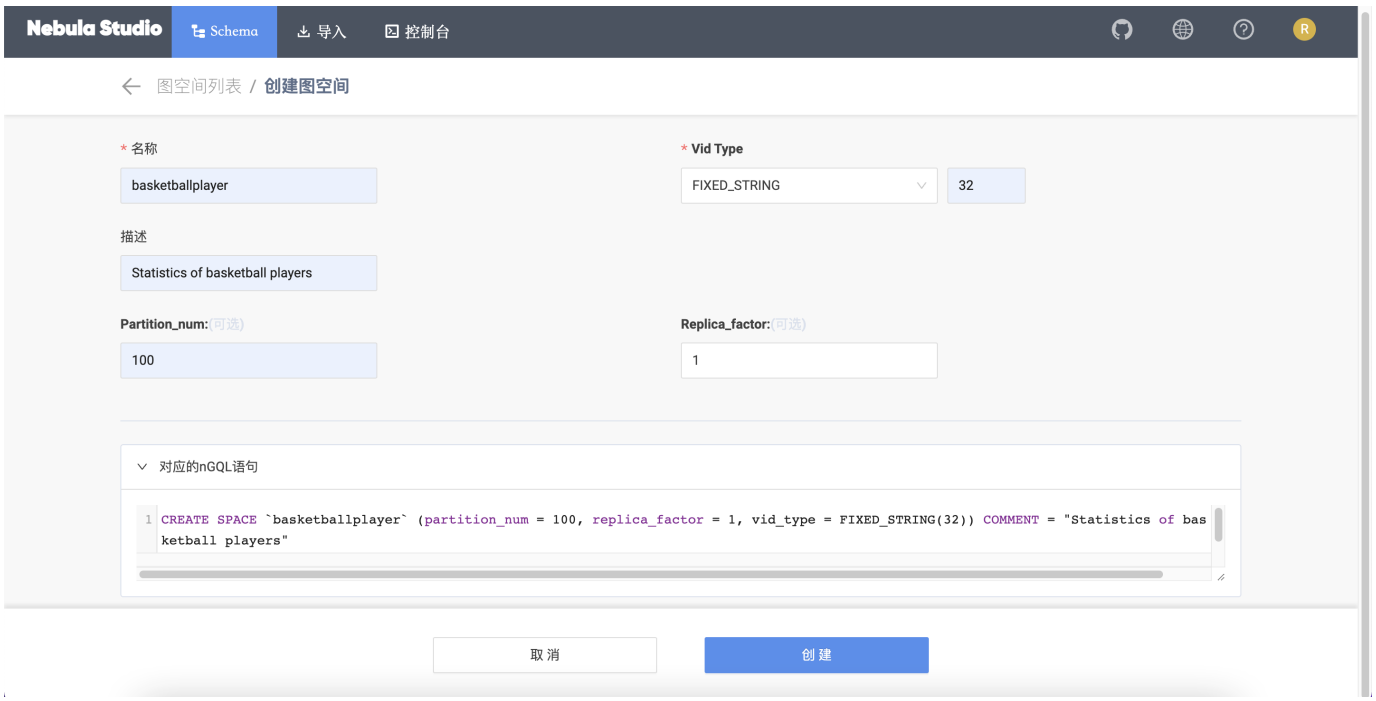
1. 在顶部导航栏里，点击 **Schema** 页签。
2. 在图空间列表上方，点击 **+** 创建图空间 按钮，完成以下配置：

- 名称：指定图空间名称，本示例中设置为 `basketballplayer`。不可与已有的图空间名称重复。
- **vid type**：图空间中点 ID（VID）的数据类型，可选值为定长字符串 `FIXED_STRING(<N>)` 或 `INT64`，一旦定义无法修改。本示例设置为 `FIXED_STRING(32)`。详细信息，参考 VID。
- 描述：图空间的描述，最大为 256 字节。默认无描述。本示例设置为 `Statistics of basketball players`。
- 可选参数：分别设置 `partition_num`、`replica_factor` 的值。在本示例中，两个参数分别设置为 `100`、`1`。详细信息，参考 `CREATE SPACE` 语法。

在 对应的 **nGQL** 语句 面板上，用户能看到上述设置对应的 nGQL 语句。如下所示：

```
CREATE SPACE basketballplayer (partition_num = 100, replica_factor = 1, vid_type = FIXED_STRING(32)) COMMENT = "Statistics of basketball players"
```

3. 配置确认无误后，点击 创建 按钮。 如果页面回到 图空间列表，而且列表中显示刚创建的图空间信息，表示图空间创建成功。



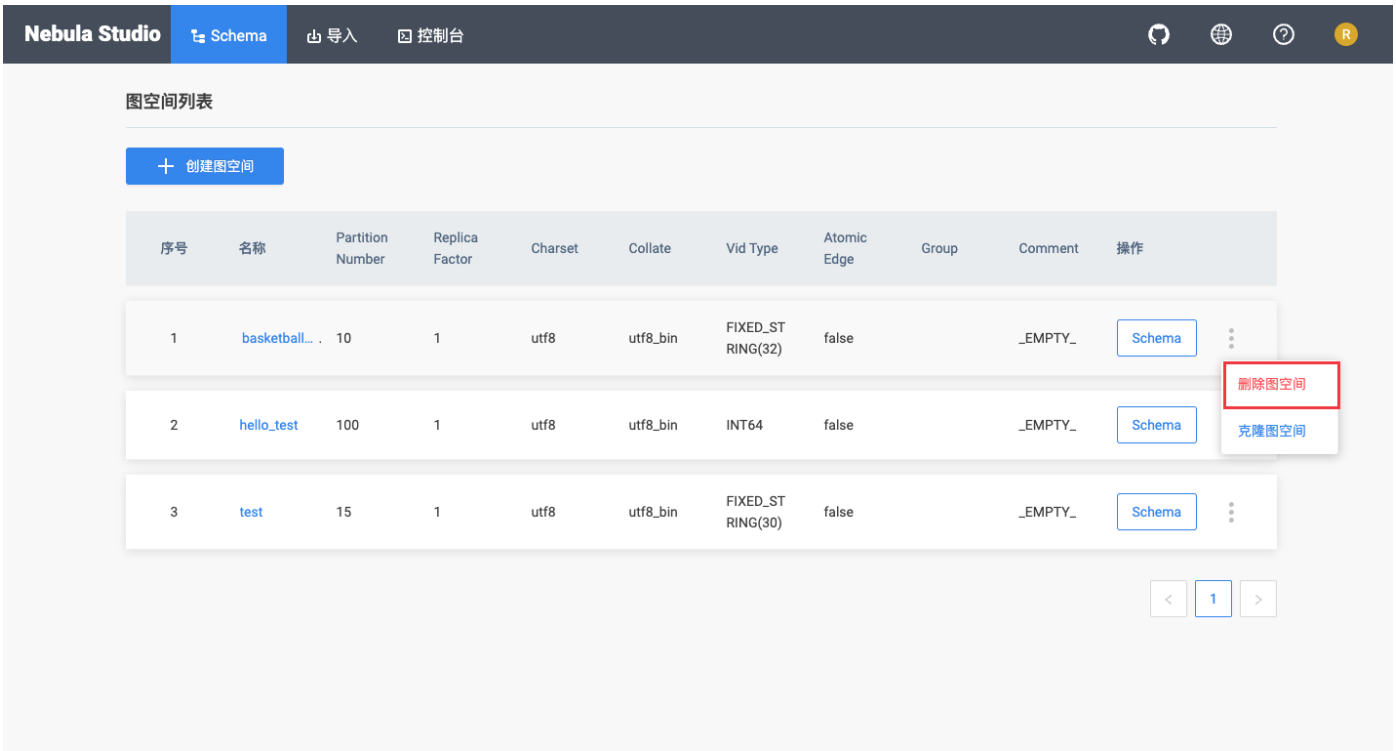
2022 vesoft Inc.

删除图空间

Danger

删除图空间会删除其中的所有数据，已删除的数据如未备份无法恢复。

1. 在顶部导航栏里，点击 **Schema** 页签。
2. 在图空间列表里，找到需要删除的图空间，并在 操作 列中，选择 删除图空间。



3. 在弹出的对话框中点击 确认。

后续操作

图空间创建成功后，用户可以开始创建或修改 Schema，包括：

- 操作 Tag
- 操作 Edge type
- 操作索引

最后更新: June 20, 2023

2022 vesoft Inc.

操作 **Tag**（点类型）

在 Nebula Graph 数据库中创建图空间后，用户需要创建 Tag（点类型）。用户可以选择使用 控制台 或者 **Schema** 管理功能操作 Tag。本文仅说明如何使用 **Schema** 管理功能操作 Tag。

前提条件

在 Studio 上操作 Tag 之前，用户必须确认以下信息：

- Studio 已经连接到 Nebula Graph 数据库。
- 图空间已经创建。
- 当前登录的账号拥有 GOD、ADMIN 或者 DBA 的权限。

创建 **TAG**

1. 在顶部导航栏中，点击 **Schema** 页签。
2. 在 图空间列表 中，找到图空间，点击图空间名称或者在 操作 列中点击 **Schema**。
3. 在 当前图空间 里确认图空间名称。用户也可以通过选择图空间名称切换图空间。
4. 点击 标签 页签，并点击 **+** 创建 按钮。
5. 在 创建标签 页面上，完成以下设置：

a. 名称：输入 Tag 名称。本示例中，输入 `player`。

b. 描述（可选）：输入 Tag 的备注。

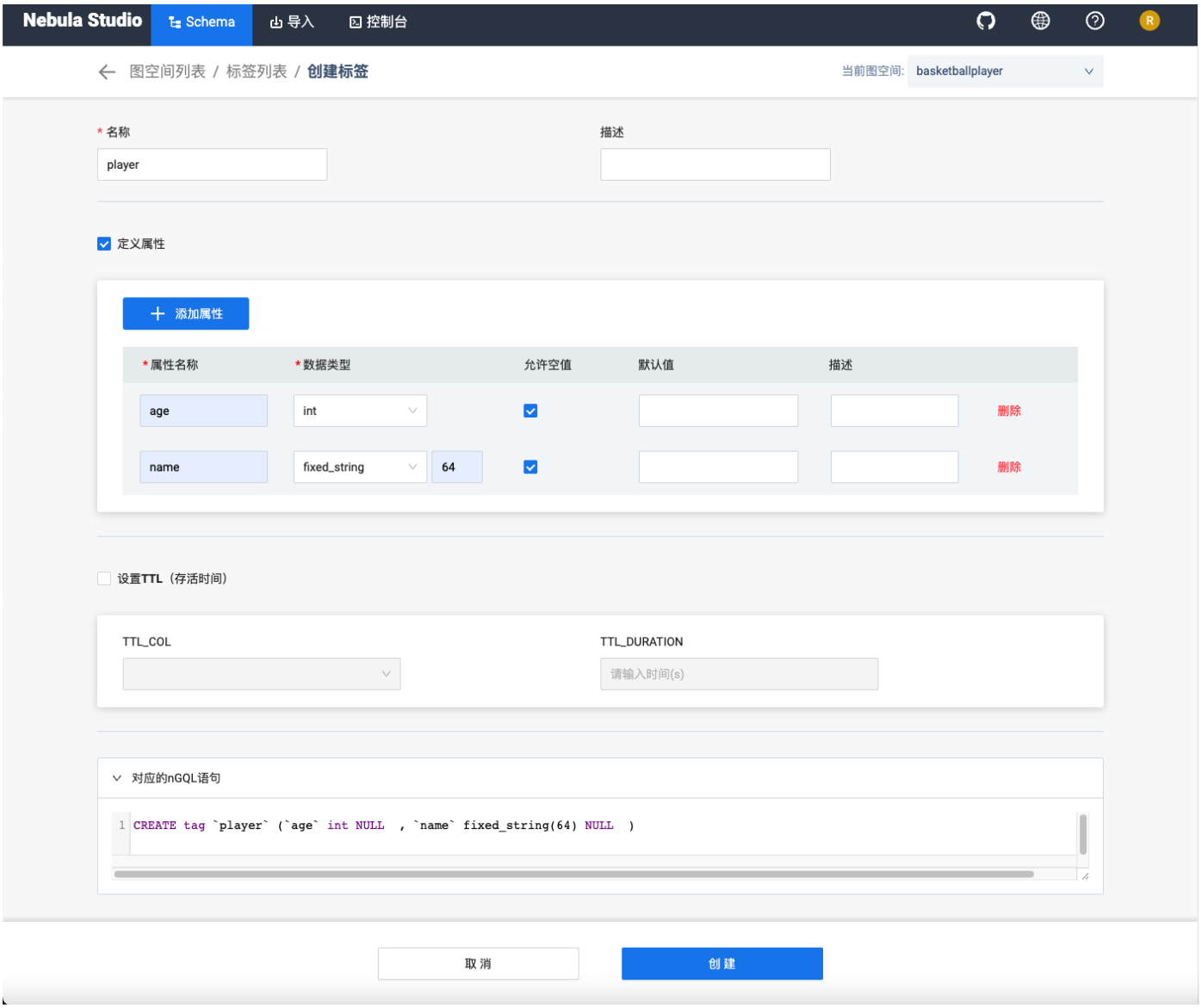
c. 定义属性（可选）：如果 Tag 需要属性，点击 **+** 添加属性，添加一个或多个属性：

- 输入属性名称。
- 选择数据类型。
- 选择是否允许空值。
- （可选）输入默认值。
- （可选）输入属性备注。

d. 设置**TTL**（存活时间）（可选）：Tag 未设置索引时，用户可以设置 TTL。勾选设置**TTL**（存活时间），设置 `TTL_COL` 和 `TTL_DURATION`（单位：秒）。详情参考 TTL 配置。

6. 完成设置后，在 对应的**nGQL**语句 面板，用户能看到与上述配置等价的 nGQL 语句。

2022 vesoft Inc.



7. 确认无误后，点击 创建 按钮。

   如果 Tag 创建成功，标签 面板会显示这个 Tag 的属性列表。

修改 TAG

1. 在顶部导航栏中，点击 **Schema** 页签。
2. 在 图空间列表 中，找到图空间，点击图空间名称或者在 操作 列中点击 **Schema**。
3. 在 当前图空间 里确认图空间名称。用户也可以通过选择图空间名称切换图空间。
4. 点击 标签 页签，找到需要修改的 Tag，并在 操作 列中，点击  图标。

2022 vesoft Inc.

5. 在 编辑 页面，用户可以选择以下操作：

- 如果要修改描述：在 描述 右侧点击 编辑，修改后点击 确认。
- 如果要修改属性：在 定义属性 区域，找到需要修改的属性，在右侧点击 编辑，修改后点击 确认。
- 如果要删除属性：在 定义属性 区域，找到需要删除的属性，在右侧点击 删除，然后点击 确认。
- 如果要添加属性：在 定义属性 区域，点击 + 添加属性，设置属性信息，点击 确认。详细说明参考创建 Tag 。
- 如果要修改 TTL 信息：在 设置 **TTL** 区域，点击 编辑，修改后点击 确认。详情参考 TTL 配置。
- 如果要删除已经配置的 TTL 信息：在 设置 **TTL** 区域，取消勾选 设置**TTL**（存活时间），然后点击 确定。
- 如果要配置 TTL 信息：在 设置 **TTL** 区域，勾选 设置**TTL**（存活时间），设置`TTL_COL`和`TTL_DURATION`（单位：秒），点击 确认。详情参考 TTL 配置。

> **Note**
>
> TTL 与索引的共存问题，详情参考 TTL。

删除 TAG

> **Danger**
>
> 删除 Tag 前先确认影响，已删除的数据如未备份无法恢复。

1. 在顶部导航栏中，点击 **Schema** 页签。
2. 在 图空间列表 中，找到图空间，点击图空间名称或者在 操作 列中点击 **Schema**。
3. 在 当前图空间 里确认图空间名称。用户也可以通过选择图空间名称切换图空间。
4. 点击 标签 页签，找到需要删除的 Tag，并在 操作 列中，点击 图标。
5. 在弹出的对话框中点击 确认。

后续操作

Tag 创建成功后，用户可以在 控制台 上逐条插入点数据，或者使用 导入 功能批量插入点数据。

最后更新: June 20, 2023

2022 vesoft Inc.

操作 **Edge type**

在 Nebula Graph 数据库中创建图空间后，用户可能需要创建 Edge type（边类型）。用户可以选择使用 控制台 或者 **Schema** 操作 Edge type。本文仅说明如何使用 **Schema** 操作 Edge type。

前提条件

在 Studio 上操作 Edge type 之前，用户必须确认以下信息:

- Studio 已经连接到 Nebula Graph 数据库。
- 图空间已经创建。
- 当前登录的账号拥有 GOD、ADMIN 或者 DBA 的权限。

创建边类型

1. 在顶部导航栏中，点击 **Schema** 页签。
2. 在 图空间列表 中，找到图空间，点击图空间名称或者在 操作 列中点击 **Schema**。
3. 在 当前图空间 里确认图空间名称。用户也可以通过选择图空间名称切换图空间。
4. 点击 边类型 页签，并点击 **+** 创建 按钮。
5. 在 创建边类型 页面上，完成以下设置:

a. 名称：输入 Edge type 名称。本示例中，输入 `serve`。

b. 描述（可选）：输入 Edge type 的备注。

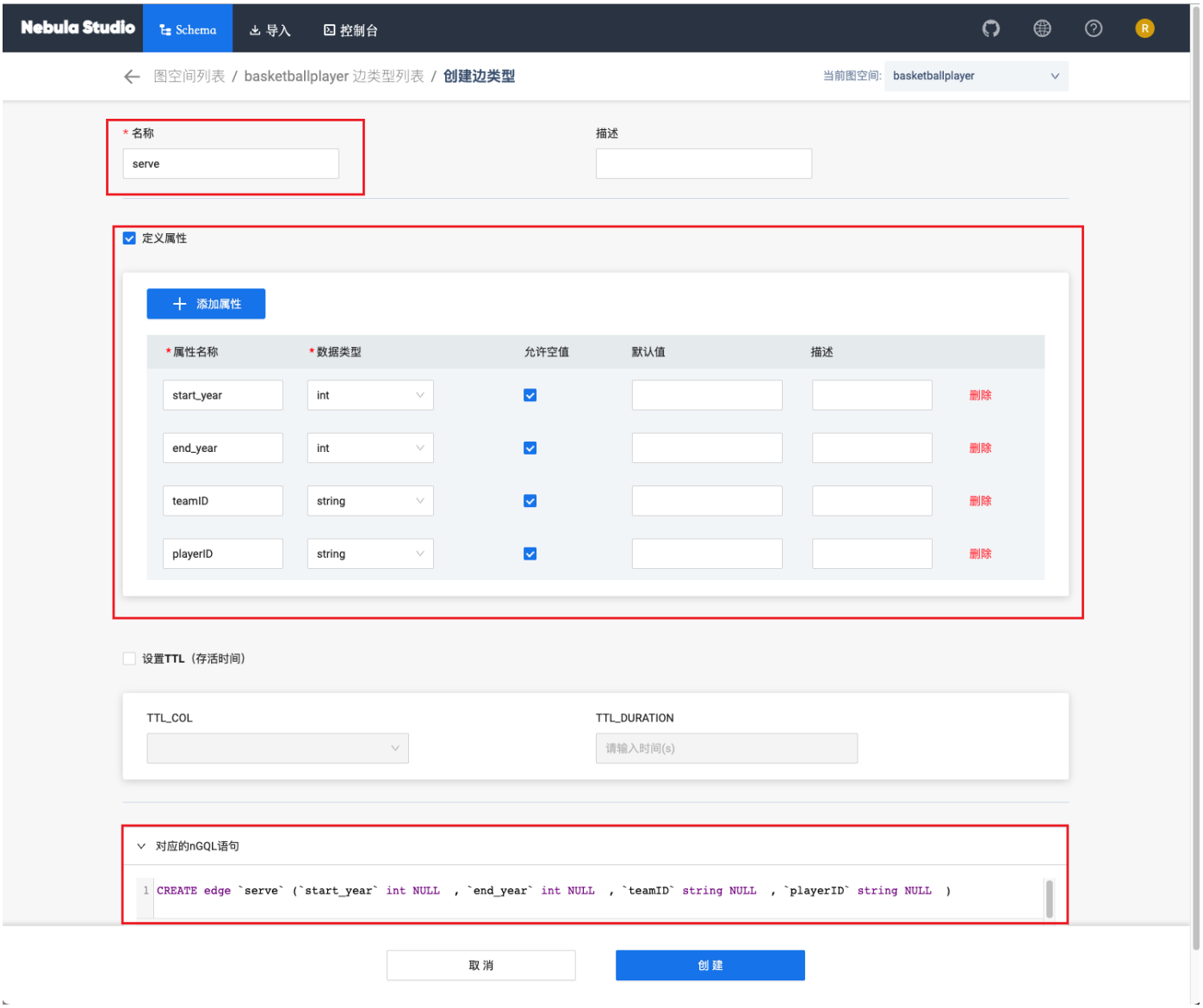
c. 定义属性（可选）：如果 Edge type 需要属性，点击 **+** 添加属性，添加一个或多个属性:

- 输入属性名称。
- 选择数据类型。
- 选择是否允许空值。
- （可选）输入默认值。
- （可选）输入属性备注。

d. 设置**TTL**（存活时间）（可选）：Edge type 未设置索引时，用户可以设置 TTL。勾选设置**TTL**（存活时间），设置 `TTL_COL` 和 `TTL_DURATION`（单位: 秒）。详情参考 TTL 配置。

6. 完成设置后，在 对应的 **nGQL** 语句 面板上，用户能看到与上述配置等价的 nGQL 语句。

2022 vesoft Inc.



7. 确认无误后，点击 创建 按钮。

   如果 Edge Type 创建成功，边类型 面板会显示这个 Edge type 的属性列表。

2022 vesoft Inc.

修改 EDGE TYPE

按以下步骤使用 **Schema** 修改 Edge type：

1. 在顶部导航栏中，点击 **Schema** 页签。
2. 在 图空间列表 中，找到图空间，点击图空间名称或者在 操作 列中点击 **Schema**。
3. 在 当前图空间 里确认图空间名称。用户也可以通过选择图空间名称完成图空间切换。
4. 点击 边类型 页签，找到需要修改的 Edge Type，并在 操作 列中，点击  图标。
5. 在 编辑 页面，用户可以选择以下操作：

- 如果要修改描述：在 描述 右侧点击 编辑，修改后点击 确认。
- 如果要修改属性：在 定义属性 区域，找到需要修改的属性，在右侧点击 编辑，修改后点击 确认。
- 如果要删除属性：在 定义属性 区域，找到需要删除的属性，在右侧点击 删除，然后点击 确认。
- 如果要添加属性：在 定义属性 区域，点击 + 添加属性，设置属性信息，点击 确认。详细说明参考创建边类型 。
- 如果要修改 TTL 信息：在 设置 **TTL** 区域，点击 编辑，修改后点击 确认。详情参考 TTL 配置。
- 如果要删除已经配置的 TTL 信息：在 设置 **TTL** 区域，取消勾选 设置**TTL**（存活时间），然后点击 确定。
- 如果要配置 TTL 信息：在 设置 **TTL** 区域，勾选 设置**TTL**（存活时间），设置 `TTL_COL` 和 `TTL_DURATION` （单位：秒），点击 确认。详情参考 TTL 配置。

> **Note**
>
> TTL 与索引的共存问题，详情请见 TTL。

删除 EDGE TYPE

> **Danger**
>
> 删除 Edge type 前先确认影响，已删除的数据如未备份无法恢复。

1. 在顶部导航栏中，点击 **Schema** 页签。
2. 在 图空间列表 中，找到图空间，点击图空间名称或者在 操作 列中点击 **Schema**。
3. 在 当前图空间 里确认图空间名称。用户也可以通过选择图空间名称切换图空间。
4. 点击 边类型 页签，找到需要修改的 Edge type，并在 操作 列中，点击  图标。
5. 在弹出的对话框中点击 确认。

后续操作

Edge type 创建成功后，用户可以在 控制台 上逐条插入边数据，或者使用 导入 功能批量插入边数据。

最后更新: June 20, 2023

2022 vesoft Inc.

操作索引

用户可以为 Tag 和 Edge type 创建索引，使得图查询时可以从拥有共同属性的同一类型的点或边开始遍历，使大型图的查询更为高效。用户可以选择使用 控制台 或者 **Schema** 操作索引。本文仅说明如何使用 **Schema** 操作索引。

> **Note**
>
> 一般在创建了 Tag 或者 Edge type 之后即可创建索引，但是，索引会影响写性能，所以，建议先导入数据，再批量重建索引。关于索引的详细信息，参考 索引介绍。

前提条件

在 Studio 上操作索引之前，用户必须确认以下信息：

- Studio 已经连接到 Nebula Graph 数据库。
- 图空间、Tag 和 Edge type 已经创建。
- 当前登录的账号拥有 GOD、ADMIN 或者 DBA 的权限。

创建索引

1. 在顶部导航栏中，点击 **Schema** 页签。
2. 在 图空间列表 中，找到图空间，点击图空间名称或者在 操作 列中点击 **Schema**。
3. 在 当前图空间 里确认图空间名称。用户也可以通过选择图空间名称切换图空间。
4. 点击 索引 页签，并点击 **+** 创建 按钮。
5. 在 创建索引 页面上，完成以下设置：

a. 索引类型：选择索引类型，即 **Tag** 或者 **Edge type**。本示例中选择 **Edge type**。

b. 关联名称：选择要关联的 Tag 或 Edge type 名称。本示例中选择 **follow**。

c. 索引名称：输入索引名称。本示例中，输入 **follow_index**。

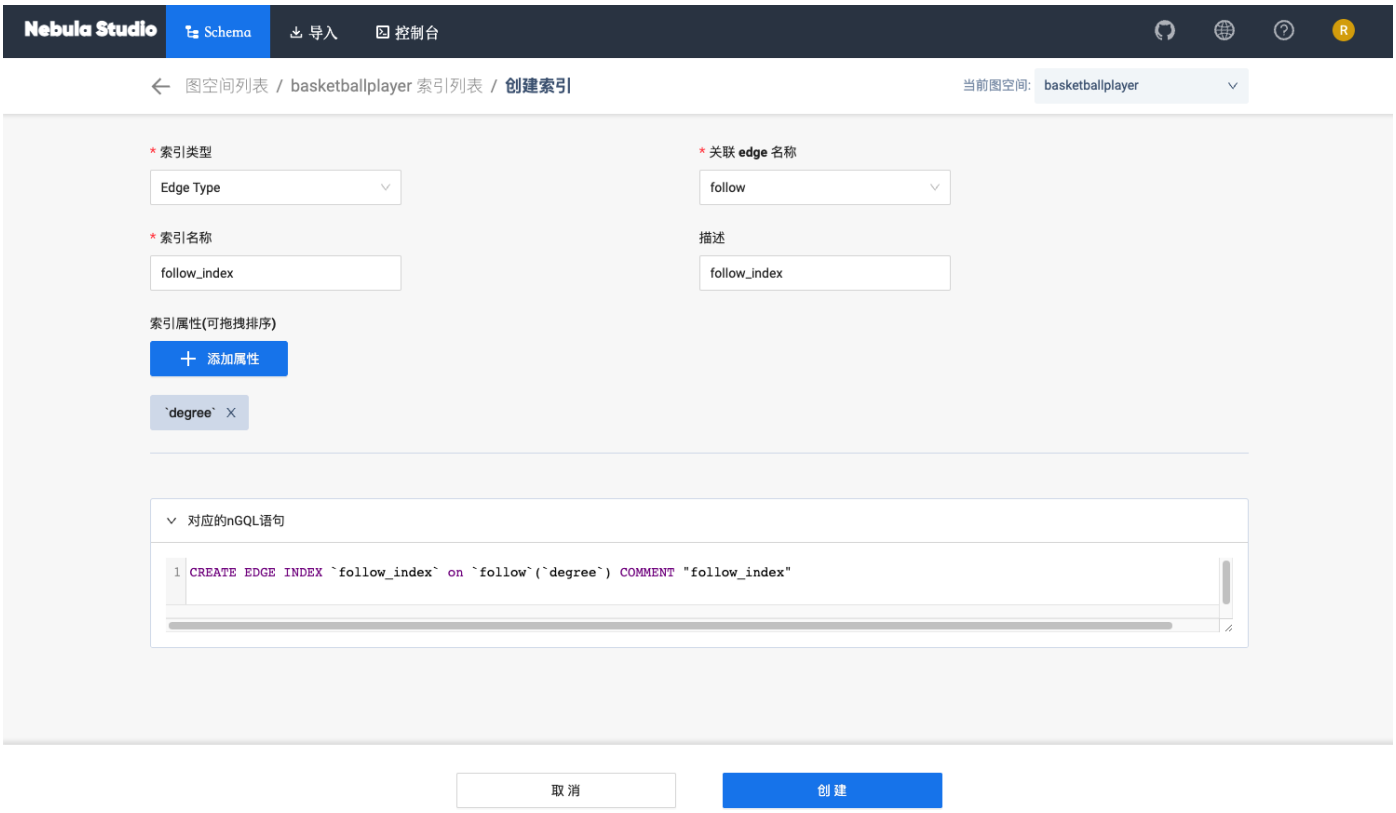
d. 描述（可选）：输入索引的备注。

e. 索引属性（可选）：点击 **+** 添加属性，在弹出的对话框列表里选择需要关联的属性，点击 确认。如果需要关联多个属性，重复这一步骤。用户可以按界面提示重排索引属性的顺序。本示例中选择 `degree`。

> **Note**
>
> 索引属性的顺序会影响 `LOOKUP` 语句的查询结果。详细信息，参考 LOOKUP。

6. 完成设置后，在 对应的 **nGQL** 面板，用户能看到与上述配置等价的 nGQL 语句。

2022 vesoft Inc.



7. 确认无误后，点击 创建 按钮。

   如果索引创建成功，索引 面板会显示这个索引的属性列表。

查看索引

1. 在顶部导航栏中，点击 **Schema** 页签。
2. 在 图空间列表 中，找到图空间，点击图空间名称或者在 操作 列中点击 **Schema**。
3. 在 当前图空间 里确认图空间名称。用户也可以通过选择图空间名称切换图空间。
4. 点击 索引 页签，在列表左上方，选择需要查看的索引类型。
5. 在列表中，找到需要查看的索引，点击索引所在行即可展开查看该索引相关的所有属性名称、数据类型。

重建索引

1. 在顶部导航栏中，点击 **Schema** 页签。
2. 在 图空间列表 中，找到图空间，点击图空间名称或者在 操作 列中点击 **Schema**。
3. 在 当前图空间 里确认图空间名称。用户也可以通过选择图空间名称切换图空间。
4. 点击 索引 页签，在列表左上方，选择索引类型。
5. 找到需要重建的索引，在 操作列 点击 重建索引。

> **Note**
>
> 关于重建索引，详情参见 REBUILD INDEX。

删除索引

1. 在顶部导航栏中，点击 **Schema** 页签。
2. 在 图空间列表 中，找到图空间，点击图空间名称或者在 操作 列中点击 **Schema**。

2022 vesoft Inc.

3. 在 当前图空间 里确认图空间名称。用户也可以通过选择图空间名称切换图空间。
4. 点击 索引 页签，在列表左上方，选择索引类型。
5. 找到需要删除的索引，并在 操作 列中，点击  图标。
6. 在弹出的对话框中点击 确认。

最后更新: June 20, 2023

2022 vesoft Inc.

14.5 故障排查

14.5.1 连接数据库错误

问题描述

按连接 Studio 文档操作，提示 配置失败。

可能的原因及解决方法

用户可以按以下步骤排查问题。

第 1 步。确认 HOST 字段的格式是否正确

必须填写 Nebula Graph 图数据库 Graph 服务的 IP 地址（`graph_server_ip`）和端口。如果未做修改，端口默认为 `9669`。即使 Nebula Graph 与 Studio 都部署在当前机器上，用户也必须使用本机 IP 地址，而不能使用 `127.0.0.1`、`localhost` 或者 `0.0.0.0`。

第 2 步。确认 用户名 和 密码 是否正确

如果 Nebula Graph 没有开启身份认证，用户可以填写任意字符串登录。

如果已经开启身份认证，用户必须使用分配的账号登录。

第 3 步。确认 NEBULA GRAPH 服务是否正常

检查 Nebula Graph 服务状态。关于查看服务的操作：

- 如果在 Linux 服务器上通过编译部署的 Nebula Graph，参考查看 Nebula Graph 服务。
- 如果使用 Docker Compose 部署和 RPM 部署的 Nebula Graph，参考查看 Nebula Graph 服务状态和端口。

如果 Nebula Graph 服务正常，进入第 4 步继续排查问题。否则，请重启 Nebula Graph 服务。

> **Note**
>
> 如果之前使用 `docker-compose up -d` 启动 Nebula Graph，必须运行 `docker-compose down` 命令停止 Nebula Graph。

第 4 步。确认 GRAPH 服务的网络连接是否正常

在 Studio 机器上运行命令（例如 `telnet <graph_server_ip> 9669`）确认 Nebula Graph 的 Graph 服务网络连接是否正常。

如果连接失败，则按以下要求检查：

- 如果 Studio 与 Nebula Graph 在同一台机器上，检查端口是否已暴露。
- 如果两者不在同一台机器上，检查 Nebula Graph 服务器的网络配置，例如，防火墙、网关以及端口。

如果按上述步骤排查后仍无法连接 Nebula Graph 服务，请前往 Nebula Graph 官方论坛咨询。

最后更新: June 20, 2023

2022 vesoft Inc.

14.5.2 无法访问 Studio

问题描述

我按照文档描述启动 Studio 后访问 `127.0.0.1:7001` 或者 `0.0.0.0:7001`，但是打不开页面，为什么?

可能的原因及解决方法

用户可以按以下顺序排查问题。

1.确认系统架构

需要确认部署 Studio 服务的机器是否为 x86_64 架构。目前 Studio 仅支持 x86_64 系统架构。

2.检查 **STUDIO** 服务是否正常启动

- 使用 RPM 或 DEB 包部署的 Studio，使用 `systemctl status nebula-graph-studio` 查看运行状态。
- 使用 tar 包部署的 Studio，使用 `sudo lsof -i:7001` 查看端口状态。
- 使用 Docker-compose 部署的 Studio，使用 `docker-compose ps` 查看运行状态。

  如果服务正常，返回结果如下。其中，`State` 列应全部显示为 `Up`。

```
     Name                          Command               State              Ports
-------------------------------------------------------------------------------------------------
nebula-web-docker_client_1     ./nebula-go-api                  Up      0.0.0.0:32782->8080/tcp
nebula-web-docker_importer_1   nebula-importer --port=569 ...   Up      0.0.0.0:32783->5699/tcp
nebula-web-docker_nginx_1      /docker-entrypoint.sh ngin ...   Up      0.0.0.0:7001->7001/tcp, 80/tcp
nebula-web-docker_web_1        docker-entrypoint.sh npm r ...   Up      0.0.0.0:32784->7001/tcp
```

Note

如果之前使用 `docker-compose up -d` 启动 Studio，必须运行 `docker-compose down` 命令停止 Studio。

如果服务没有正常运行，请重新启动 Studio。详细信息，参考部署 Studio。

3.确认访问地址

如果 Studio 与浏览器在同一台机器上，用户可以在浏览器里使用 `localhost:7001`、`127.0.0.1:7001` 或者 `0.0.0.0:7001` 访问 Studio。

如果两者不在同一台机器上，必须在浏览器里输入 `<studio_server_ip>:7001`。其中，`studio_server_ip` 是指部署 Studio 服务的机器的 IP 地址。

4.确认网络连通性

运行 `curl <studio_server_ip>:7001 -I` 确认是否正常。如果返回 `HTTP/1.1 200 OK`，表示网络连通正常。

如果连接被拒绝，则按以下要求检查:

- 如果浏览器与 Studio 在同一台机器上，检查端口是否已暴露。
- 如果两者不在同一台机器上，检查 Studio 所在机器的网络配置，例如，防火墙、网关以及端口。

如果按上述步骤排查后仍无法访问 Studio，请前往 Nebula Graph 官方论坛咨询。

最后更新: June 20, 2023

2022 vesoft Inc.

14.5.3 常见问题

为什么我无法使用某个功能?

如果发现无法使用某个功能，建议按以下步骤排除问题:

1. 确认 Nebula Graph 是最新版本。如果使用 Docker Compose 部署 Nebula Graph 数据库，建议运行 `docker-compose pull && docker-compose up -d` 拉取最新的 Docker 镜像，并启动容器。
2. 确认 Studio 是最新版本。详细信息参考版本更新。
3. 搜索论坛或 GitHub 的 nebula 和 nebula-web-docker 项目，确认是否已经有类似的问题。
4. 如果上述操作均未解决问题，欢迎在论坛上提交问题。

最后更新: June 20, 2023

2022 vesoft Inc.

15. Nebula Dashboard（社区版）

15.1 什么是 Nebula Dashboard（社区版）

Nebula Dashboard（简称 Dashboard）是一款用于监控 Nebula Graph 集群中机器和服务状态的可视化工具。本文主要介绍社区版 Dashboard，企业版详情参见什么是 Nebula Dashboard（企业版）。

Enterpriseonly

企业版增加了可视化创建集群、批量导入集群、快速扩缩容等功能，点击定价查看更多。

15.1.1 产品功能

- 监控集群中所有机器的状态，包括 CPU、内存、负载、磁盘和流量。
- 监控集群中所有服务的信息，包括服务 IP 地址、版本和监控指标（例如查询数量、查询延迟、心跳延迟等）。
- 监控集群本身的信息，包括集群的服务信息、分区信息、配置和长时任务。

15.1.2 适用场景

如果有以下任一需求，都可以使用 Dashboard：

- 需要方便快捷地监测关键指标，集中呈现业务的多个重点信息，保证业务正常运行。
- 需要多维度（例如时间段、聚合规则、指标）监控集群。
- 故障发生后，需要复盘问题，确认故障发生时间、异常现象。

15.1.3 注意事项

- 监控数据默认更新频率约为 7 秒。
- 监控数据默认保留 14 天，即只能查询最近 14 天内的监控数据。

Note

监控服务由 prometheus 提供，更新频率和保留时间等都可以自行修改。详情请参见 prometheus 官方文档。

15.1.4 版本兼容性

Nebula Graph 的版本和 Dashboard 社区版的版本对应关系如下。

Nebula Graph 版本	**Dashboard** 版本
2.5.0 ~ 3.1.0	3.1.0
2.5.x ~ 3.1.0	1.1.1
2.0.1 ~ 2.5.1	1.0.2
2.0.1 ~ 2.5.1	1.0.1

2022 vesoft Inc.

15.1.5 更新说明

Release

最后更新: June 20, 2023

2022 vesoft Inc.

15.2 部署 Dashboard

安装部署 Dashboard 涉及 5 种服务，本文将详细介绍如何通过 tar 包安装部署。下载和编译 Nebula Dashboard 的最新源码，请参见 GitHub nebula dashboard 页面的说明。

15.2.1 前提条件

在部署 Dashboard 之前，用户需要确认以下信息：

- Nebula Graph 服务已经部署并启动。详细信息参考 Nebula Graph 安装部署。
- 确保以下端口未被使用：
- 9200
- 9100
- 9090
- 8090
- 7003
- 使用的 Linux 发行版为 CentOS，安装有版本为 v10.12.0 以上的 Node.js，安装有版本为 1.13 及以上的 Go。

15.2.2 下载 Dashboard

根据需要下载 tar 包，建议选择最新版本。

Dashboard 安装包	**Nebula Graph** 版本
nebula-dashboard-1.1.2.x86_64.tar.gz	2.5.x ~ 3.1.x

15.2.3 目录结构说明

执行命令 `tar -xvf nebula-dashboard-1.1.2.x86_64.tar.gz` 解压缩，目录 `nebula-dashboard/vendors` 内一共有 4 个子目录，说明如下。

目录名称	说明	端口号
node-exporter	收集集群中机器的资源信息，包括 CPU、内存、负载、磁盘和流量。	9100
nebula-stats-exporter	收集集群的性能指标，包括服务 IP 地址、版本和监控指标（例如查询数量、查询延迟、心跳延迟等）。	9200
prometheus	存储监控数据的时间序列数据库。	9090
nebula-http-gateway	为集群服务提供 HTTP 接口，执行 nGQL 语句与 Nebula Graph 数据库进行交互。	8090

4 个目录需要按照部署需要，复制到各个机器上，详情请参见下文。

15.2.4 操作步骤

部署 **`node-exporter`** 服务

> **Note**
>
> 集群中的每个机器都需要部署 `node-exporter` 服务。

2022 vesoft Inc.

在目录 `node-exporter` 内执行如下命令启动服务：

```
$ nohup ./node-exporter --web.listen-address=":9100" &
```

服务启动后，可以在浏览器中输入 `<IP>:9100` 检查服务是否正常启动。

部署 **nebula-stats-exporter** 服务

> **Note**
>
> 只需要在 `nebula-dashboard` 服务所在机器部署 `nebula-stats-exporter` 服务。

1. 在目录 `nebula-stats-exporter` 内修改文件 `config.yaml`，配置所有服务的 HTTP 端口，示例如下：

```
clusters:
  - name: nebula
    instances:
      - name: metad0
        endpointIP: 192.168.8.157
        endpointPort: 19559
        componentType: metad
      - name: metad1
        endpointIP: 192.168.8.155
        endpointPort: 19559
        componentType: metad
      - name: metad2
        endpointIP: 192.168.8.154
        endpointPort: 19559
        componentType: metad
      - name: graphd0
        endpointIP: 192.168.8.157
        endpointPort: 19669
        componentType: graphd
      - name: graphd1
        endpointIP: 192.168.8.155
        endpointPort: 19669
        componentType: graphd
      - name: graphd2
        endpointIP: 192.168.8.154
        endpointPort: 19669
        componentType: graphd
      - name: storaged0
        endpointIP: 192.168.8.157
        endpointPort: 19779
        componentType: storaged
      - name: storaged1
        endpointIP: 192.168.8.155
        endpointPort: 19779
        componentType: storaged
      - name: storaged2
        endpointIP: 192.168.8.154
        endpointPort: 19779
        componentType: storaged
```

2. 执行如下命令启动服务：

```
$ nohup  ./nebula-stats-exporter --listen-address=":9200" --bare-metal --bare-metal-config=./config.yaml &
```

服务启动后，可以在浏览器中输入 `<IP>:9200` 检查服务是否正常启动。

2022 vesoft Inc.

部署 **prometheus** 服务

> **Note**
>
> 只需要在 nebula-dashboard 服务所在机器部署 prometheus 服务。

1. 在目录 prometheus 内修改文件 prometheus.yaml，配置 node-exporter 服务和 nebula-stats-exporter 服务的 IP 地址和端口，示例如下：

```
global:
  scrape_interval:     5s
  evaluation_interval: 5s
scrape_configs:
  - job_name: 'nebula-stats-exporter'
    static_configs:
      - targets: [
          '192.168.xx.100:9200'  #  nebula-stats-exporter 服务的 IP 地址和端口。
        ]
  - job_name: 'node-exporter'
    static_configs:
      - targets: [
          '192.168.xx.100:9100',  #  node-exporter 服务的 IP 地址和端口。
          '192.168.xx.101:9100'
        ]
```

- scrape_interval：收集监控数据的间隔时间。默认为 1 分钟。
- evaluation_interval：告警规则扫描时间间隔。默认为 1 分钟。

2. 执行如下命令启动服务：

```
$ nohup ./prometheus --config.file=./prometheus.yaml &
```

服务启动后，可以在浏览器中输入 <IP>:9090 检查服务是否正常启动。

部署 **nebula-http-gateway** 服务

> **Note**
>
> 只需要在 nebula-dashboard 服务所在机器部署 nebula-http-gateway 服务。

在目录 nebula-http-gateway 内执行如下命令启动服务：

```
$ nohup ./nebula-httpd &
```

服务启动后，可以在浏览器中输入 <IP>:8090 检查服务是否正常启动。

部署 **dashboard** 服务

1. 在目录 nebula-dashboard 内修改文件 config.json，配置 Graph 服务的 IP 地址，端口和配置代理服域名，示例如下：

```
  port: 7003
  proxy:
    gateway:
      target: "127.0.0.1:8090"  //  nebula-http-gateway 服务的 IP 和端口号。
    prometheus:
      target: "127.0.0.1:9090"  // Prometheus 服务的 IP 和端口号。
    nebulaServer:
      ip: "192.168.8.143"  // 任一 Graph 服务的机器 IP。
      port: 9669 // Graph 服务的端口号。
  ...
```

2. 在目录 nebula-dashboard 内执行如下命令启动服务：

```
$ nohup ./dashboard &
```

服务启动后，可以在浏览器中输入 <IP>:7003 检查服务是否正常启动。

2022 vesoft Inc.

15.2.5 停止 Dashboard

如果需要停止 Dashboard，可以使用 `kill <pid>` 的方式停止，示例如下：

```
$ kill $(lsof -t -i :9100) # 停止 node-exporter 服务
$ kill $(lsof -t -i :9200) # 停止 nebula-stats-exporter 服务
$ kill $(lsof -t -i :9090) # 停止 prometheus 服务
$ kill $(lsof -t -i :8090) # 停止 nebula-http-gateway 服务
$ kill $(lsof -t -i :7003) # 停止 dashboard 服务
```

最后更新: June 20, 2023

2022 vesoft Inc.

15.3 连接 Dashboard

Dashboard 部署完成后，可以通过浏览器登录使用 Dashboard。

15.3.1 前提条件

- Dashboard 相关服务已经启动。详情请参见部署 Dashboard。
- 建议使用 Chrome 58 及以上的版本的 Chrome 浏览器，否则可能有兼容问题。

15.3.2 操作步骤

1. 确认 `nebula-dashboard` 服务所在机器的 IP 地址，在浏览器中输入 `<IP>:7003` 打开登录页面。
2. 输入 Nebula Graph 数据库的账号和密码。

   Note

   输入 Nebula Graph 数据库的账号和密码前，确保在 `config.json` 文件中配置了部署 Graph 服务的机器 IP。更多信息，参见部署 Dashboard。

   - 如果 Nebula Graph 已经启用身份验证，用户可以使用已创建的账号连接 Dashboard。
   - 如果 Nebula Graph 未启用身份验证，用户只能使用默认用户 `root` 和任意密码连接 Dashboard。

   有关如何启用身份验证请参见身份验证。
3. 选择 Nebula Graph 的版本。

   Note

   请确保选择的版本与 `config.json` 文件中配置的 Nebula Graph 版本一致,否则会有兼容性问题。更多信息，参见部署 Dashboard。

2022 vesoft Inc.



4. 单击登录。

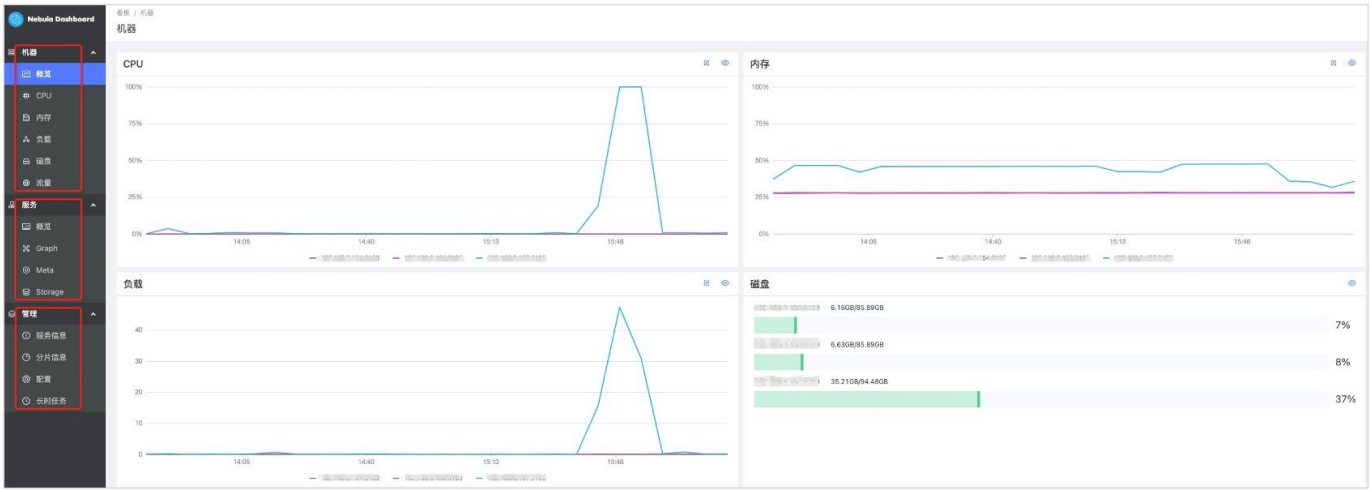
最后更新: June 20, 2023

2022 vesoft Inc.

15.4 Dashboard 页面介绍

Dashboard 页面主要分为机器、服务、管理三个部分，本文将详细介绍这些界面。

15.4.1 页面概览



2022 vesoft Inc.

15.4.2 机器页面介绍

机器页面分为如下几个子页面：

- 概览

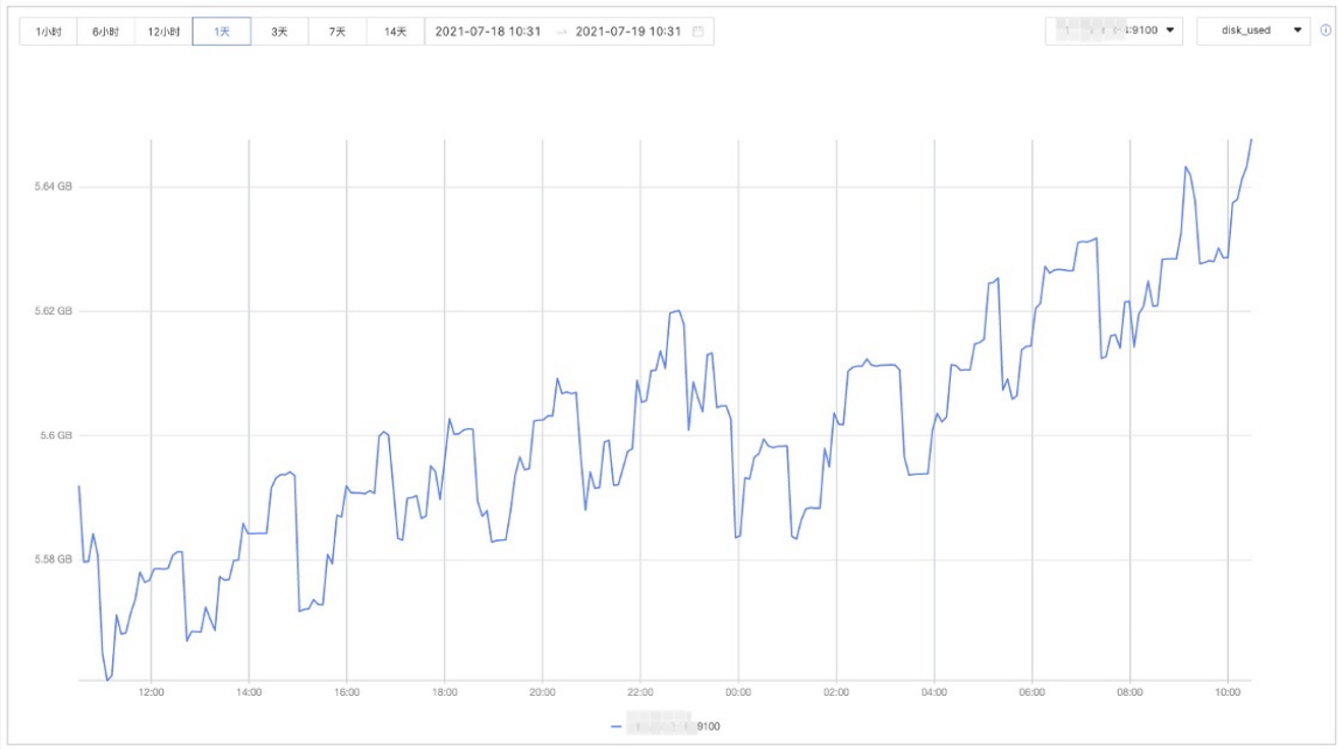
  可以查看最近一天的 CPU、内存、负载、磁盘和上下行流量变化情况。

  如果需要查看某一项更详细的监控指标，可以单击模块右上角的  按钮，或者在左侧单击监控项。

- CPU、内存、负载、磁盘、流量

  分类展示机器各个维度的详细监控数据。

- 默认可以选择最多 14 天的监控数据进行查看，也可以快捷选择 1 小时、6 小时、12 小时、1 天、3 天、7 天和 14 天。
- 可以选择需要查看的机器和监控指标。监控指标详情请参见监控指标说明。
- 可以设置基线，作为参考标准线。



2022 vesoft Inc.

15.4.3 服务页面介绍

服务页面分为如下几个子页面：

- 概览

  可以查看最近一天各种服务的监控指标变化情况，右上角还可以切换到版本页面查看所有服务的 IP 地址和版本。

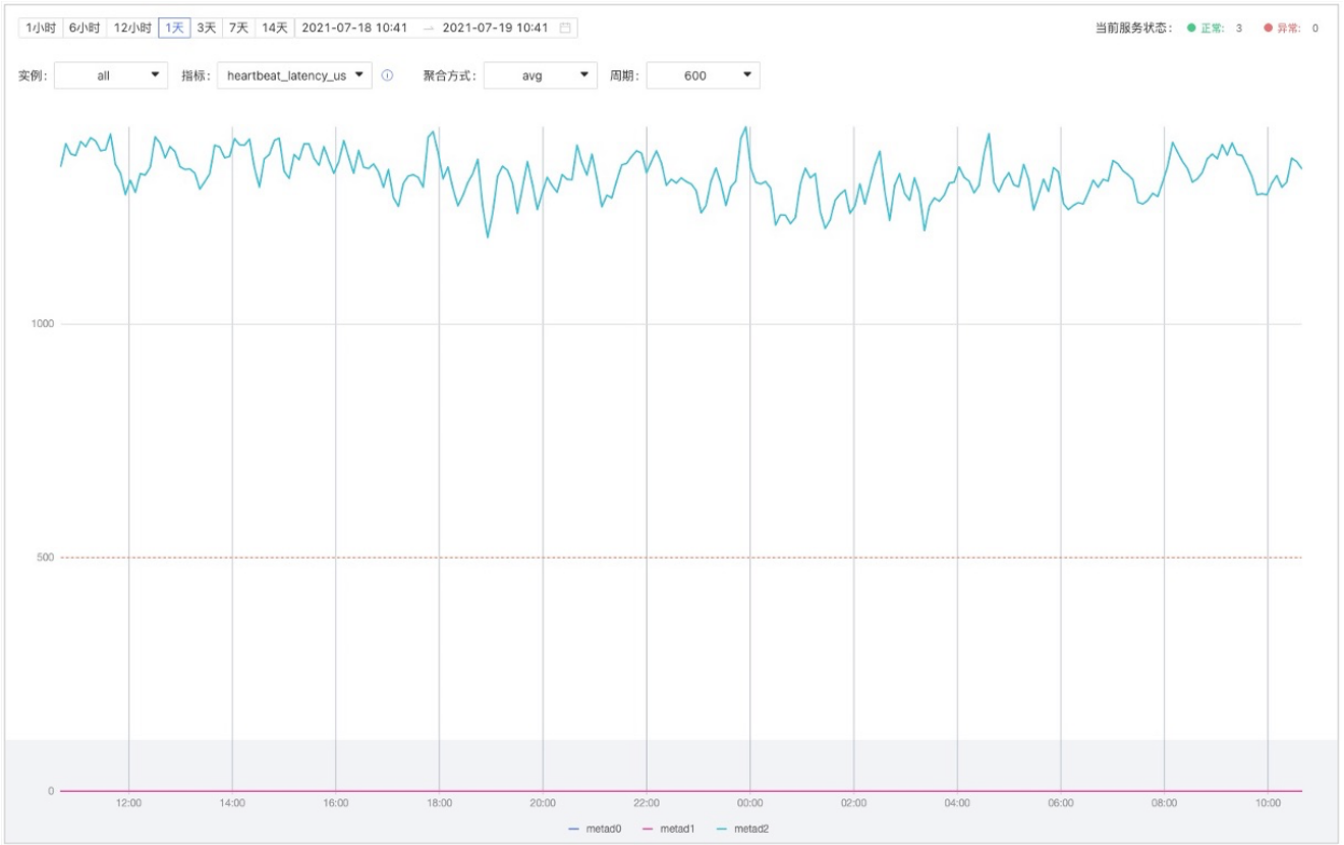
  如果需要查看某一种服务更详细的监控指标，可以单击模块右上角的  按钮，或者在左侧单击具体服务。

  > **Note**
  >
  > 当前社区版的概览页仅支持每种服务设置两个监控指标，可以单击模块内的设置按钮进行调整。

- Graph、Meta、Storage

  分类展示各个服务的详细监控数据。

- 默认可以选择最多 14 天的监控数据进行查看，也可以快捷选择 1 小时、6 小时、12 小时、1 天、3 天、7 天和 14 天。
- 可以选择需要查看监控数据的机器、监控指标、聚合方式和周期。监控指标详情请参见监控指标说明。
- Graph 服务支持一系列基于图空间的监控指标。详情，参见图空间监控。
- 可以设置基线，作为参考标准线。
- 可以查看当前服务的状态。



2022 vesoft Inc.

15.4.4 管理页面介绍

Note

非 `root` 用户只能查看服务信息和有空间权限的分片信息，无法查看配置和长时任务。

管理页面分为如下几个子页面：

- 服务信息

  展示 Storage 服务的基本信息，包括主机信息、版本 Commit ID、leader 总数、分片分布和 leader 分布。

- 分片信息

  可以选择不同图空间，查看分片信息。参数说明如下。

参数	说明
`Partition ID`	分片序号。
`Leader`	分片的 leader 副本的 IP 地址和端口。
`Peers`	分片所有副本的 IP 地址和端口。
`Losts`	分片的故障副本的 IP 地址和端口。

- 配置

  展示 Graph 和 Storage 服务的配置信息。暂不支持在线修改配置，详情请参见配置管理。

- 长时任务

  展示所有作业的信息。暂不支持在线管理作业，详情请参见作业管理。

15.4.5 其他

在页面左下角，还可以进行如下操作：

- 退出登录
- 切换中英文
- 查看当前 Dashboard 版本
- 查看帮助信息
- 折叠侧边栏

最后更新: June 20, 2023

2022 vesoft Inc.

15.5 监控指标说明

本文介绍 Dashboard 中展示的 Nebula Graph 监控指标。

15.5.1 机器

Note

- 以下机器指标适用于 Linux 操作系统。
- 磁盘容量和流量的默认单位为字节（Byte），页面显示时单位会随着数据量级而变化，例如流量低于 1 KB/s 时单位为 Bytes/s。
- 对于高于 v1.0.2 版本的社区版 Dashboard，机器的 Buff 和 Cache 所占的内存没有被计算在内存使用率中。

CPU

参数	说明
`cpu_utilization`	CPU 已使用百分比
`cpu_idle`	CPU 空闲百分比
`cpu_wait`	等待 IO 操作的 CPU 百分比
`cpu_user`	用户空间（非 Nebula Graph 图空间）占用的 CPU 百分比
`cpu_system`	内核空间（非 Nebula Graph 内核空间）占用的 CPU 百分比

内存

参数	说明
`memory_utilization`	内存已使用百分比
`memory_used`	已使用内存（不包括缓存）。
`memory_free`	空闲内存

负载

参数	说明
`load_1m`	最近 1 分钟系统平均负载
`load_5m`	最近 5 分钟系统平均负载
`load_15m`	最近 15 分钟系统平均负载

2022 vesoft Inc.

磁盘

参数	说明
`disk_used`	磁盘已使用存储空间
`disk_free`	磁盘剩余存储空间
`disk_readbytes`	磁盘每秒读取的字节数
`disk_writebytes`	磁盘每秒写入的字节数
`disk_readiops`	磁盘每秒的读请求数量
`disk_writeiops`	磁盘每秒的写请求数量
`inode_utilization`	inode 已使用百分比

流量

参数	说明
`network_in_rate`	网卡每秒接收的字节数
`network_out_rate`	网卡每秒发送的字节数
`network_in_errs`	网卡每秒接收错误的字节数
`network_out_errs`	网卡每秒发送错误的字节数
`network_in_packets`	网卡每秒接收的数据包数量
`network_out_packets`	网卡每秒发送的数据包数量

15.5.2 服务

周期

指标统计的时间范围，当前支持 5 秒、60 秒、600 秒和 3600 秒，分别表示最近 5 秒、最近 1 分钟、最近 10 分钟和最近 1 小时。

聚合方式

参数	说明
`rate`	周期内平均每秒操作的速率
`sum`	周期内操作的总和
`avg`	周期内响应平均耗时
`P75`	周期内响应耗时从小到大排列，顺序处于 75%位置的分位数
`P95`	周期内响应耗时从小到大排列，顺序处于 95%位置的分位数
`P99`	周期内响应耗时从小到大排列，顺序处于 99%位置的分位数
`P999`	周期内响应耗时从小到大排列，顺序处于 99.9%位置的分位数

Note

以下为 Dashboard 获取内核的全量指标，但 Dashboard 仅展示重要的指标。

2022 vesoft Inc.

Graph

参数	说明
num_active_queries	当前正在执行的查询数。
num_active_sessions	当前活跃的会话数量。
num_aggregate_executors	聚合（Aggregate）算子的执行次数。
num_auth_failed_sessions_bad_username_password	因用户名密码错误导验证失败的会话数量。
num_auth_failed_sessions_out_of_max_allowed	因为超过 FLAG_OUT_OF_MAX_ALLOWED_CONNECTIONS 参数导致的验证登录的失败的 session 数量。
num_auth_failed_sessions	登录验证失败的会话数量。
num_indexscan_executors	索引扫描（IndexScan）算子的执行次数。
num_killed_queries	被终止的查询数量。
num_opened_sessions	服务端建立过的会话数量。
num_queries	查询次数。
num_query_errors_leader_changes	因查询错误而导致的 Leader 变更的次数。
num_query_errors	查询错误次数。
num_reclaimed_expired_sessions	服务端主动回收的过期的会话数量。
num_rpc_sent_to_metad_failed	Graphd 服务发给 Metad 的 RPC 请求失败的数量。
num_rpc_sent_to_metad	Graphd 服务发给 Metad 服务的 RPC 请求数量。
num_rpc_sent_to_storaged_failed	Graphd 服务发给 Storaged 服务的 RPC 请求失败的数量。
num_rpc_sent_to_storaged	Graphd 服务发给 Storaged 服务的 RPC 请求数量。
num_sentences	Graphd 服务接收的语句数。
num_slow_queries	慢查询次数。
num_sort_executors	排序（Sort）算子的执行次数。
optimizer_latency_us	优化器阶段延迟时间。
query_latency_us	查询平均延迟时间。
slow_query_latency_us	慢查询平均延迟时间。
num_queries_hit_memory_watermark	达到内存水位线的语句的数量。

2022 vesoft Inc.

Meta

参数	说明
commit_log_latency_us	Raft 协议中 Commit 日志的延迟时间。
commit_snapshot_latency_us	Raft 协议中 Commit 快照的延迟时间。
heartbeat_latency_us	心跳延迟时间。
num_heartbeats	心跳次数。
num_raft_votes	Raft 协议中投票的次数。
transfer_leader_latency_us	Raft 协议中转移 Leader 的延迟时间。
num_agent_heartbeats	AgentHBProcessor 心跳次数。
agent_heartbeat_latency_us	AgentHBProcessor 延迟时间。
replicate_log_latency_us	Raft 复制日志至大多数节点的延迟。
num_send_snapshot	Raft 发送快照至其他节点的次数。
append_log_latency_us	Raft 复制日志到单个节点的延迟。
append_wal_latency_us	Raft 写入单条 WAL 的延迟。
num_grant_votes	Raft 投票给其他节点的次数。
num_start_elect	Raft 发起投票的次数。

2022 vesoft Inc.

Storage

2022 vesoft Inc.

参数	说明
add_edges_atomic_latency_us	添加边单次延迟。
add_edges_latency_us	添加边的平均延迟时间。
add_vertices_latency_us	添加点的平均延迟时间。
commit_log_latency_us	Raft 协议中 Commit 日志的延迟时间。
commit_snapshot_latency_us	Raft 协议中 Commit 快照的延迟时间。
delete_edges_latency_us	删除边的平均延迟时间。
delete_vertices_latency_us	删除点的平均延迟时间。
get_neighbors_latency_us	查询邻居平均延迟时间。
num_get_prop	GetPropProcessor 执行的次数。
num_get_neighbors_errors	GetNeighborsProcessor 执行出错的次数。
get_prop_latency_us	GetPropProcessor 执行的延迟时间。
num_edges_deleted	删除的边数量。
num_edges_inserted	插入的边数量。
num_raft_votes	Raft 协议中投票的次数。
num_rpc_sent_to_metad_failed	Storaged 服务发给 Metad 服务的 RPC 请求失败的数量。
num_rpc_sent_to_metad	Storaged 服务发给 Metad 服务的 RPC 请求数量。
num_tags_deleted	删除的 Tag 数量。
num_vertices_deleted	删除的点数量。
num_vertices_inserted	插入的点数量。
transfer_leader_latency_us	Raft 协议中转移 Leader 的延迟时间。
lookup_latency_us	LookupProcessor 执行的延迟时间。
num_lookup_errors	LookupProcessor 执行时出错的次数。
num_scan_vertex	ScanVertexProcessor 执行的次数。
num_scan_vertex_errors	ScanVertexProcessor 执行时出错的次数。
update_edge_latency_us	UpdateEdgeProcessor 执行的延迟时间。
num_update_vertex	UpdateVertexProcessor 执行的次数。
num_update_vertex_errors	UpdateVertexProcessor 执行时出错的次数。
kv_get_latency_us	Getprocessor 的延迟时间。
kv_put_latency_us	PutProcessor 的延迟时间。
kv_remove_latency_us	RemoveProcessor 的延迟时间。
num_kv_get_errors	GetProcessor 执行出错次数。
num_kv_get	GetProcessor 执行次数。
num_kv_put_errors	PutProcessor 执行出错次数。
num_kv_put	PutProcessor 执行次数。
num_kv_remove_errors	RemoveProcessor 执行出错次数。

2022 vesoft Inc.

参数	说明
num_kv_remove	RemoveProcessor 执行次数。
forward_tranx_latency_us	传输平均延迟时间。
scan_edge_latency_us	ScanEdgeProcessor 执行的延迟时间。
num_scan_edge_errors	ScanEdgeProcessor 执行时出错的次数。
num_scan_edge	ScanEdgeProcessor 执行的次数。
scan_vertex_latency_us	ScanVertexProcessor 执行的延迟时间。
num_add_edges	添加边的次数。
num_add_edges_errors	添加边时出错的次数。
num_add_vertices	添加点的次数。
num_start_elect	Raft 发起投票的次数
num_add_vertices_errors	添加点时出错的次数。
num_delete_vertices_errors	删除点时出错的次数。
append_log_latency_us	Raft 复制日志到单个节点的延迟。
num_grant_votes	Raft 投票给其他节点的次数。
replicate_log_latency_us	Raft 复制日志到大多数节点的延迟。
num_delete_tags	删除 Tag 的次数。
num_delete_tags_errors	删除 Tag 时出错的次数。
num_delete_edges	删除边的次数。
num_delete_edges_errors	删除边时出错的次数。
num_send_snapshot	发送快照的次数。
update_vertex_latency_us	UpdateVertexProcessor 执行的延迟时间。
append_wal_latency_us	Raft 写入单条 WAL 的延迟。
num_update_edge	UpdateEdgeProcessor 执行的次数。
delete_tags_latency_us	删除 Tag 的平均延迟时间。
num_update_edge_errors	UpdateEdgeProcessor 执行时出错的次数。
num_get_neighbors	GetNeighborsProcessor 执行的次数。
num_get_prop_errors	GetPropProcessor 执行时出错的次数。
num_delete_vertices	删除点的次数。
num_lookup	LookupProcessor 执行的次数。

2022 vesoft Inc.

图空间级别监控指标

参数	说明
num_active_queries	当前正在执行的查询数。
num_queries	查询次数。
num_sentences	Graphd 服务接收的语句数。
optimizer_latency_us	优化器阶段延迟时间。
query_latency_us	查询平均延迟时间。
num_slow_queries	慢查询次数。
num_query_errors	查询报错语句数量。
num_query_errors_leader_changes	因查询错误而导致的 Leader 变更的次数。
num_killed_queries	被终止的查询数量。
num_aggregate_executors	聚合（Aggregate）算子的执行次数。
num_sort_executors	排序（Sort）算子的执行次数。
num_indexscan_executors	索引扫描（IndexScan）算子的执行次数。
num_oom_queries	导致内存耗尽的语句数量。
num_auth_failed_sessions_bad_username_password	因用户名密码错误导验证失败的会话数量。
num_auth_failed_sessions	登录验证失败的会话数量。
num_opened_sessions	服务端建立过的会话数量。
num_queries_hit_memory_watermark	达到内存水位线的语句的数量。
num_reclaimed_expired_sessions	服务端主动回收的过期的会话数量。
num_rpc_sent_to_metad_failed	Graphd 服务发给 Metad 的 RPC 请求失败的数量。
num_rpc_sent_to_metad	Graphd 服务发给 Metad 服务的 RPC 请求数量。
num_rpc_sent_to_storaged_failed	Graphd 服务发给 Storaged 服务的 RPC 请求失败的数量。
num_rpc_sent_to_storaged	Graphd 服务发给 Storaged 服务的 RPC 请求数量。
slow_query_latency_us	慢查询平均延迟时间。

最后更新: June 20, 2023

2022 vesoft Inc.

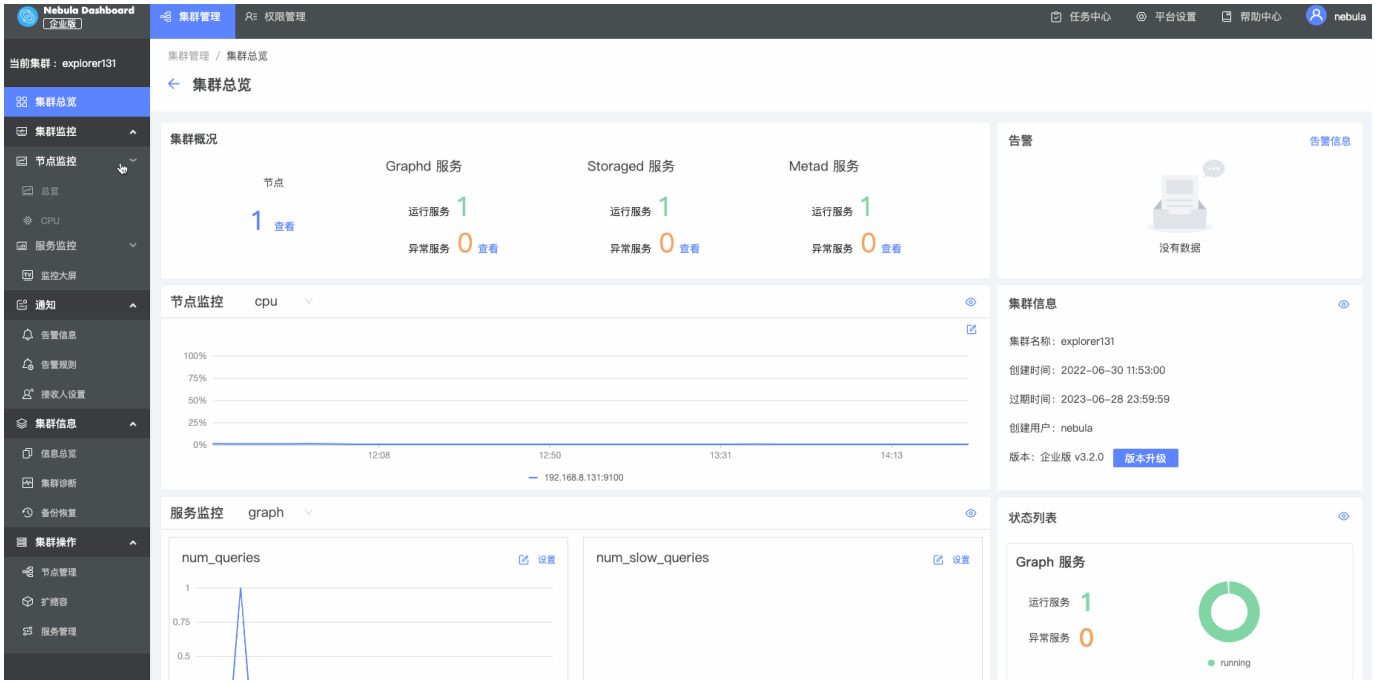
16. Nebula Dashboard（企业版）

16.1 什么是 Nebula Dashboard（企业版）

Nebula Dashboard（简称 Dashboard）是一款用于监控和管理 Nebula Graph 多集群中机器和服务状态的可视化工具。本文主要介绍企业版 Dashboard，社区版详情参见什么是 Nebula Dashboard（社区版）。

Note

用户也可以在线试用 Dashboard部分功能。



16.1.1 产品功能

- 创建指定版本的 Nebula Graph 集群，支持批量导入节点、一键添加服务等功能。
- 支持在可视化界面进行集群导入、数据平衡、扩容缩容等操作。
- 支持管理多集群并可以查看最近 14 天内集群的操作记录。
- 支持在可视化页面进行服务启动、停止、重启操作。
- 支持快速更新集群中 Storage 及 Graph 服务的配置。
- 支持全局调整监控数据的页面更新频率。
- 监控集群中所有服务的信息，包括服务 IP 地址、版本和监控指标（例如查询数量、查询延迟、心跳延迟等）。
- 监控集群中所有机器的状态，包括 CPU、内存、负载、磁盘和流量。
- 监控集群本身的信息，包括集群的服务信息、分区信息、配置和长时任务。
- 支持根据监控信息设置告警通知。

16.1.2 适用场景

- 针对大规模集群，需要可视化的运维监控平台。
- 需要方便快捷地监测关键指标，集中呈现业务的多个重点信息，保证业务正常运行。

2022 vesoft Inc.

- 需要多维度（例如时间段、聚合规则、指标）监控集群。
- 故障发生后，需要复盘问题，确认故障发生时间、异常现象。

16.1.3 注意事项

- 监控数据默认保留 14 天，即只能查询最近 14 天内任意时间段的监控数据。
- 只支持 2.5.0 及以上版本的 Nebula Graph。
- 建议使用最新版本的 Chrome 访问 Dashboard。
- 建议使用官方提供的安装包进行集群创建或导入。

> **Note**
>
> 监控服务由 prometheus 提供，更新频率和保留时间等都可以自行修改。详情请参见 prometheus 官方文档。

16.1.4 版本兼容性

Nebula Graph 的版本和 Dashboard 企业版的版本对应关系如下。

Nebula Graph 版本	**Dashboard** 版本
2.5.0 ~ 3.1.0	3.1.0
2.5.x ~ 3.1.0	3.0.4
2.5.1 ~ 3.0.0	1.1.0
2.0.1 ~ 2.6.1	1.0.2
2.0.1 ~ 2.6.1	1.0.1
2.0.1 ~ 2.6.1	1.0.0

16.1.5 视频

- Nebula Dashboard Demo 介绍（v3.0 版本）（2 分 57 秒）

最后更新: June 20, 2023

2022 vesoft Inc.

16.2 部署 Dashboard 企业版

本文将详细介绍如何安装并部署 Dashboard 企业版。

16.2.1 前提条件

在部署 Dashboard 之前，用户需要确认以下信息：

- 选择并下载符合版本的 Dashboard。关于 Dashboard 和 Nebula Graph 的版本对应关系，参见版本兼容性。
- Dashboard 支持 SQLite 和 MySQL 存储 Dashboard 元数据。如果用户使用 MySQL，需准备版本为 5.7 以上的 MySQL 环境，创建名称为 `dashboard` 的数据库，字符集为 `utf8` 。

  Note

  如果使用 SQLite 存储 Dashboard 元数据，则无需准备 SQLite 环境。

- 确保在安装开始前，以下端口处于未被使用状态。

端口号	说明
7005	Dashboard 提供 web 服务的端口。
9090	Prometheus 服务的端口。
9200	nebula-stats-exporter 服务的端口。
9093	Alertmanager 服务的端口。接收 Prometheus 告警，发送告警通知给 Dashboard。
9100	node-exporter 的服务端口。在集群创建后，自动部署在目标机器内，收集集群中机器的资源信息，包括 CPU、内存、负载、磁盘和流量。

- 准备 License。

  Enterpriseonly

  License 仅在企业版提供，申请 License 需填写 Nebula Dashboard 试用申请。

16.2.2 使用 TAR 包部署

安装

1. 根据需要下载 TAR 包，建议选择最新版本。

   Enterpriseonly

   用户可以在线申请试用 Dashboard 企业版；如需购买，通过邮箱(inquiry@vesoft.com)联系销售人员。点击定价查看更多。

2. 使用 `tar -xzvf` 解压 TAR 包。

```
$ tar -xzvf nebula-dashboard-ent-<version>.linux-amd64.tar.gz -C <install_path>
```

例如：

```
$ tar -xzvf nebula-dashboard-ent-3.1.0.linux-amd64.tar.gz -C /usr/local/
```


2022 vesoft Inc.

3. 使用 `vim /usr/local/nebula-dashboard-ent/config/config.yaml` 命令设置配置文件。

```
server:
  host: 0.0.0.0 # 指定能访问 Dashboard 的地址段。
  port: 7005 # 访问 Dashboard 企业版服务的默认端口。
  certFile: "" # 通过 HTTPS 访问 Dashboard 的本地设备证书路径。
  keyFile: "" # 通过 HTTPS 访问 Dashboard 的本地设备证书密钥路径。

database:
  dialect: sqlite # 使用的数据库类型，目前支持 SQLite 和 MySQL，默认使用 SQLite。
  autoMigrate: true # 自动创建数据库表，默认为 true。
  # SQLite 相关信息，当 dialect 为 sqlite 时配置。
  sqliteFilePath: data/dashboard.sqlite.db # SQLite 数据库保存信息的默认路径。
  # MySQL 相关信息，当 dialect 为 mysql 时配置。
  host: 192.168.8.157 # 连接的 MySQL 数据库的 IP 地址。
  port: 3306 # 连接的 MySQL 数据库的端口号。
  username: root # 登陆 MySQL 的账户。
  password: nebula # 登陆 MySQL 的密码。
  name: dashboard # 对应的数据库名称。

# exporter 端口信息
exporter:
  nodePort: 9100 # node-exporter 服务的端口。
  nebulaPort: 9200 # nebula-stats-exporter 服务的端口。

# 服务信息
proxy:
  prometheus:
    target: "127.0.0.1:9090" # prometheus 服务的 IP 地址和端口。
  alertmanager:
    target: "127.0.0.1:9093" # alertmanager 服务的 IP 地址和端口。

# 邀请 LDAP 账号的发件人邮箱信息
mail:
  host: smtp.office365.com #  SMTP 服务器地址。
  port: 587 # SMTP 服务器的端口号。
  username: "" # SMTP 服务器的用户名。
  password: "" # SMTP 服务器的密码。

# 系统信息
system:
  webAddress: http://127.0.0.1:7005 # 外网访问 Dashboard 的地址，也可以设置为域名。用于一些接口回调，例如邮件被邀请人可以使用此链接访问 Dashboard。
  messageStore: 90 # 保存告警消息的天数，默认 90 天。

# LDAP 信息
ldap:
  server: ldap://127.0.0.1 # LDAP 服务器地址。
  bindDN: cn=admin,dc=vesoft,dc=com # LDAP 服务器登录用户名。
  bindPassword: "" # LDAP 服务器登录密码。
  baseDN: dc=vesoft,dc=com # 查询用户数据的路径。
  userFilter: "&(objectClass=*)" # 查询条件。
  emailKey: mail # LDAP 数据库存放邮箱信息的字段名。
...
# 备份和恢复
br:
backupLogsDir: "logs/br/backupLogsDir"
restoreLogsDir: "logs/br/restoreLogsDir"
```

4. 拷贝 License 至 `nebula-dashboard-ent` 目录下。

```
$ cp -r <license> <dashboard_path>
```

例如：

```
$ cp -r nebula.license /usr/local/nebula-dashboard-ent
```

5. 启动 Dashboard。

可以使用以下命令一键启动 Dashboard。

```
$ cd /usr/local/nebula-dashboard-ent/scripts
$ sudo ./dashboard.service start all
```

或是执行以下命令，分别启动 Prometheus、nebula-dashboard-server、nebula-stats-exporter 和 Alertmanager 依赖服务以启动 Dashboard。

```
$ cd scripts
$ sudo ./dashboard.service start prometheus # 启动 Prometheus 服务
$ sudo ./dashboard.service start webserver #启动 nebula-dashboard-server 服务
$ sudo ./dashboard.service start exporter #启动 nebula-stats-exporter 服务
$ sudo ./dashboard.service start alertmanager #启动 Alertmanager 服务
```


2022 vesoft Inc.

Note

在启动 Dashboard 后，如再修改配置文件的话，在修改后，可在 `scripts` 目录下执行 `dashboard.service restart all` 同步修改至 Dashboard 客户端页面中。

管理 **Dashboard** 服务

Dashboard 使用脚本 `dashboard.service` 管理服务，包括启动、停止和查看。

```
$ sudo <dashboard_path>/dashboard/scripts/dashboard.service
[-v] [-h]
<start|stop|status>  <prometheus|webserver|exporter|gateway|all>
```

参数	说明
`dashboard_path`	Dashboard 安装路径。
`-v`	显示详细调试信息。
`-h`	显示帮助信息。
`start`	启动服务。
`stop`	停止服务。
`status`	查看服务状态。
`prometheus`	管理 prometheus 服务。
`webserver`	管理 webserver 服务。
`exporter`	管理 exporter 服务。
`gateway`	管理 gateway 服务。
`all`	管理所有服务。

示例

Dashboard 的安装在当前目录下，用户可以用以下命令管理服务。

```
$ sudo /dashboard/scripts/dashboard.service start all #启动 Dashboard 所有服务。
$ sudo /dashboard/scripts/dashboard.service stop all #停止 Dashboard 所有服务。
$ sudo /dashboard/scripts/dashboard.service status all #查看 Dashboard 所有服务状态。
$ sudo /dashboard/scripts/dashboard.service restart all #重启 Dashboard 所有服务状态。
```

16.2.3 使用 RPM 包部署

安装

1. 下载 RPM 包。

...terpriseonly

用户可以在线申请试用 Dashboard 企业版；如需购买，通过邮箱(inquiry@vesoft.com)联系销售人员。点击定价查看更多。

2. 使用 `sudo rpm -ivh <package_name>` 命令安装 RPM 包。

   例如，安装 Dashboard 企业版需要运行以下命令，默认安装路径为 `/usr/local/nebula-dashboard-ent`：

   ```
   sudo rpm -ivh nebula-dashboard-ent-<version>.x86_64.rpm
   ```

   也可以使用以下命令安装到指定路径：

2022 vesoft Inc.

```
sudo rpm -ivh nebula-dashboard-ent-xxx.rpm --prefix=<path>
```

在可视化的安装过程中，用户需要输入 License 的路径，选择存放元数据的数据库，目前支持 SQLite 和 MySQL。

例如：

SQLite

```
Nebula Dashboard Enterprise version need license, please enter the license file path(~/nebula.license): /home/vesoft/license/nebula.license

Which database do you want to use? [sqlite | mysql]:sqlite # 不填默认为 SQLite。

Do you want to start the service now? [Y/N]: y
```

MySQL

```
Nebula Dashboard Enterprise version need license, please enter the license file path(~/nebula.license): /home/vesoft/license/nebula.license  # license的存放路径。

Which database do you want to use? [sqlite | mysql]:mysql

Step1: set mysql database config

Enter mysql host(127.0.0.1): 192.168.8.157  # 连接的 MySQL 数据库的 IP 地址。

Enter mysql service port(3306):  # 连接的 MySQL 数据库的端口号。默认3306。

Enter mysql username(root): # 登录 MySQL 的账户。

Enter mysql password(nebula): # 登录 MySQL 的密码。

Enter mysql database name(dashboard): # MySQL 数据库名称。

Do you want to start the service now? [Y/N]: y
```

3. （可选）使用以下命令查看所有服务状态、启动服务。

```
sudo systemctl list-dependencies nebula-dashboard.target  # 查看状态
sudo systemctl start nebula-dashboard.target   # 启动服务
```

用户也可以执行以下命令查看、启停单一服务，例如：

```
sudo systemctl {status|stop|start}  {nbd-prometheus.service|nbd-alert-manager.service|nbd-stats-exporter.service|nbd-webserver.service}
```

4. （可选）如需配置集群告警信息的接收人及配置 LDAP 账户，使用 `vim /usr/local/nebula-dashboard-ent/config/config.yaml` 命令添加以下配置。

```
# 邀请 LDAP 账号的发件人邮箱信息
mail:
  host: smtp.office365.com #  SMTP 服务器地址。
  port: 587 # SMTP 服务器的端口号。
  username: "" # SMTP 服务器的用户名。
  password: "" # SMTP 服务器的密码。
# 系统信息
system:
  webAddress: http://127.0.0.1:7005 # 邮件被邀请人连接 Dashboard 的访问地址。
  messageStore: 90 # 保存告警消息的天数，默认 90 天。
# LDAP 信息
ldap:
  server: ldap://127.0.0.1 # LDAP 服务器地址。
  bindDN: cn=admin,dc=vesoft,dc=com # LDAP 服务器登录用户名。
  bindPassword: "" # LDAP 服务器登录密码。
  baseDN: dc=vesoft,dc=com # 查询用户数据的路径。
  userFilter: "&(objectClass=*)" # 查询条件。
  emailKey: mail # LDAP 数据库存放邮箱信息的字段名。
```

查看日志

用户可以在 `/var/log/messages` 路径下查看 Dashboard 企业版的操作日志。

例如：

```
sudo cat /var/log/messages
```

或者执行以下命令查看 Dashboard 中各个服务的日志：

```
journalctl -u {nbd-prometheus.service|nbd-alert-manager.service|nbd-stats-exporter.service|nbd-webserver.service} -b
```


2022 vesoft Inc.

例如，查看 Prometheus 服务的日志：

```
journalctl -u nbd-prometheus.service -b
```

卸载

执行以下命令卸载通过 RPM 包部署的 Dashboard。

```
sudo rpm -e <package_name>
```

16.2.4 使用 DEB 包部署

安装

1. 下载 DEB 包。

> **Enterpriseonly**

用户可以在线申请试用 Dashboard 企业版；如需购买，通过邮箱(inquiry@vesoft.com)联系销售人员。点击定价查看更多。

2. 使用 `sudo dpkg -i <package_name>` 命令安装 DEB 包。

例如，安装 Dashboard 企业版需要运行以下命令，默认安装路径为 `/usr/local/nebula-dashboard-ent`：

```
sudo dpkg -i nebula-dashboard-ent-3.1.0.ubuntu1804.amd64.deb
```

> **Note**

使用 DEB 包安装 Dashboard 企业版时不支持自定义安装路径。

在可视化的安装过程中，用户需要输入 License 的路径，选择存放元数据的数据库，目前支持 SQLite 和 MySQL。

例如：

SQLite

```
Nebula Dashboard Enterprise version need license, please enter the license file path(~/nebula.license): /home/vesoft/license/nebula.license

Which database do you want to use? [sqlite | mysql]:sqlite # 不填默认为 SQLite。

Do you want to start the service now? [Y/N]: y
```

MySQL

```
Nebula Dashboard Enterprise version need license, please enter the license file path(~/nebula.license): /home/vesoft/license/nebula.license  # license的存放路径。

Which database do you want to use? [sqlite | mysql]:mysql

Step1: set mysql database config

Enter mysql host(127.0.0.1): 192.168.8.157  # 连接的 MySQL 数据库的 IP 地址。

Enter mysql service port(3306):  # 连接的 MySQL 数据库的端口号。默认3306。

Enter mysql username(root): # 登录 MySQL 的账户。

Enter mysql password(nebula): # 登录 MySQL 的密码。

Enter mysql database name(dashboard): # MySQL 数据库名称。

Do you want to start the service now? [Y/N]: y
```

3. （可选）使用以下命令查看所有服务状态、启动服务。

```
sudo systemctl list-dependencies nebula-dashboard.target  # 查看状态
sudo systemctl start nebula-dashboard.target   # 启动服务
```


2022 vesoft Inc.

用户也可以执行以下命令查看、启停单一服务，例如：

```
sudo {systemctl status|stop|start}  {nbd-prometheus.service|nbd-alert-manager.service|nbd-stats-exporter.service|nbd-webserver.service}
```

4. （可选）如需配置集群告警信息的接收人及配置 LDAP 账户，使用 `vim /usr/local/nebula-dashboard-ent/config/config.yaml` 命令添加以下配置。

```
# 邀请 LDAP 账号的发件人邮箱信息
mail:
  host: smtp.office365.com #  SMTP 服务器地址。
  port: 587 # SMTP 服务器的端口号。
  username: "" # SMTP 服务器的用户名。
  password: "" # SMTP 服务器的密码。
# 系统信息
system:
  webAddress: http://127.0.0.1:7005 # 邮件被邀请人连接 Dashboard 的访问地址。
  messageStore: 90 # 保存告警消息的天数，默认 90 天。
# LDAP 信息
ldap:
  server: ldap://127.0.0.1 # LDAP 服务器地址。
  bindDN: cn=admin,dc=vesoft,dc=com # LDAP 服务器登录用户名。
  bindPassword: "" # LDAP 服务器登录密码。
  baseDN: dc=vesoft,dc=com # 查询用户数据的路径。
  userFilter: "&(objectClass=*)" # 查询条件。
  emailKey: mail # LDAP 数据库存放邮箱信息的字段名。
```

查看日志

用户可以在 `/var/log/syslog` 路径下查看 Dashboard 企业版的操作日志。

例如：

```
sudo cat /var/log/syslog
```

或者执行以下命令查看 Dashboard 中各个服务的日志：

```
journalctl -u {nbd-prometheus.service|nbd-alert-manager.service|nbd-stats-exporter.service|nbd-webserver.service} -b
```

例如，查看 Prometheus 服务的日志：

```
journalctl -u nbd-prometheus.service -b
```

卸载

执行以下命令卸载通过 DEB 包部署的 Dashboard。

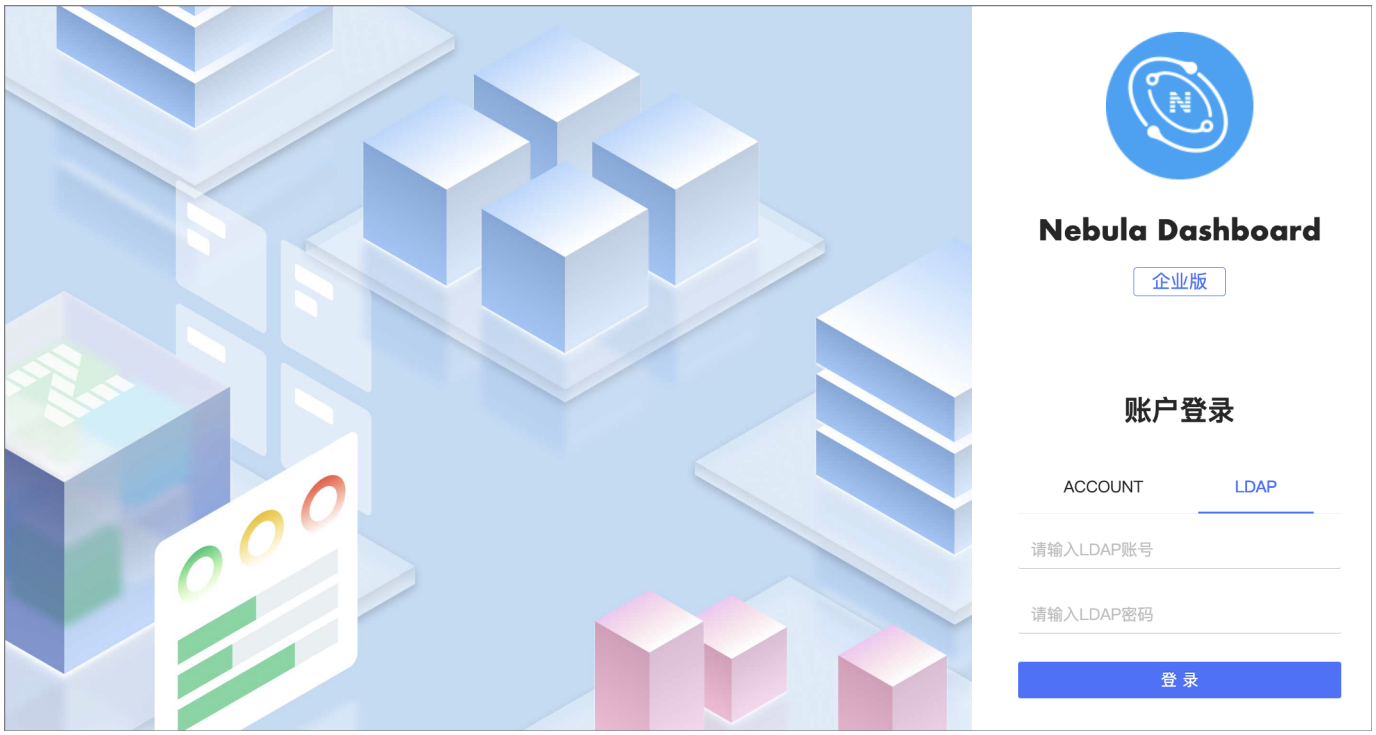
```
sudo dpkg -r <package_name>
```

16.2.5 连接 Dashboard

成功启动 Dashboard 后，在浏览器地址栏输入 `http://<ip_address>:7005` 。

如在浏览器窗口中看到以下登录界面，则表示已经成功部署并启动了 Dashboard。

2022 vesoft Inc.



Note

首次登录企业版 Nebula Dashboard 的时候，页面显示最终用户许可协议的内容，请仔细阅读并单击同意。

用户可以使用初始化用户名 `nebula` 和密码 `nebula` 登录 Dashboard。登录 Dashboard 后可创建 LDAP 账户和普通账户，然后使用不同的账户登录 Dashboard。关于账户的详情，参见权限管理。

最后更新: June 20, 2023

2022 vesoft Inc.

16.3 Nebula Dashboard 企业版 License

License 是为用户提供的软件授权证书。用户在部署 Nebula Dashboard 企业版时，需要设置 License 才能正常启动 Nebula Dashboard 企业版。本文介绍 Nebula Dashboard 企业版的 License 相关信息。

16.3.1 注意事项

- 没有设置 License 时，Nebula Dashboard 企业版无法启动。
- 请勿修改 License，否则会导致 License 失效。
- License 快过期时，请及时发送邮件至**inquiry@vesoft.com**申请续期。
- License 的过期缓冲为 14 天：
- 过期前 30 天和过期当天，服务启动时会打印日志进行提醒。
- 过期后仍可继续使用 14 天。
- 过期 14 天后，服务无法启动，并会打印日志进行提醒。

16.3.2 获取方式

用户需发送邮件至**inquiry@vesoft.com**申请 Nebula Dashboard 企业版 License。

> **Note**
>
> 填写 Nebula Dashboard 试用申请可获取 30 天免费试用的 License。

16.3.3 Nebula Dashboard 企业版 License 说明

Nebula Dashboard 企业版 License 是一个名为 `nebula.license` 的文件，内容示例如下：

```
----------License Content Start----------
{
  "vendor": "vesoft",
  "organization": "doc",
  "issuedDate": "2022-06-06T16:00:00.000Z",
  "expirationDate": "2023-05-31T15:59:59.000Z",
  "product": "nebula_graph_dashboard",
  "version": ">=3.2.0",
  "licenseType": "enterprise",
  "gracePeriod": 14,
  "clusterCode": "BAIAEAiAQAAG"
}
----------License Content End----------

----------License Key Start----------
Rrjip5cxxxxxxxxxxxxx5zKoQ==
----------License Key End----------
```


2022 vesoft Inc.

License 文件包含生效时间、过期时间等信息。说明如下。

参数	说明
`vendor`	发放渠道。
`organization`	用户名称。
`issuedDate`	License 生效时间。
`expirationDate`	License 过期时间。
`product`	产品类型。Nebula Dashboard 企业版的产品类型为 `nebula_graph_dashboard` 。
`version`	版本支持的信息。
`licenseType`	License 类型（预留参数）。包括 `enterprise` 、`samll_bussiness` 、`pro` 、`individual` 。
`gracePeriod`	证书过期后可继续使用服务的缓冲时间（单位天），超过缓冲期后停止服务。试用版的 License 过期后无缓冲期，默认值为 0。
`clusterCode`	用户的硬件信息，也是集群的唯一标识码。试用版的 License 中无此参数。

16.3.4 使用方式

关于如何使用 Nebula Dashboard 企业版 License，参见部署 Dashboard 企业版。

16.3.5 续期操作

按照以下操作续期 Nebula Dashboard 企业版 License。

1. 发送邮件至 `inquiry@vesoft.com` 申请新的 Nebula Dashboard 企业版 License 文件 `nebula.license` 。
2. 在 Nebula Dashboard 企业版的安装目录下（例如，`/usr/local/nebula-dashboard-ent`），使用新的 License 文件替换旧的 License 文件。

> **Note**
>
> Nebula Dashboard 企业版 License 过期后，用户无法登录 Dashboard。为了保证业务正常运行，请及时更新 License。

最后更新: June 20, 2023

2022 vesoft Inc.

16.4 创建及导入集群

16.4.1 创建集群

本文介绍如何通过 Dashboard 创建集群。

2022 vesoft Inc.

操作步骤

按以下方式创建集群:

2022 vesoft Inc.

1. 在 Dashboard 页面顶部，单击集群管理。
2. 在集群管理页面，单击创建集群标签。

2022 vesoft Inc.

3. 在创建集群页面，完成以下配置：

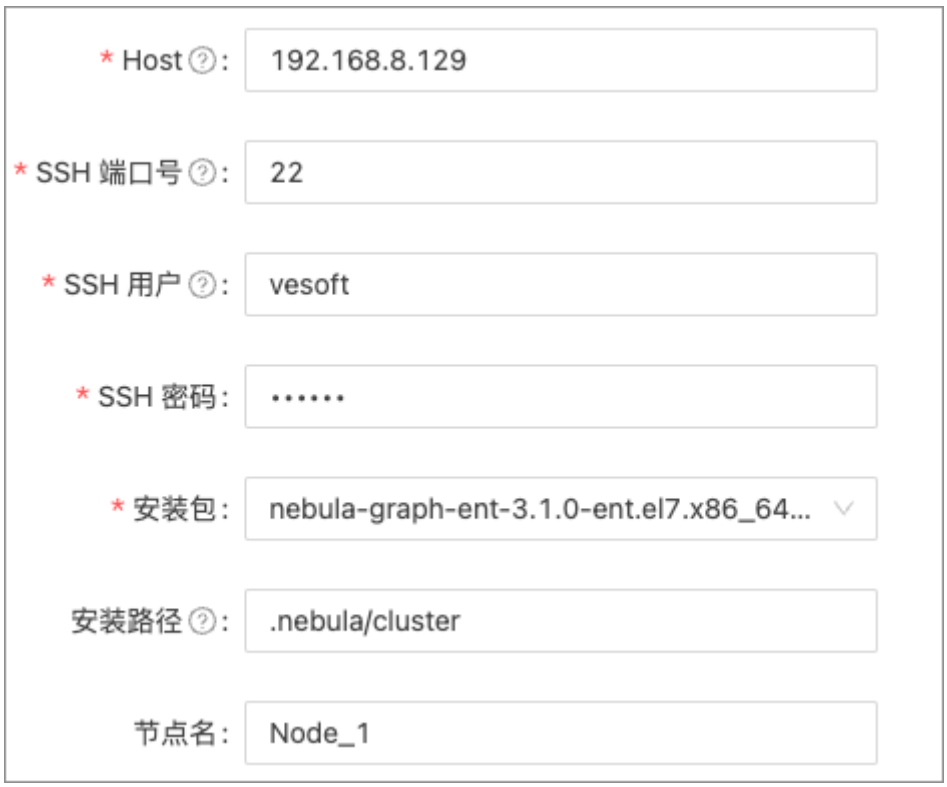
- 输入集群名称，最大可输入 15 个字符，本示例设置为 `test` 。
- 选择 Nebula Graph 安装版本。本示例设置为 `企业版 v3.1.0` 。

> **Note**
>
> Dashboard 默认内置企业版的 Nebula Graph 安装包。如果用户需要安装其他版本的 Nebula Graph，可以在安装包管理页面下载或上传，详情参见安装包管理。

- 单击上传证书。

> **Note**
>
> 如果创建社区版的 Nebula Graph，则不需要上传 License 证书，忽略此步骤。

- 添加节点信息，提供两种方式：
- 添加节点，添加单个节点的相关信息，可多次添加。



a. 配置每个 Host 的 IP 信息，本示例设置为 `192.168.8.129` 。

b. 配置 SSH 信息，本示例设置如下：SSH 端口号为 `22` ，SSH 用户名为 `vesoft` ，SSH 密码为 `nebula` 。

c. 选择 Nebula Graph 安装包，本示例为 `nebula-graph-ent-3.1.0-ent.el7.x86_64.rpm` 。

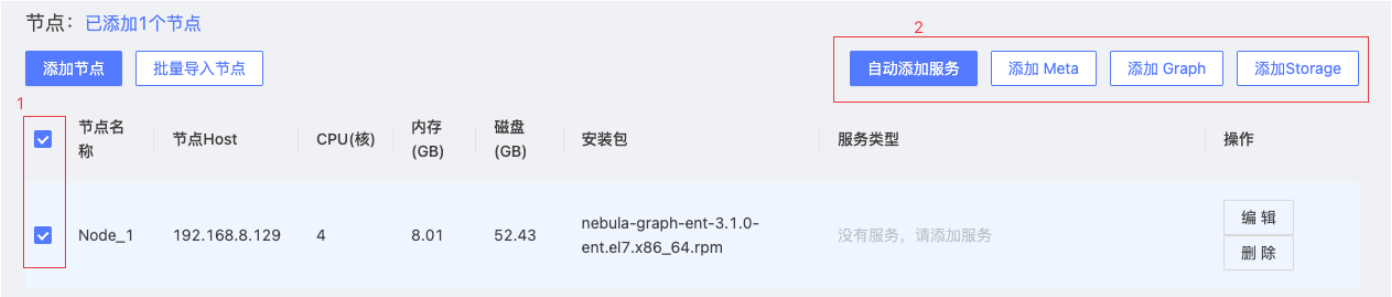
d. 自定义集群的安装路径。本示例为默认安装路径 `.nebula/cluster` 。

e. （可选）输入节点名，可以对节点进行备注。本示例设置为 `Node_1` 。

- 批量导入节点，批量添加多个节点的相关信息。批量导入需要先选择安装包后单击下载 **CSV** 模版。按照模版填写后上传，尽量确保节点信息正确，否则容易造成上传失败。

2022 vesoft Inc.

4. 勾选节点并在节点右上方，单击需要添加的服务。创建集群需要给节点添加 3 种类型的服务。如果不熟悉 Nebula Graph 架构，建议单击自动添加服务按钮。



5. （可选）在下方的服务中，选择编辑 Meta、Graph、Storage 服务的端口号、HTTP 端口号，单击确认保存。
6. 单击创建集群，确定配置信息无误且节点无冲突后，单击确认。
7. 在集群管理页面中的列表中出现状态为 `installing` 的集群，需等待 3-10 分钟，状态变为 `healthy` 即集群创建成功。如果服务状态为 `unhealthy`，表示集群服务中存在非正常运行的服务，请单击详情进行查看。

后续操作

成功创建集群后，用户可以对集群进行操作，详情见集群操作。

最后更新: June 20, 2023

2022 vesoft Inc.

16.4.2 导入集群

本文介绍如何通过 Dashboard 导入集群。当前版本仅支持官方下载的 DEB、RPM 包部署的集群和 Dashboard 创建的集群导入，暂不支持导入使用 Docker 和 Kubernetes 方式部署的集群。

2022 vesoft Inc.

操作步骤

Caution

在同一集群下，服务需要统一版本。不支持在同一集群中导入不同版本的 Nebula Graph 实例。

1. 在各个服务的配置文件中，将 `<meta|graph|storage>_server_addrs` 和 `local_ip` 中的IP地址改为本机的IP地址，然后启动服务。

   具体步骤，参见配置管理和管理服务。

2. 在集群管理页面，单击导入集群标签。

3. 在导入集群页面，输入连接 Nebula Graph 的信息：

- Graphd Host：<其中一个 Graphd 进程的虚机 IP>:<端口号>。本示例设置为 `192.168.8.157:9669`。
- 用户名：连接 Nebula Graph 的账号，本示例设置为 `vesoft`。
- 密码：连接 Nebula Graph 的密码，本示例设置为 `nebula`。

Note

因为 Nebula Graph 默认不启用身份验证，所以，一般情况下用户可以使用 `root` 账号和任意密码连接 Nebula。 当 Nebula Graph 启用了身份验证后，用户只能使用指定的账号和密码连接 Nebula。关于 Nebula Graph 的身份验证功能，参考 Nebula Graph 用户手册。

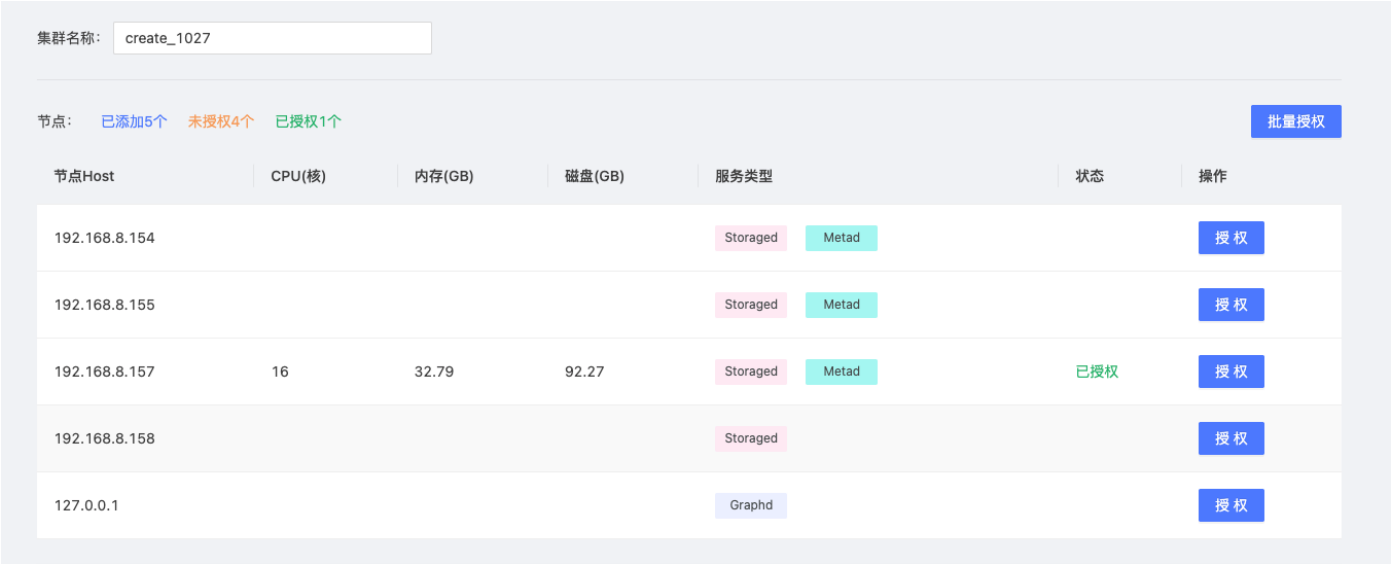
4. 在连接服务节点页面完成以下配置：

- 输入集群的名称，最大可输入 15 个字符，本示例设置为 `create_1027`。
- 对节点进行授权，授权需输入每个节点的 SSH 用户名及密码，以及选择是否使用 `sudo` 执行命令。

Notice

如果用户 SSH 账号没有操作 Nebula Graph 的权限，但是可以免密执行 `sudo` 命令，需要选择使用 **sudo**。

- 批量授权需要上传 CSV 文件。请根据下载的 CSV 文件，编辑每个节点授权信息，尽量确保节点信息正确，否则容易造成上传失败。
- 页面中节点状态变为已授权，则该节点授权成功。



5. 确保所有节点都授权成功，单击导入集群。

2022 vesoft Inc.

后续操作

成功导入集群后，用户可以对集群进行操作，详情见总览。

最后更新: June 20, 2023

2022 vesoft Inc.

16.5 集群管理

16.5.1 集群总览

本文主要介绍 Dashboard 的集群总览页面。

在 Dashboard 页面顶部，单击集群管理，然后在集群列表右侧，单击详情，即可进入指定集群的集群总览页面。

概览

Dashboard 的集群总览页面分为五个部分：

- 集群概况
- 告警
- 集群信息
- 节点监控
- 状态列表
- 服务监控

集群概况

在集群概况中，显示了节点数量，Graphd、Storaged、Metad 正在运行服务及异常服务数量。单击查看按钮，快速查看节点信息及异常服务。

告警

在告警区域，系统按照告警消息的严重级别（`emergency` > `critical` > `warning`）展示最近触发的 5 条告警消息。

单击告警信息查看告警信息列表。有关告警消息的详情及设置，参见监控告警。

集群信息

集群信息区域显示集群名称、创建时间、过期时间、创建用户、和版本的信息。

- 集群名称：显示 Nebula Graph 集群的名称。
- 创建时间：显示 Nebula Graph 集群的创建时间。
- 过期时间：显示 Nebula Graph 集群对应的 License 的过期时间。

> **terpriseonly**
>
> 只有当创建或导入的集群为 Nebula Graph 企业版时，才会显示过期时间。

- 创建用户：显示创建或导入的 Nebula Graph 集群时使用的 Dashboard 用户名。
- 版本：显示 Nebula Graph 集群的版本信息。右侧有版本升级按钮，方便快速跳转至版本升级页面。

在集群信息区域右上方，单击  进入集群信息详情页面，查看集群的详细信息，包括名字、创建时间、创建用户、版本、创建用户的角色。

> **terpriseonly**
>
> 针对 Nebula Graph 企业版集群，页面还会显示证书区域：
>
> - 显示集群使用的 License 的详细信息，包括集群 License 的状态、认证对象、生效时间、过期时间、支持创建的集群版本、证书类型。
> - 提供上传证书功能。集群 **License** 过期后，无法进行当前集群的操作。可单击上传证书上传新的 License 文件。

2022 vesoft Inc.

节点监控

- 支持快速查看节点监控信息，可单击切换展示的信息，默认显示 CPU 信息。
- 支持在页面上单击  插入基准线。
- 单击  快速跳转至节点监控页面，查看详情信息。

状态列表

用饼图的方式形象的展示节点运行服务状态。

服务监控

- 默认显示 `query_latency_us`（查询平均延迟）和 `slow_query_latency_us`（慢查询平均延迟）的信息。
- 单击  设置 插入基准线。
- 单击  快速跳转至服务监控页面，查看详情信息。

最后更新: June 20, 2023

2022 vesoft Inc.

16.5.2 集群监控

本文主要介绍 Dashboard 的集群监控中的节点监控、服务监控以及可视化的监控大屏。

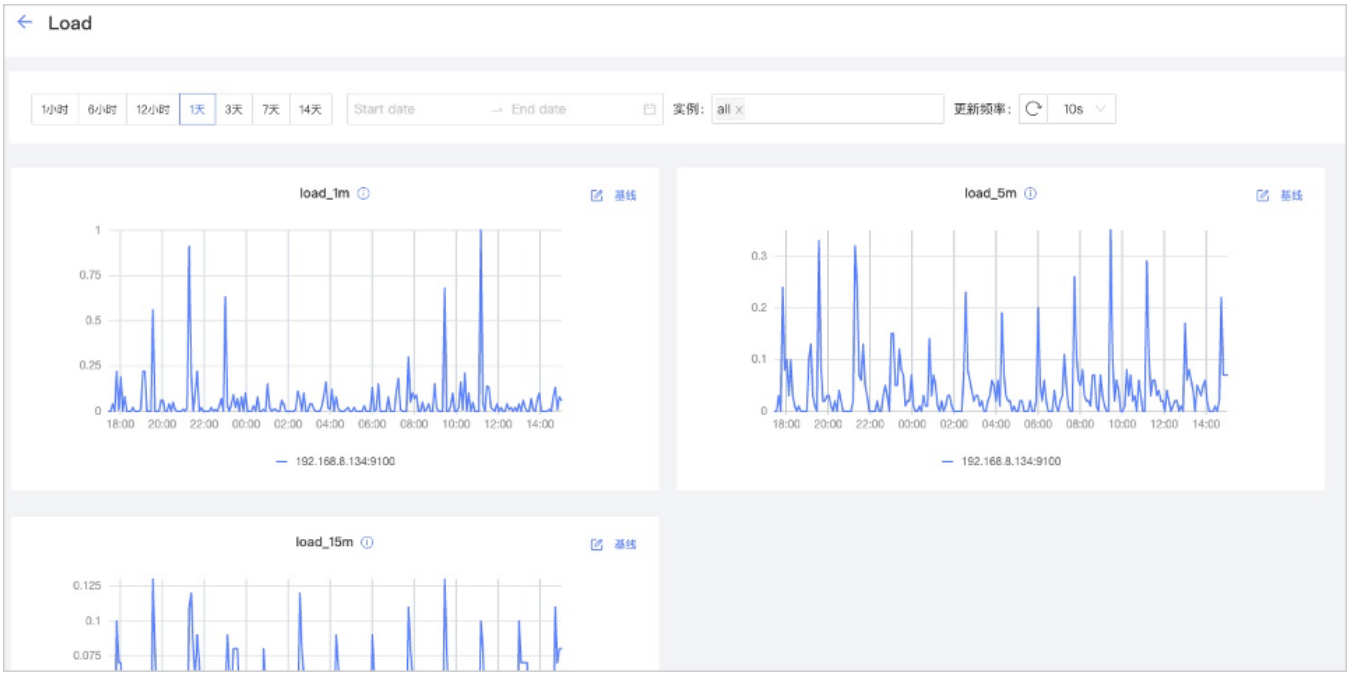
在 Dashboard 顶部导航栏，单击集群管理，在目标集群右侧单击详情，在左侧导航栏的集群监控下包含节点监控、服务监控和监控大屏。

节点监控

单击集群监控->节点监控进入节点监控页面。

用户可快速查看 CPU、Memory、Load、Disk 和 Network In/Out 变化情况。

- 默认最多可选择 14 天的监控数据进行查看，支持选择时间段，也可以快捷选择最近 1 小时、6 小时、12 小时、1 天、3 天、7 天和 14 天。
- 默认勾选显示所有实例的监控数据，可以在实例框内调整。
- 页面的监控数据默认不自动更新，可以调整更新频率让页面自动更新，也可以单击 按钮手动更新。
- 如果需要设置基线，作为参考标准线，可以单击模块右上角的 按钮。
- 如果需要查看某一项更详细的监控指标，可以单击模块右上角的 按钮，在示例中选择 Load 查看详情信息，如下图。



- 可以设置监控时间段、实例、更新频率和基线。
- 可以选择需要查看的机器和监控指标。监控指标详情请参见监控指标说明。

服务监控

单击集群监控->服务监控进入服务监控页面。

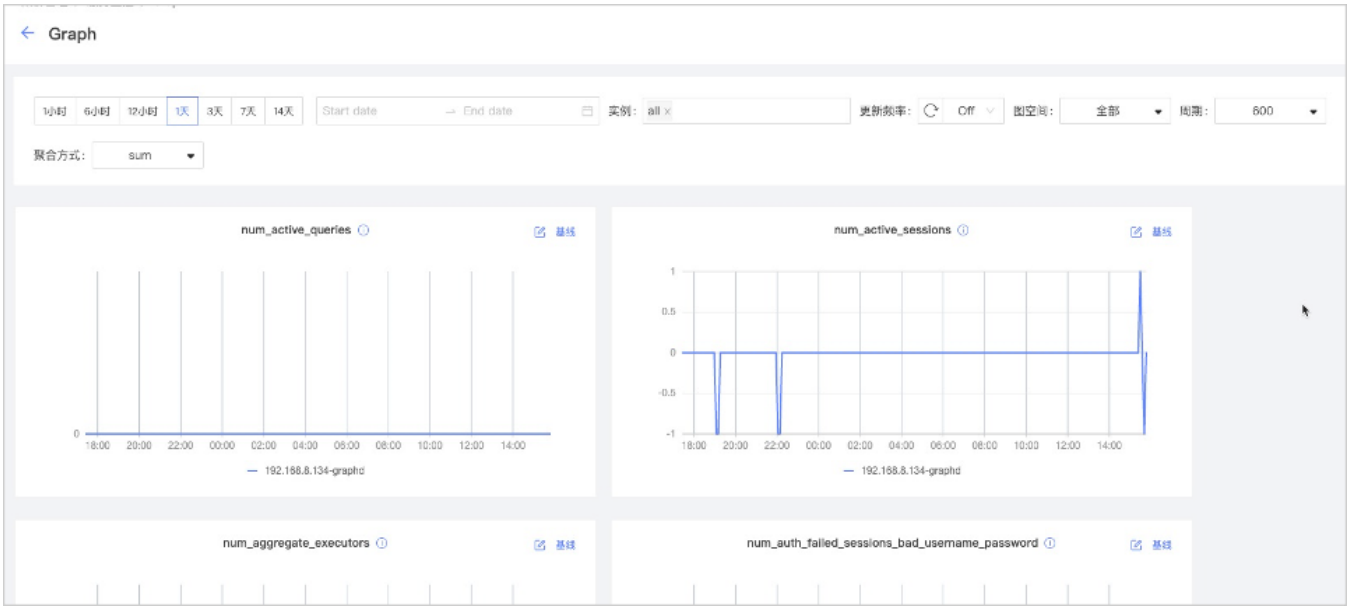
用户可快速查看 Graph、Meta、Storage 服务的信息，右上角显示正常服务和异常服务的数量。

2022 vesoft Inc.

Note

当前企业版的服务监控页仅支持每种服务设置两个监控指标，可以单击模块内的设置按钮进行调整。

- 默认最多可选择 14 天的监控数据进行查看，支持选择时间段，也可以快捷选择最近 1 小时、6 小时、12 小时、1 天、3 天、7 天和 14 天。
- 默认勾选显示所有实例的监控数据，可以在实例框内调整。
- 页面的监控数据默认不自动更新，可以调整更新频率让页面自动更新，也可以单击 按钮手动更新。
- 可以查看集群内所有服务的状态。
- 如果需要查看某一项更详细的监控指标，可以单击模块右上角的 按钮，在示例中选择 `Graph` 查看详情信息，如下图。



- 可以设置监控时间段、实例、更新频率、周期、聚合方式和基线。
- 可以查看对应服务的所有监控指标。监控指标详情请参见监控指标说明。
- Graph 服务支持一系列基于图空间的监控指标。详情参见下文图空间监控。

图空间监控

Caution

使用图空间指标前，用户需要在 Graph 服务中，设置 `enable_space_level_metrics` 为 `true` 。具体操作，参见更新配置。

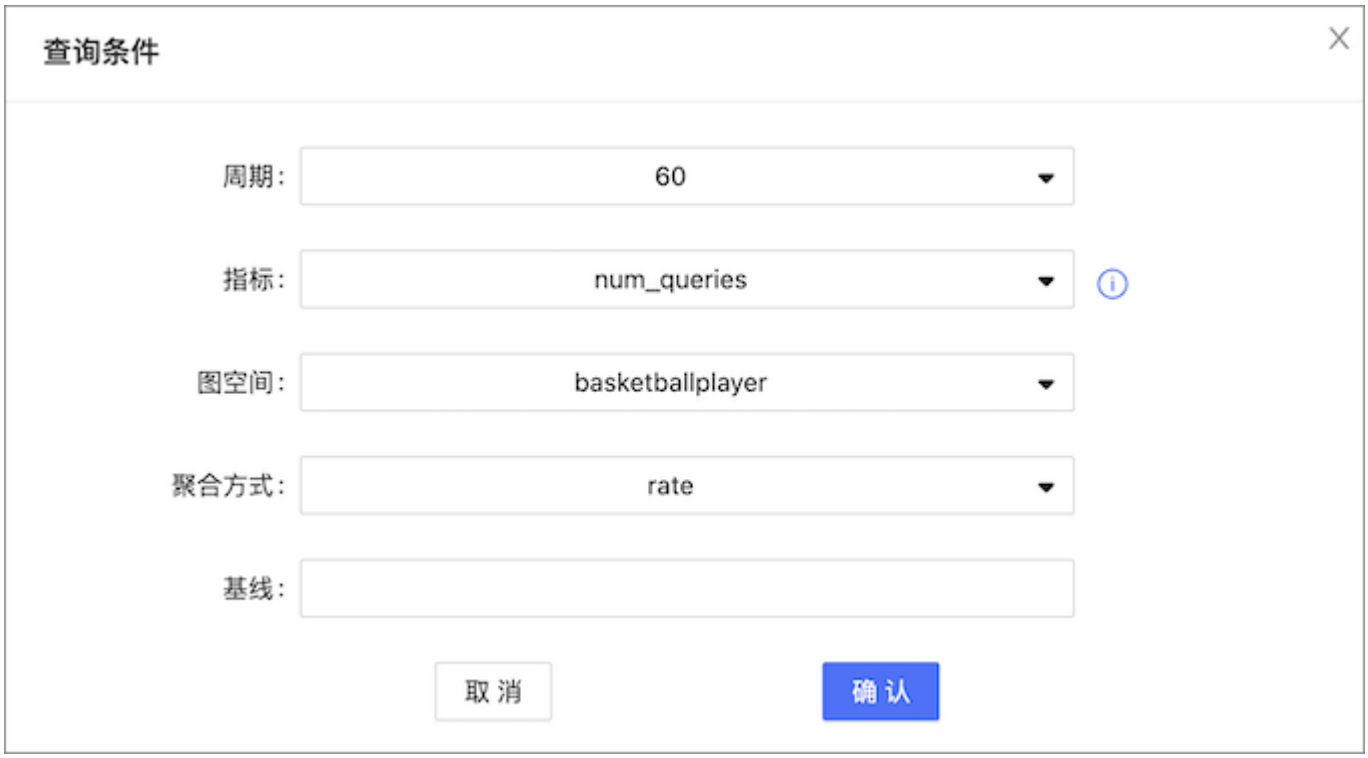
监控指标不兼容性

如果图空间的名称包括特殊字符，可能会有图空间监控指标数据不显示的问题。

服务监控页面还可以监控图空间级别指标。只有当图空间指标的行为被触发后，用户才能指定图空间查看对应的图空间指标的信息。图空间的监控指标记录不同图空间的数据。目前，只有 Graph 服务支持基于图空间的监控指标。

Dashboard 支持的图空间指标，参见图空间。

2022 vesoft Inc.



可视化监控大屏

Dashboard 企业版的可视化监控大屏帮助用户一目了然地把握集群的健康状态、服务及节点信息。

单击集群监控->监控大屏进入监控大屏页面。

2022 vesoft Inc.



大屏区域	显示信息
中间区域上方	1. 集群的健康度，为集群的健康打分。关于打分规则，参见下文 Note 说明。 2. 运行的节点信息和数量、集群中运行中的服务个数和异常服务的个数。 3. 当前时间的节点 CPU 和内存使用率。 4. 告警信息。系统按照告警消息的严重级别（emergency>critical>warning）展示最近触发的 5 条告警消息。更多信息，参见监控告警。
中间区域下方	在不同时间段 4 个 Graph 服务指标的监控信息。4 个指标分别是： 1. num_active_sessions 2. num_slow_queries 3. num_active_queries 4. num_query_errors
左侧区域	1. 集群的每秒查询率 QPS (Query Per Second)。 2. 在不同时间段两个 Storage 服务指标的监控信息。两个指标分别是 add_edges_latency_us、add_vertices_latency_us。
右侧区域	在不同时间段节点相关指标的信息，指标包括： 1. cpu_utilization 2. memory_utilization 3. load_1m 4. disk_readbytes 5. disk_writebytes

有关监控指标的具体信息，参见监控指标说明。

2022 vesoft Inc.

Note

集群的健康打分规则如下：

- 最高 100 分，最低 13 分。
- 当 100≥健康度≥80，分数为蓝色；当 80＞健康度≥60，分数为黄色；当健康度＜60，分数为红色。
- 算法：(1-异常服务数/总服务数)*100%。
- 除第一个出现的 `emergency` 级别的告警扣 40 分外，其余 `emergency` 级别及其他级别的告警每个扣 10 分。

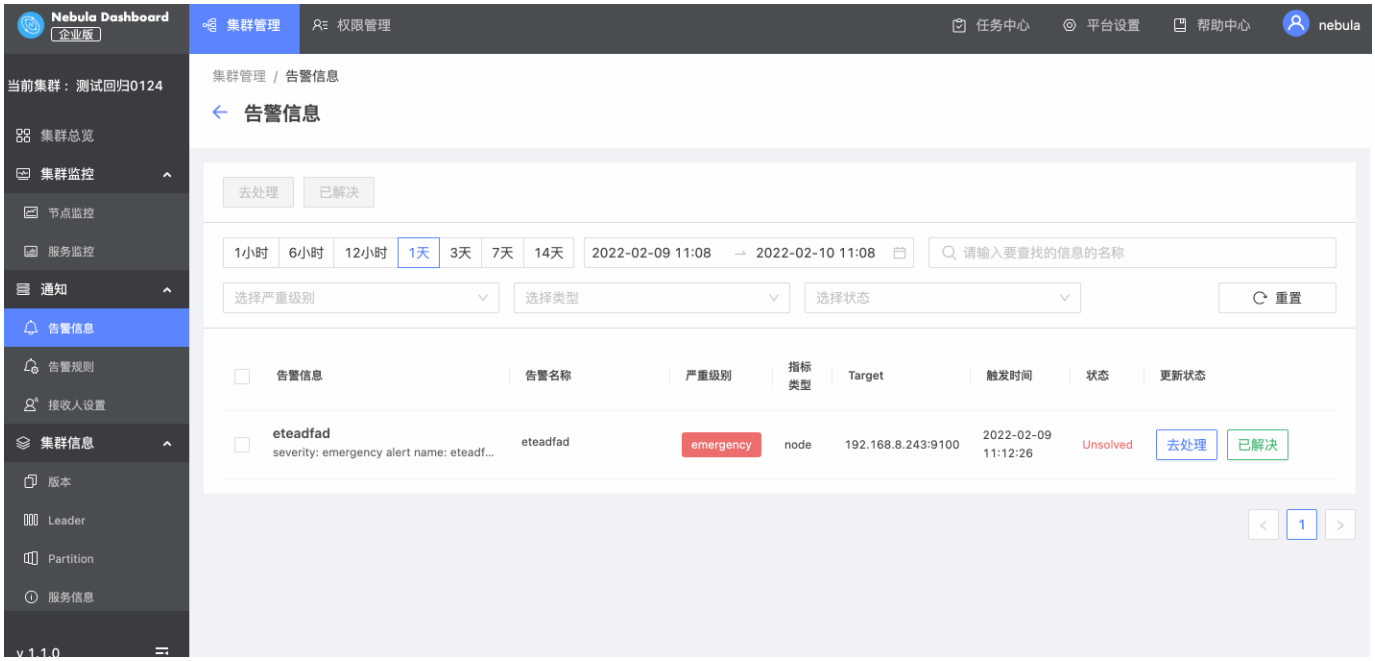
最后更新: June 20, 2023

2022 vesoft Inc.

16.5.3 监控告警

监控告警是监控 Nebula Graph 集群的监控指标进行报警的服务。用户可以查看告警信息，设置告警规则和告警接收方。

在 Dashboard 顶部导航栏，单击集群管理，在目标集群右侧单击详情，在左侧导航栏的通知下包含告警信息、告警规则和接收人设置。



告警信息

告警信息会以弹框的形式显示，有查看和静默按钮：

- 单击查看，会跳转至通知->告警信息页面，可以查看详细的告警信息。
- 单击静默，对应的告警规则在 2 小时内不会再次触发。

在告警信息页面可进行如下操作：

- 支持在搜索框中根据告警的名称搜索目标告警信息。
- 支持通过日期筛选告警信息。同时支持不同时间段的告警信息筛选，时间段包括 1 小时、6 小时、12 小时、1 天、3 天、7 天和 14 天。
- 支持根据信息的严重级别、类型、状态，筛选告警信息。用户可单击重置置空所有的筛选项。
- 用户可标记告警信息的处理状态，告警信息处理状态默认为 `unsolved`，用户可手动标记为 `处理中` 和 `已解决` 状态。

用户不可删除告警信息，但可在 `nebula-dashboard-ent/config/config.yaml` 文件中，通过修改 `messageStore` 参数的值来设置保存告警信息的天数。Nebula Dashboard 默认保存 90 天内的信息。具体操作，参见部署 Dashboard。

告警规则

对集群监控指标进行告警前，需要设置告警规则。用户可以自定义告警规则或者使用 Dashboard 内置的规则。

2022 vesoft Inc.

创建自定义规则

用户创建自定义规则的步骤如下：

1. 在 Dashboard 顶部导航栏中，单击集群管理，然后在目标集群右侧，单击详情。
2. 在集群管理页面左侧导航栏中，单击通知->告警规则。
3. 在告警规则页面，单击自定义规则，然后单击右侧的创建规则。
4. 设置创建规则的所需的信息。

a. 在基本信息配置页中，设置告警名称、级别和频率。

参数	说明
告警名称	为告警规则设置名称。名称只能包含小写字母、数字及连字符（`-`），且必须以小写字母或数字开头及结尾；最长 253 个字符。
严重级别	为告警规则设置级别：分为 `emergency` 、 `critical` 和 `warning` 三个级别。
警报频率	设置每隔多长时间告警一次，单位分（Min）。

b. 在条件配置页中，设置指标类型、规则和告警持续时间。

参数	说明
指标类型	包括节点和服务（graphd、storaged、metad）两种指标类型。
指标规则	为节点或者服务设置指标规则。详细介绍，参见监控指标。
持续时间	设置警报持续多长时间后触发告警。单位分（Min）。

c. 在消息设置配置页中，可看到系统自动生成的总规则及消息，然后单击提交。

> **Note**
>
> 不建议新手用户修改消息内容，否则创建规则时，可能会报错。

查看自定义规则

在自定义规则列表页，用户可进行以下操作：

- 搜索告警规则，并根据规则的严重级别、类型、指标和状态，筛选规则。
- 单击重置置空所有的筛选项。
- 打开和关闭自定义告警规则，开启状态显示 **active**，关闭状态显示 **disable**。

修改自定义规则

用户可在自定义规则列表中，选中目标规则，单击编辑图标  修改规则。

删除自定义规则

用户可在自定义规则列表中，选中目标规则，单击删除图标  删除规则。

内置规则

内置规则是 Dashboard 为用户提供的默认规则。用户可打开和关闭内置规则，开启状态显示 **active**，关闭状态显示 **disable**。

> **Note**
>
> 内置规则不可被修改和删除。

2022 vesoft Inc.

告警接收人

用户可为目标集群设置接收告警消息的邮箱，并且查看 Webhook 的地址及 Webhook 的开启状态。有关 Webhook 的介绍，参见全局设置。

1. 在 Dashboard 顶部导航栏中，单击集群管理，然后在目标集群右侧，单击详情。
2. 在集群管理页面左侧导航栏中，单击通知->接收人设置。
3. 在接收人设置页面：

- 单击 **Mail**，填入为目标集群接收告警消息的邮箱地址，然后单击添加。
- 单击 **Webhook**，查看 Webhook 的地址及 Webhook 的开启状态。

最后更新: June 20, 2023

2022 vesoft Inc.

16.5.4 集群信息

本文主要介绍 Dashboard 的集群信息，包括信息总览和集群诊断。用户可以通过信息总览查看 Nebula Graph 内核相关信息，通过集群诊断查看集群的诊断信息并根据诊断报告做出相应的调整。

在 Dashboard 顶部导航栏，单击集群管理，单击目标集群右侧详情，在左侧导航栏的集群监控下包含信息总览和集群诊断。

信息总览

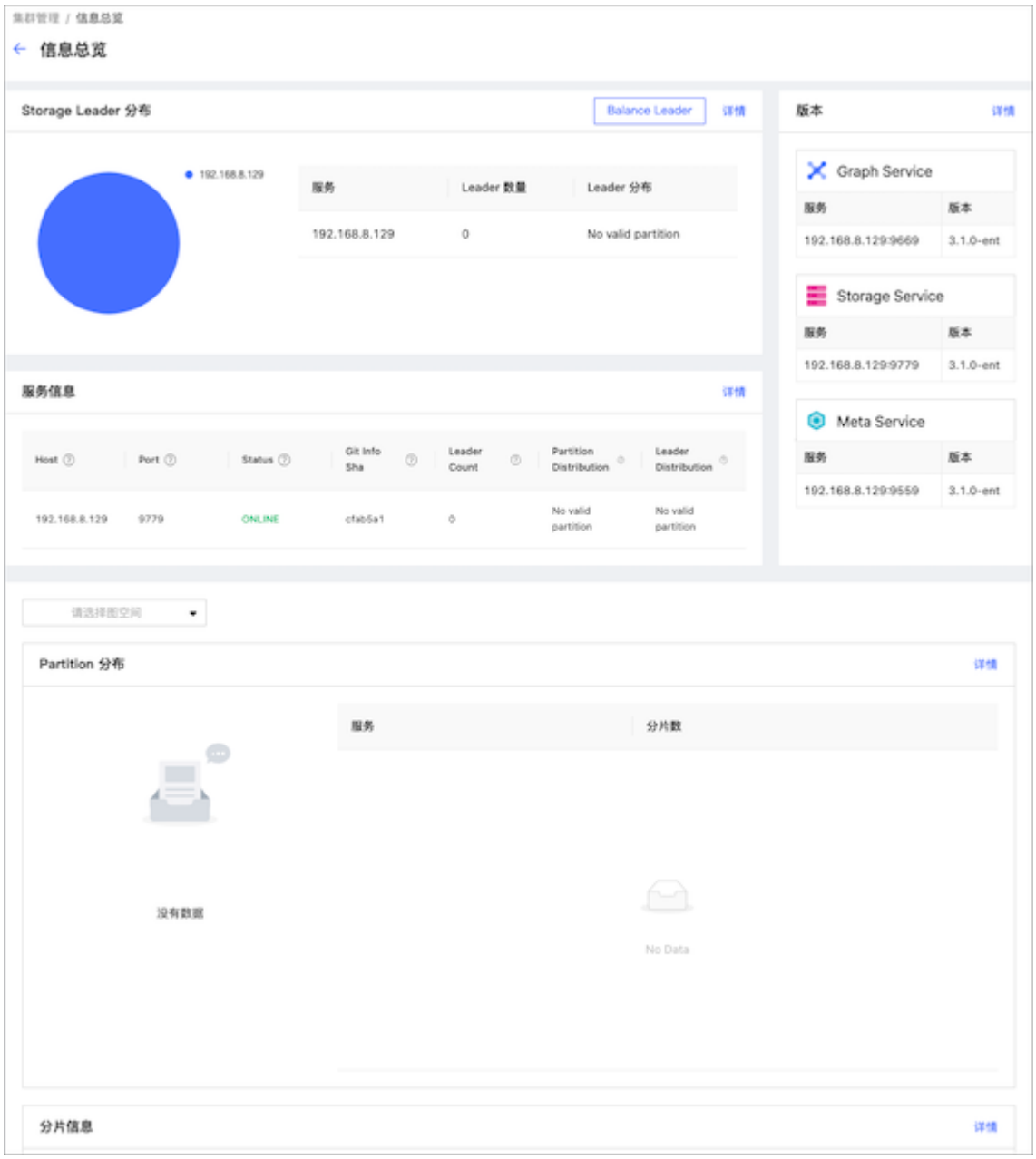
Note

在查看集群信息之前，用户需要选择任意一个在线的 Graph 服务地址，输入登录 Nebula Graph 的账号（非 Dashboard 登录账号）和对应密码。

Caution

用户需要保证 Nebula Graph 服务已经部署并启动。详细信息，参见 Nebula Graph 安装部署。

2022 vesoft Inc.



用户可以在信息总览页面查看 Nebula Graph 内核相关信息，包括 Storage 服务信息、Storage Leader 分布、Nebula Graph 各个服务的版本信息及所在节点信息、分片的分布情况及详细信息。

STORAGE LEADER 分布

显示 Leader 数量及 Leader 的分布。

- 单击右上角的 **Balance Leader** 按钮可以快速在 Nebula Graph 集群中均衡分布 Leader。关于 Leader 的详细信息，参见 Storage 服务。
- 单击右上角的详情，查看 Leader 分布的详细信息。

版本

显示 Nebula Graph 所有服务版本及服务地址。单击右上角的详情，查看更多信息。

2022 vesoft Inc.

服务信息

展示 Storage 服务的基本信息。参数说明如下：

参数	说明
`Host`	主机地址
`Port`	主机端口号
`Status`	主机状态
`Git Info Sha`	版本 Commit ID
`Leader Count`	Leader 总数
`Partition Distribution`	分片分布
`Leader Distribution`	Leader 分布

单击右上角的详情，查看更多信息。

PARTITION 分布

左上方选择指定图空间：

- 查看指定图空间的分片分布情况。显示所有 Storage 服务的 IP 地址、端口，及对应 Storage 服务中的分片数量。
- 单击 **Balance Data** 均衡分布当前图空间中的所有分片。
- 单击 **Balance Data Remove** 迁移指定的 Storage 服务中的所有分片至其他 Storage 服务中，操作前系统会先引导用户选择 Storage 服务所在的节点 IP。

单击右上角的详情，查看更多信息。

分片信息

显示分片信息。用户需要在左上角选择图空间，查看分片信息。参数说明如下：

参数	说明
`Partition ID`	分片序号。
`Leader`	分片的 leader 副本的 IP 地址和端口。
`Peers`	分片所有副本的 IP 地址和端口。
`Losts`	分片的故障副本的 IP 地址和端口。

单击右上角的详情，查看更多信息，通过右上角的输入框，输入分片 ID，筛选展示的数据。

集群诊断

用户可以在集群诊断页面，单击开始诊断自动生成最近一天的诊断报告，便于分析集群出现的问题。详情参见集群诊断。

最后更新: June 20, 2023

2022 vesoft Inc.

16.5.5 集群诊断

集群诊断是在指定的时间范围内，定位及分析当前集群出现的问题，并将诊断结果和集群相关的监控信息生成网页版的诊断报告。

功能介绍

- 通过诊断报告可以排查出当前集群出现的问题，并可以对问题进行解决。
- 快速了解集群内各个节点以及服务，Session，服务配置的基本情况。
- 根据诊断的报告信息，做出运维建议和集群预警。

诊断入口

1. 在 Dashboard 企业版顶部导航栏，单击集群管理。
2. 单击目标集群右侧详情。
3. 在左侧导航栏，单击集群信息->集群诊断。

创建诊断

1. 选择诊断的时间范围。支持自定义诊断时间和通过时间区间设置集群诊断的范围，包括：`1 小时` 、`6 小时` 、`12 小时` 、`1 天` 、`3 天` 、`7 天` 、`14 天` 。

> **Caution**
>
> 集群诊断是对集群历史数据的诊断，因此用户设置的诊断范围的结束时间不能超过当前时间。

2. 在集群诊断页面，单击开始诊断。



3. 当诊断状态由 **generating** 变为 **success** 时，即表示诊断已经完成。

查看诊断报告

在诊断列表中，单击目标诊断右侧的详情。

2022 vesoft Inc.

集群诊断报告中包含以下内容：

- 诊断结果
- 基本信息
- 负载
- Network
- Session
- 服务信息
- 配置信息

诊断结果

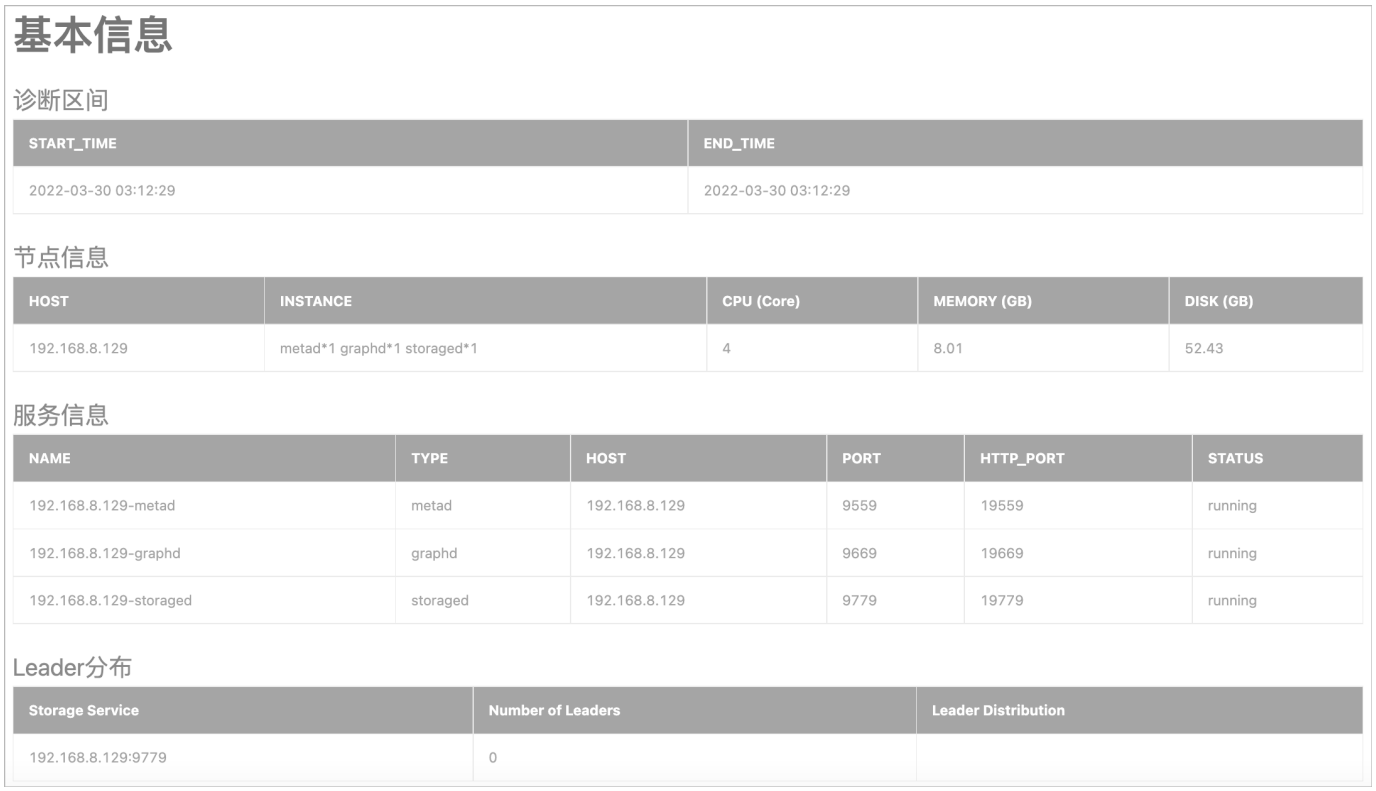
- 当以下参数出现异常时，诊断结果中会显示相应的信息，包括参数名、类型、严重程度、详情。

参数	说明
`num_queries_hit_memory_watermark`	执行的 nGQL 语句中达到内存水位值的数量。
`graphd_down`	Graph 服务停止运行。
`storaged_down`	Storage 服务停止运行。
`metad_down`	Meta 服务停止运行。
`node-exporter down`	用于收集节点系统数据的服务停止运行。

- 当未诊断到任何异常时，诊断结果中不显示诊断信息。

2022 vesoft Inc.

基本信息



基本信息

诊断区间

START_TIME	END_TIME
2022-03-30 03:12:29	2022-03-30 03:12:29

节点信息

HOST	INSTANCE	CPU (Core)	MEMORY (GB)	DISK (GB)
192.168.8.129	metad*1 graphd*1 storaged*1	4	8.01	52.43

服务信息

NAME	TYPE	HOST	PORT	HTTP_PORT	STATUS
192.168.8.129-metad	metad	192.168.8.129	9559	19559	running
192.168.8.129-graphd	graphd	192.168.8.129	9669	19669	running
192.168.8.129-storaged	storaged	192.168.8.129	9779	19779	running

Leader分布

Storage Service	Number of Leaders	Leader Distribution
192.168.8.129:9779	0	

- 诊断区间：显示生成报告的时间范围，包括开始时间和结束时间。
- 节点信息：显示集群中节点的数量、CPU、内存、磁盘等信息。

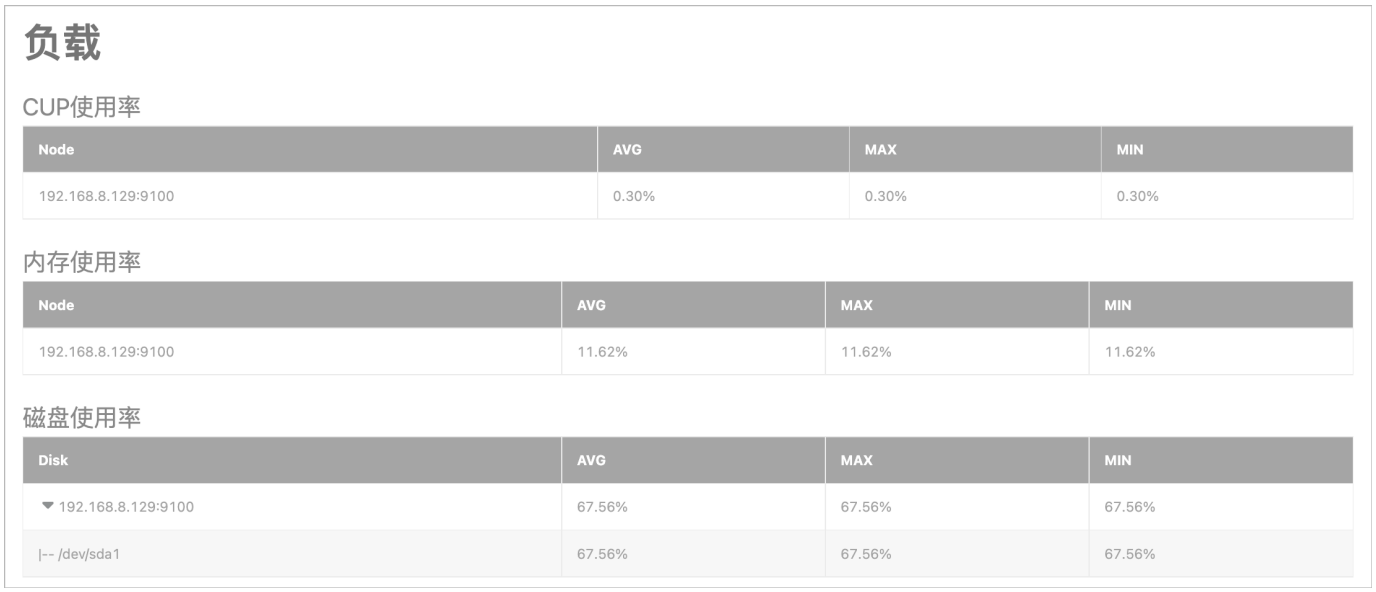
参数	说明
`HOST`	服务器的 IP 地址。
`INSTANCE`	该服务器部署的实例数量，例如：`metad*1 graphd*1 storaged*1` 。
`CPU`	CPU 核数，单位 Core。
`MEMORY`	表示服务器的内存大小，单位是 GB。
`DISK`	表示服务器磁盘大小，单位是 GB。

- 服务信息：显示 Nebula Graph 各个服务的类型、所在节点 IP、端口、HTTP 端口、运行状态。
- **Leader**分布：显示 Storage 服务中 Leader 分布情况。

参数	说明
`Storage Service`	显示 Storage 服务的访问地址。
`Number of Leaders`	显示 Storage 服务中的 Leader 总数 。
`Leader Distribution`	显示 Storage 服务中不同图空间的 Leader 分布数。

2022 vesoft Inc.

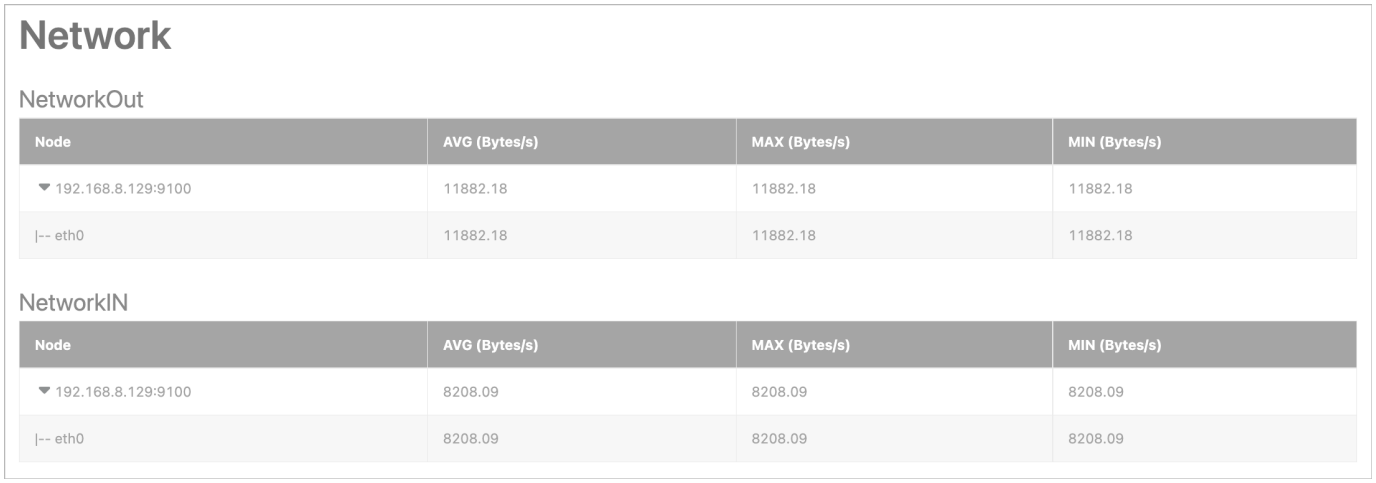
负载



显示集群所有服务器节点的负载信息，包括以下指标的平均值（AVG）、最大值（MAX）、最小值（MIN）：

- 内存使用率：显示节点内存使用率，单位是 %。
- **CPU**使用率：显示节点 CPU 使用率，单位是 %。
- 磁盘使用率：显示节点磁盘的总使用率，及节点中各个磁盘的使用率，单位是 %。

NETWORK

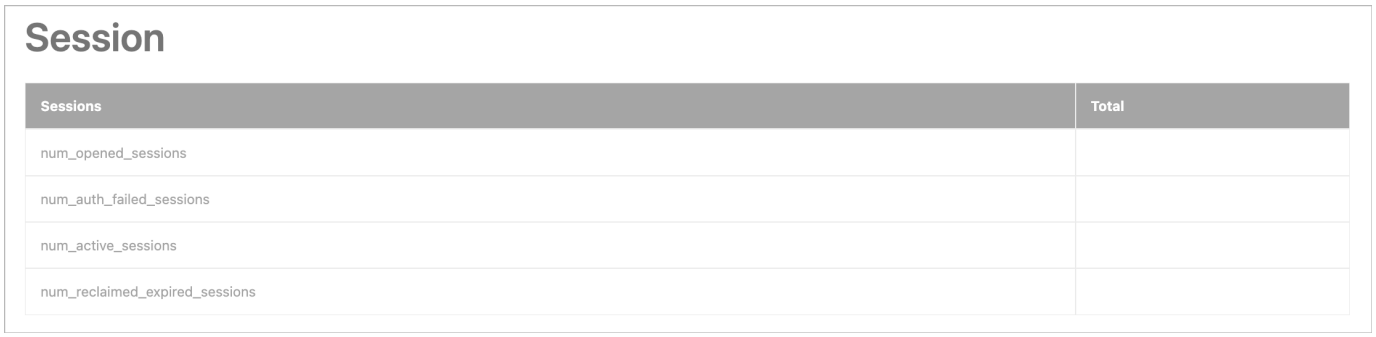


显示集群所有服务器节点的网络流量信息，包括以下指标的平均值（AVG）、最大值（MAX）、最小值（MIN）：

- **NetworkOut**：显示集群中各个服务器节点的网络流出速度的大小，及每个节点中各网卡的流出速度大小，单位是 Bytes/s。
- **NetworkIn**：显示集群中各个服务器节点的网络流入速度的大小，及每个节点中各网卡的流入速度大小，单位是 Bytes/s。

2022 vesoft Inc.

SESSION



显示集群中所有 Graph 服务的 Session 相关信息。

参数	说明
`num_opened_sessions`	服务端建立过的会话数量。
`num_auth_failed_sessions`	登录验证失败的会话数量。
`num_active_sessions`	当前活跃的会话数量。
`num_reclaimed_expired_sessions`	服务端主动回收的过期的会话数量。

2022 vesoft Inc.

服务信息

显示集群各服务稳定性相关的指标:

- **Graph**:

Graph

该表显示的是 Graph 服务稳定性相关指标。TOTAL为总数，ERROR为发生错误的个数，P75周期内响应耗时从小到大排列，顺序处于 75% 位置的分位数值。P95周期内响应耗时从小到大排列，顺序处于 95% 位置的分位数值。P99周期内响应耗时从小到大排列，顺序处于 99% 位置的分位数值。P999周期内响应耗时从小到大排列，顺序处于 99.9% 位置的分位数值。

METRIC_NAME	TOTAL	ERROR	P75	P95	P99	P999
slow_queries	0		-2147.483648s	-2147.483648s	-2147.483648s	-2147.483648s
num_killed_queries	0					
num_queries_hit_memory_watermark	0					
num_rpc_sent_to_metad	0	0				
query	0	0	-2147.483648s	-2147.483648s	-2147.483648s	-2147.483648s

参数	说明
`METRIC_NAME`	`query`： 查询次数。 `slow_queries`： 慢查询次数。 `num_killed_queries`： 被终止的查询数量。 `num_queries_hit_memory_watermark`： 执行的 nGQL 语句中达到内存水位值的数量。 `num_rpc_sent_to_metad`： Graphd 服务发给 Metad 服务的 RPC 请求数量。

- **Meta**:

Meta

该表显示的是 Meta 服务稳定性相关指标。TOTAL为总数，ERROR为发生错误的个数，P75周期内响应耗时从小到大排列，顺序处于 75% 位置的分位数值。P95周期内响应耗时从小到大排列，顺序处于 95% 位置的分位数值。P99周期内响应耗时从小到大排列，顺序处于 99% 位置的分位数值。P999周期内响应耗时从小到大排列，顺序处于 99.9% 位置的分位数值

METRIC_NAME	TOTAL	P75	P95	P99
heartbeat	0	-2147.483648s	-2147.483648s	-2147.483648s

参数	说明
`METRIC_NAME`	`heartbeat`： 心跳次数。

- **Storage**:

Storage

该表显示的是 Storage 服务 稳定性相关指标。TOTAL为总数，ERROR为发生错误的个数，P75周期内响应耗时从小到大排列，顺序处于 75% 位置的分位数值。P95周期内响应耗时从小到大排列，顺序处于 95% 位置的分位数值。P99周期内响应耗时从小到大排列，顺序处于 99% 位置的分位数值。P999周期内响应耗时从小到大排列，顺序处于 99.9% 位置的分位数值。

METRIC_NAME	TOTAL	ERROR	P75	P95	P99
delete_edges	0	0	-2147.483648s	-2147.483648s	-2147.483648s
delete_tags	0	0	-2147.483648s	-2147.483648s	-2147.483648s
num_rpc_sent_to_metad	0	0			
delete_vertices	0	0	-2147.483648s	-2147.483648s	-2147.483648s

参数	说明
`METRIC_NAME`	`delete_vertices`： 删除的点的数量。 `delete_edges`： 删除的边的数量。 `delete_tags`： 删除的 Tag 的数量。 `num_rpc_sent_to_metad`： Storaged 服务发给 Metad 服务的 RPC 请求数量。

2022 vesoft Inc.

其他参数说明如下：

参数	说明
`TOTAL`	该项监控执行的总次数。
`ERROR`	发生错误的个数。
`P75`	P75 周期内响应耗时从小到大排列，顺序处于 75% 位置的分位数值。
`P95`	P95 周期内响应耗时从小到大排列，顺序处于 95% 位置的分位数值。
`P99`	P99 周期内响应耗时从小到大排列，顺序处于 99% 位置的分位数值。
`P999`	P999 周期内响应耗时从小到大排列，顺序处于 99.9% 位置的分位数值。

配置信息

列出当前集群中 Graph、Meta、Storage 服务的所有配置信息。

关于各个服务的配置信息的详情，参见配置管理。

最后更新: June 20, 2023

2022 vesoft Inc.

16.5.6 集群操作

本文主要介绍 Dashboard 企业版的集群操作，包括节点管理、扩缩容、服务管理、更新配置、成员管理和版本升级。

节点管理

在节点管理页面中，展示所有的节点详情信息，包括节点名称、Host 及 SSH 用户名称、CPU 核等信息。

- 单击添加节点输入 Host 信息、SSH 端口号、用户、密码信息，选择 Nebula Graph 安装包，可快速添加节点。
- 在节点列表中，单击  按钮，查看对应节点服务名、服务类型、服务状态、运行路径等信息。
- 单击节点监控可快速跳转至节点监控页面，详情信息见集群监控。
- 单击服务管理可快速跳转至服务管理页面。
- 单击编辑节点可修改 SSH 端口号、用户名、密码。
- 当节点上无服务时，可以删除节点。

扩缩容

> **Enterpriseonly**
>
> 只有创建或导入企业版 Nebula Graph 集群时，用户才会在左侧导航栏中看到扩缩容功能。

- 在扩缩容页面，用户可以快速添加节点、批量导入节点，并对已存在的节点添加 **Graph** 服务及 **Storage** 服务。
- 单击重置按钮，可以恢复到初始状态。

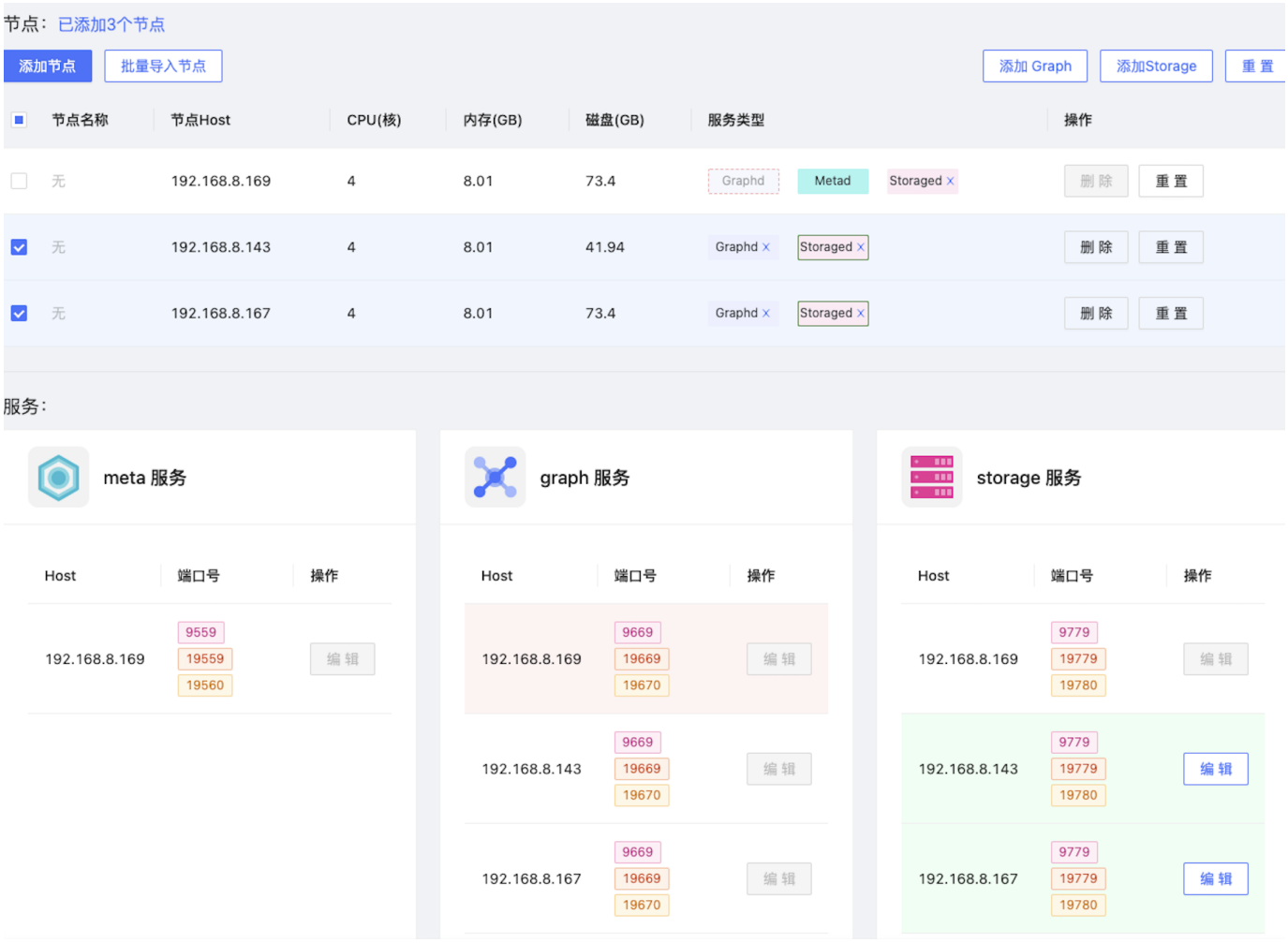
> **Caution**

- 目前仅支持对 graphd 和 storaged 进行扩缩容，不支持移除或添加 metad 服务。扩缩容集群时，建议提前备份数据，以便故障发生时回滚数据。关于扩缩容，详情参见如何扩缩容。
- 集群中每种类型的服务至少要有一个，且单个节点上仅能部署一个同类型服务。

在该示例中，新增了节点为 `192.168.8.143` 和 `192.168.8.167` 的 Storage 服务，删除了节点为 `192.168.8.169` 的 Graph 服务。虚线框并置灰服务名表示移除的服务，实线框表示添加的服务。

在下方的服务中，绿色表示即将新增的服务，红色表示即将移除的服务。可以修改新增服务的端口号和 HTTP 端口号信息。

2022 vesoft Inc.



服务管理

- 在服务管理页面，用户可以选择服务类型、服务状态、Host 对展示的数据进行筛选，快速选中一个或多个服务，一键启动/停止/重启服务。
- 单击  图标，可快速查看服务监控。

Danger

单击停止/重启，会立即中断进行中的任务，可能会导致数据不一致，请在业务低峰期执行该操作。

2022 vesoft Inc.

更新配置

更新配置页面可以修改 Storage 及 Graph 服务的配置文件，具体参数及说明见 Storage 服务配置和 Graph 服务配置。更新配置文件为批量操作，将会修改每一个 Storage/Graph 的配置文件。

- 单击保存后，在下一次服务重启后配置即生效。
- 单击保存并重启可直接重启服务使配置立即生效。

> **Danger**
>
> 单击保存并重启，会立即中断进行中的任务重启集群，可能会导致数据不一致，请在业务低峰期执行该操作。

成员管理

所有角色为 `admin` 的账户和角色为 `user` 的集群创建者可以为集群添加管理员。默认集群创建者为集群的 `owner` 角色，并显示在成员管理页面。添加的集群管理员为集群的 `operator` 角色。

关于账户的角色详情，参见权限管理。

添加集群管理员

集群管理员必需是用户创建的 Dashboard 企业版账户。关于如何创建账户，参见权限管理。

添加集群管理员步骤如下：

1. 在 Dashboard 企业版顶部导航栏，单击集群管理。
2. 在目标集群右侧，单击详情。
3. 在左侧导航栏，单击集群操作->成员管理。
4. 在成员管理页面，单击左上方的搜索框。
5. 在下拉列表中，选择需要添加的账户，然后单击右侧的添加。

其他操作

- 在成员管理页面右上方的搜索框中，搜索集群管理员。
- 单击  删除集群管理员。

版本升级

Dashboard 企业版支持升级现有 Nebula Graph 集群的版本。

> **Caution**

- 升级过程中会替换二进制文件，升级速度较快，但是仍会停机重启集群。
- 不支持自动回滚，升级失败时需要手动重新升级。
- 升级不支持暂停、取消。

2022 vesoft Inc.

Note

- 仅支持升级版本不低于 **3.0.0** 的 Nebula Graph 集群。
- 不支持跨大版本升级集群。
- 支持社区版升级为企业版（需要上传校验 License）或企业版升级为社区版。
- 支持升级为当前大版本内的任意小版本（包括比当前版本低的小版本）。

1. 在 Dashboard 企业版顶部导航栏，单击集群管理。
2. 在目标集群右侧，单击详情。
3. 在左侧导航栏，单击集群操作->版本升级。
4. 在版本升级页面确认当前**Nebula Graph**版本，选择待升级目标版本，单击下一步。

Note

如果没有找到合适的版本，单击包管理中心，下载或上传所需的版本安装包。详情参见安装包管理。

5. 进行升级检查，然后单击下一步。 集群在升级期间将会停机，升级完成后会自动重启服务。升级前您可以通过查看诊断报告了解集群线上运行情况，从而帮助您判断当前集群停机升级是否合适。
6. 再次确认升级信息，包括集群名称、当前**Nebula Graph**版本和待升级**Nebula Graph**版本，然后单击升级。 用户可以在在任务中心查看升级任务信息，类型为 `version update` 。

最后更新: June 20, 2023

2022 vesoft Inc.

16.5.7 使用 Nebula Dashboard 备份和恢复 Nebula Graph 数据

为了防止操作失误或者系统故障导致数据丢失，我们需要经常对数据进行备份，Nebula Graph 提供 Backup & Restore（BR）工具可帮助用户备份和恢复图数据。Nebula Dashboard 企业版集成了 BR 能力并提供简单的 UI，用户只需几步即可进行数据备份和恢复操作。本文介绍如何使用 Nebula Dashboard 备份和恢复 Nebula Graph 数据。

使用限制

- 目前仅支持将数据备份至兼容 S3 协议的云存储中（例如阿里云对象存储 OSS、MinIO、Ceph RGW 等），不支持备份至本地。

> **Note**
>
> 如需备份至本地，请参见备份与恢复。

- 目前仅支持数据的全量备份和全量恢复，不支持增量备份和增量恢复。
- 不支持图空间级别数据的备份和恢复。
- 备份数据只能被恢复至原集群，不能跨集群恢复数据。
- 不支持断点续传备份和恢复数据。
- 目前仅支持查看备份和恢复操作产生的日志。

前提条件

- 使用 Dashboard 创建 Nebula Graph 集群。
- 开通兼容 S3 协议的云存储服务并创建存储桶。具体操作，参见对应云存储服务的文档。

操作步骤

操作入口

1. 在 Nebula Dashboard 企业版顶部导航栏，单击集群管理。
2. 在目标集群右侧详情。
3. 在左侧导航栏，单击集群信息->备份恢复。

2022 vesoft Inc.

备份数据

用户通过创建备份文件的方式将备份的数据存至云存储服务中，操作如下：

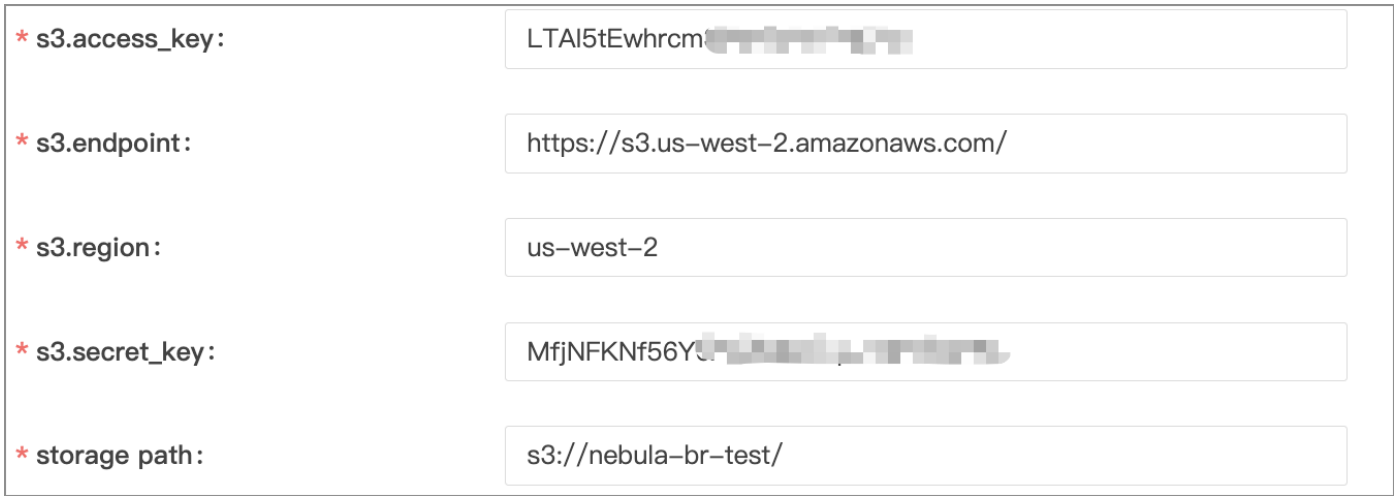
2022 vesoft Inc.

1. 在备份恢复页面，单击备份列表。
2. 在页面右上角，单击 **S3** 配置。
3. 填写对应的云存储的配置信息，并单击提交。

配置项	说明
s3.access_key	Access Key ID，用于标识用户，例如，`AKIAI44QH8DHBxxxx`。
s3.endpoint	表示云存储服务对外服务的访问域名 URL，例如 `https://s3.us-east-2.amazonaws.com`。不支持带 `bucket_name` 的 URL，例如 `https://{bucket_name}.s3.us-west-2.amazonaws.com`。
s3.region	数据中心所在的物理位置，例如 `us-east-1`。
s3.secret_key	Access Key Secret，用于加密签名字符串和用来验证签名字符串的密钥，例如 `je7MtGbClwBF/2Zp9Utk/h3yCoxxxx`。
storage path	数据的存储路径，路径必须以 `s3` 开头，例如 `s3://br-test/backup/`。

以下列举阿里云对象存储 OSS 和 Amazon S3 的示例配置信息：

- Amazon S3 的示例配置如下：



- 阿里云对象存储 OSS 的示例配置如下：



Caution

备份数据至阿里云 OSS 时，需将备份路径中的 `oss` 替换为 `s3`，例如将 `oss://nebula-br-test/` 改成 `s3://nebula-br-test/`。

2022 vesoft Inc.

4. 在页面右上角，单击创建备份。
5. 在创建备份页面，单击检查环境检查相关配置是否正常运行，然后单击提交。

   环境检查包括：

- Nebula Graph 需正常运行。
- 登录云存储服务的密钥未过期。
- 业务流量。仅检查用户业务的 QPS 是否为 0，当 QPS 不为 0 时，页面提示用户在业务流量低峰期进行备份操作。

> **Note**
>
> 如果 Nebula Graph 运行异常或者云存储服务的登录信息有误，用户将无法提交备份。

6. 在备份列表中查看创建的备份文件。

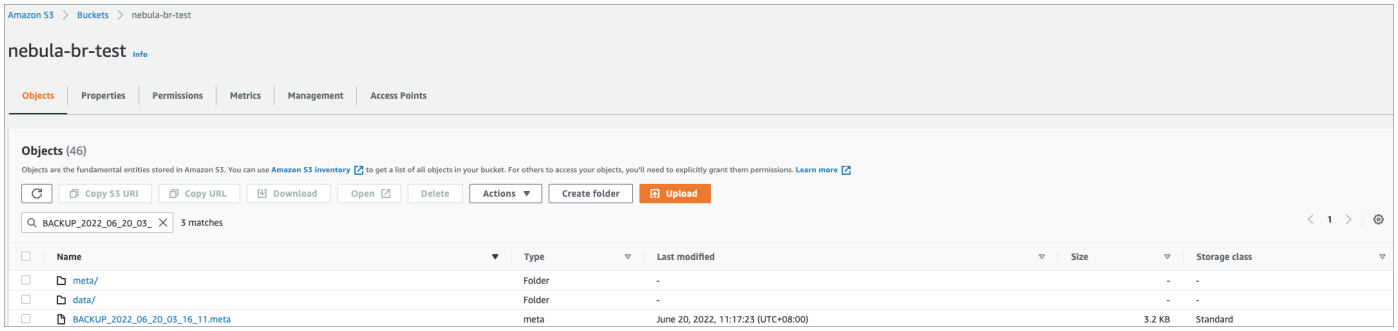


- 备份列表中默认显示 30 天内创建的备份，展示备份名称、时间、状态、图空间、存储方式、存储路径、相关操作。
- 备份状态包括 `running` 、 `success` 、 `failed` 。
- 相关操作包括恢复备份、查看恢复历史、查看备份操作产生的日志、删除备份。
- 用户可以通过设置筛选条件（创建时间和状态）或者搜索备份名称查看创建的备份。

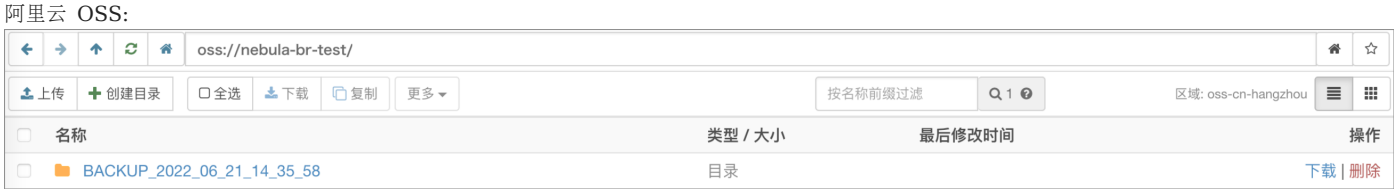
> **Note**
>
> 在上一次备份操作未完成前，不能再次执行备份操作。

7. 检查存储服务中是否有上述创建的备份文件。 成功创建的备份会被存储至上述用户设置的存储路径中，例如 `s3://nebula-br-test` 。

- Amazon S3:



- 阿里云 OSS:



2022 vesoft Inc.

Danger

请勿修改生成备份文件信息，包括名称、保存路径等，否则将无法恢复数据。

恢复数据

用户将备份数据存储至云存储服务后，可以恢复备份的数据至图数据库中。

Caution

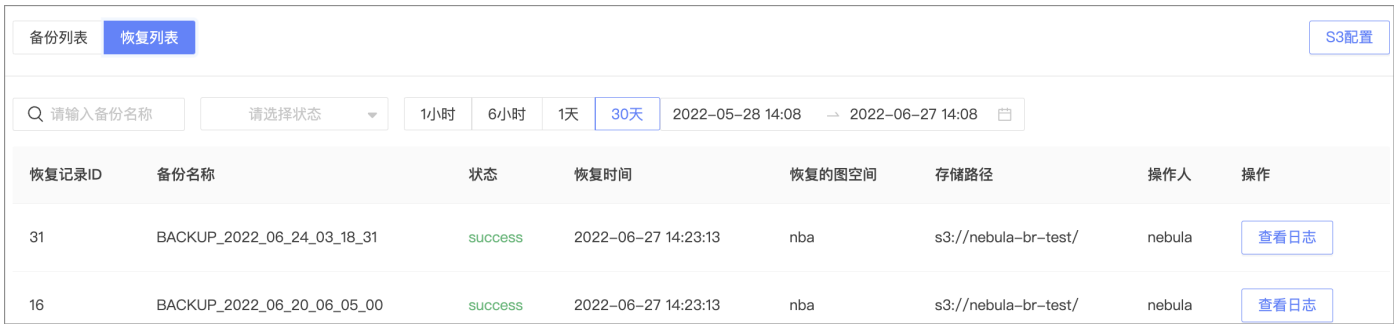
- 恢复数据前，请确认保存至云存储服务中的备份文件的名称、存储路径等无改动，否则数据恢复会失败。
- 恢复数据的过程中，集群中的所有数据会被移除，且替换为备份文件中的数据。
- 恢复过程为离线执行，用户在恢复数据的过程中无法进行其他操作。

按照以下步骤执行恢复操作：

1. 在备份恢复页面，单击备份列表。
2. 在目标备份文件右侧，单击恢复。
3. 单击检查环境，环境检查通过后，单击提交。

   环境检查包括：

   - Nebula Graph 需正常运行。
   - 登录云存储服务的密钥未过期。
   - 无业务流量的进入。
4. 在恢复列表页面，查看恢复记录。



- 恢复记录不可被删除。
- 恢复列表中默认显示 30 天内创建的恢复记录
- 恢复记录展示记录 ID、备份名称、恢复状态、恢复时间、恢复的图空间、存储路径、操作人、及查看恢复操作产生的日志。
- 恢复状态包括 `running` 、 `success` 、 `failed` 。

Note

在上一次恢复操作未完成前，不能再次执行恢复操作。

- 用户可以通过设置筛选条件（创建时间和状态）或者搜索备份名称查看恢复记录。

最后更新: June 20, 2023

2022 vesoft Inc.

16.5.8 操作记录

本文主要介绍 Dashboard 的操作记录。

在 Dashboard 顶部导航栏，单击集群管理，单击目标集群右侧详情，然后在左侧导航栏，单击操作记录进入操作记录页面。

在操作页面可以看到 1 小时、6 小时、12 小时、1 天、3 天、7 天和 14 天的操作记录，操作人、操作、时间、集群的信息。

最后更新: June 20, 2023

2022 vesoft Inc.

16.5.9 其他设置

本文主要介绍 Dashboard 的其他设置。

在 Dashboard 顶部导航栏，单击集群管理，单击目标集群右侧详情，然后在左侧导航栏，单击其他设置进入其他设置页面。

- 集群信息：显示集群名称、创建时间、创建用户信息、集群所有者名称。
- 解除绑定：解除绑定后从平台移除该集群的信息，对集群的服务和 Nebula 数据不做任何操作。

> **Note**
>
> 在执行该操作时，用户需要输入集群名称确定解除绑定。

- 删除集群：删除集群会停止集群服务同时保留 Nebula 数据，并解除与平台的信息绑定，请谨慎操作。

> **Note**
>
> 在执行该操作时，用户需要输入集群名称确定删除集群。

最后更新: June 20, 2023

2022 vesoft Inc.

16.6 任务中心

在 Nebula Dashboard 企业版中集群的创建和扩缩容需要一定的等待时间，用户可以通过任务中心功能感知此类操作的进度。任务中心展示正在进行的操作的进度及历史操作。

任务中心的任务指用户对集群做的操作。目前任务中心支持的任务类型包括集群创建和集群扩缩容。

在 Nebula Dashboard 企业版顶部导航栏中，单击任务中心可查看相关任务信息。

16.6.1 运行中的任务

在任务中心页面中，单击运行中任务后，用户可看到创建中或扩缩容中的集群的任务的进度。

- 单击任务名称可查看正在进行中的任务的 ID、节点名称、操作类型、触发时间、操作人。
- 单击任务信息查看任务详情。

16.6.2 历史任务

在任务中心页面的历史任务页签中，显示所有结束的任务。

- 用户可通过任务状态、类型、触发时间筛序历史任务。
- 在目标历史任务右侧，单击任务信息查看历史任务的详情；单击任务日志查看任务执行日志。

16.6.3 删除任务

目前不可取消正在进行中的任务，也不可删除历史任务。

最后更新: June 20, 2023

2022 vesoft Inc.

16.7 权限管理

用户可以使用不同账户登录 Nebula Dashboard 企业版，不同账户拥有不同权限。本文介绍账户类型、账户角色及角色权限等。

16.7.1 账户类型

Dashboard 企业版的初始化账户名为 `nebula` 和密码为 `nebula`。使用初始化账户名和密码登录 Dashboard 企业版后，即可创建不同类型的账户：LDAP 账户和普通账户。

LDAP 账户

Dashboard 企业版通过对接 LDAP（Lightweight Directory Access Protocol），使得用户可以使用企业内部账户体系登录 Dashboard 企业版。

使用 LDAP 账户前，需要进行 LDAP 配置：

1. 在 Dashboard 企业版的安装路径中，如 `nebula-graph-dashboard-ent/nebula-dashboard-ent/config`，找到配置文件 `config.yaml`。
2. 在 `config.yaml` 中，添加以下内容。

```
ldap:
  server: ldap://127.0.0.1
  bindDN: cn=admin,dc=vesoft,dc=com
  bindPassword: kenshin
  baseDN: dc=vesoft,dc=com
  userFilter: "&(objectClass=*)"
  emailKey: mail
```

参数	说明
server	LDAP 服务器地址。
bindDN	LDAP 服务器登录用户名。
bindPassword	LDAP 服务器登录密码。
baseDN	查询用户数据的路径。
userFilter	查询条件。
emailKey	LDAP 数据库存放邮箱信息的字段名。

3. 重启 Dashboard 企业版生效配置。

成功配置 LDAP 后，使用 Dashboard 企业版的初始化账户名 `nebula` 和密码 `nebula` 登录 Dashboard 企业版，然后可创建 LDAP 账户。关于如何创建账户，参见创建账户。

普通账户

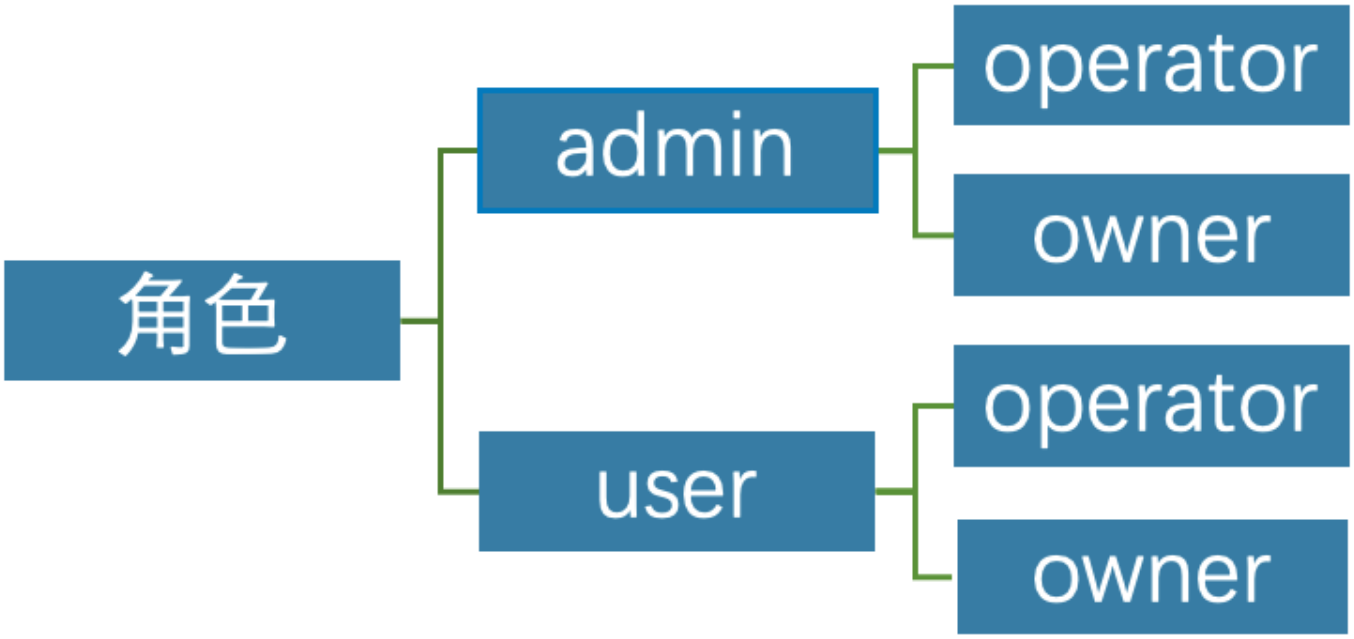
除了 LDAP 账户，其他账户均为普通账户。

16.7.2 账户角色

用户可以为账户设置不同的角色，不同的角色拥有的权限不同。Dashboard 企业版中的账户角色有两种类型：系统角色（`admin` 和 `user`）和集群角色（`owner` 和 `operator`）。

系统角色和集群角色的关系及介绍如下。

2022 vesoft Inc.



系统角色：

角色名	权限	说明
admin	1. 创建账户。 2. 修改已有账户的角色。 3. 进行平台设置、系统级别告警设置。 4. 删除账户。	1. 系统中可以有多个 `admin` 角色，即系统管理员角色。 2. 默认 `admin` 是所有集群的 `operator` ，即可以看到所有集群。 3. 默认显示在集群成员列表中。 `owner` 无法将 `admin` 成员移除，除非将 `admin` 成员转换成 `user` ，系统将从集群中自动移除 `admin` 成员。
user	1. 系统维度只有只读权限。 2. `admin` 新建一个 `user` 角色的账户后，若没有分配对应的集群，则该账户无法查看任何集群。 3. 可以创建集群，成为该集群的 `owner` 。	1. 普通用户角色。 2. 系统中可以有多个 `user` 。

集群角色：

角色名	权限	说明
`operator`	1. 扩缩容集群。 2. 设置集群告警。 3. 进行节点管理。 4. 进行服务管理。	1. 集群操作者角色。 2. 一个集群可以有多个 `operator` 。
`owner`	1. 拥有 `operator` 的所有权限。 2. 解绑和删除集群。 3. 添加和移除 `operator` 账户。 4. 转移 `owner` 角色。	1. 集群所有者角色。 2. 一个集群只能有一个 `owner` 。

2022 vesoft Inc.

16.7.3 创建账户

角色为 `admin` 的账户可以创建账户。步骤如下：

1. 在 Dashboard 企业版页面的顶部，单击权限管理，然后单击创建。
2. 选择创建方式并填入信息，然后单击确定。

- 邀请：设置被邀请人的企业邮箱和角色。被邀请人单击邮件中的 **Accept** 按钮激活账户后，然后单击登录自动跳转至 Dashboard 企业版登录页面。被邀请人使用企业邮箱账号和邮箱密码即可登录 Dashboard 企业版。

> **Note**
>
> 选择邀请方式添加账户时，确保企业 LDAP 服务器中已添加被邀请人的邮箱。

- 创建账户：设置新账户的登录名称、密码、角色。关于角色的详情，参见账户角色。

16.7.4 查看账户

已创建的账户会显示在权限管理页面中。

- 用户可以查看账户的用户名、账号类型、角色、关联集群、创建时间。
- 账号类型：包括 **platform** 和 **ldap**。**platform** 为普通账户；**ldap** 为 LDAP 账户。
- 角色：显示角色名，包括 **admin** 和 **user**。关于角色的详情，参见上文。
- 关联集群：显示该账户可以操作的所有集群。如果集群为该账户所创建，相关集群附带 `owner` 标签。
- 用户可以在搜索框中搜索账户，也可以通过选择关联集群来筛选账户。

16.7.5 其他操作

- 在权限管理页面的操作列下，单击  编辑账户信息。
- 在权限管理页面的操作列下，单击  删除账户。

最后更新: June 20, 2023

2022 vesoft Inc.

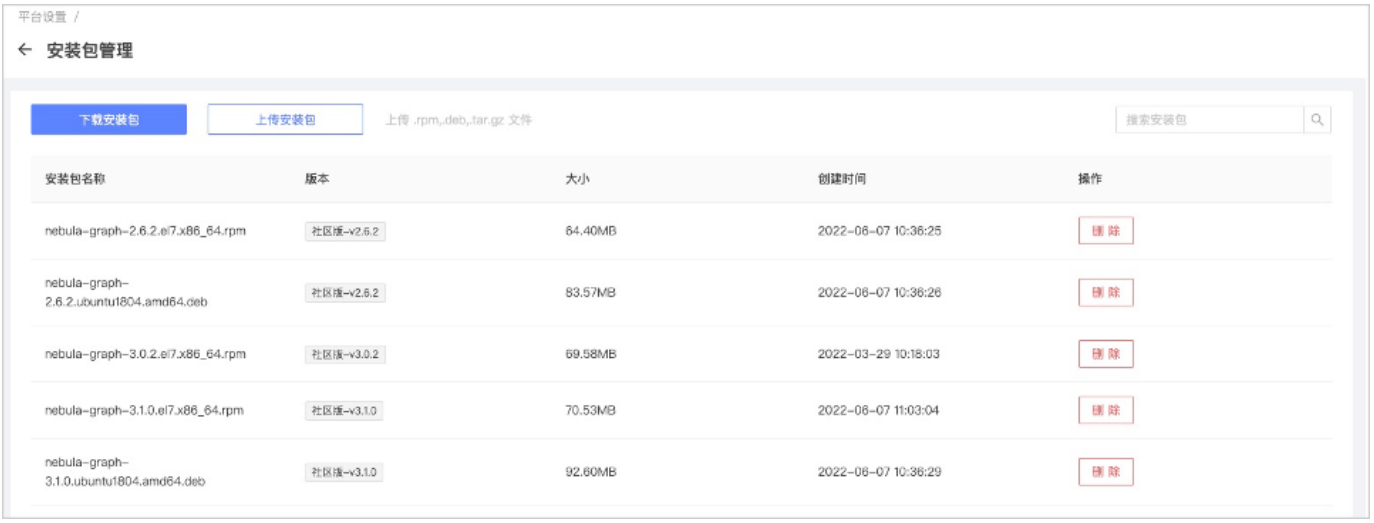
16.8 安装包管理

Nebula Dashboard 企业版支持管理 Nebula Graph 安装包，可以下载社区版的各个版本，或者手动上传安装包。

16.8.1 注意事项

- 只有 `admin` 角色的用户可以管理安装包。
- 不支持下载企业版安装包。请发送邮件至 **inquiry@vesoft.com** 申请企业版安装包，然后手动上传。

16.8.2 查看安装包



1. 在 Dashboard 企业版顶部导航栏，单击平台设置。
2. 在左侧导航栏单击安装包管理，右侧会显示已有安装包的列表，展示安装包名称、版本、大小和创建时间。

   用户可以通过右上角的搜索框进行筛选。

16.8.3 下载安装包

1. 在 Dashboard 企业版顶部导航栏，单击平台设置。
2. 在左侧导航栏单击安装包管理。
3. 单击下载安装包，选择要下载的安装包。说明如下：

   Note

   下载已存在的安装包，会提示覆盖原有安装包。

   - 版本：支持 v2.5 以上的稳定版本。
   - 平台：支持 CentOS 7/8、Ubuntu 1604/1804/2004。
   - 安装包类型：支持 RPM、DEB、tar.gz。
4. 单击下载。

   下载任务可以在任务中心查看任务信息，类型为 `package download`。如果任务状态为 `success`，返回安装包管理页面，即可查看到新下载的安装包。

2022 vesoft Inc.

16.8.4 上传安装包

如果在下载安装包的列表里没有需要的安装包，用户可以手动上传安装包，例如上传企业版安装包。

1. 在 Dashboard 企业版顶部导航栏，单击平台设置。
2. 在左侧导航栏单击安装包管理。
3. 单击上传安装包，选择本地要上传的安装包。类型可以是 RPM、DEB 或 tar.gz。查看界面上方的上传进度，等待上传完成。

上传任务可以在任务中心查看任务信息，类型为 `package upload` 。如果任务状态为 `success` ，返回安装包管理页面，即可查看到新上传的安装包。

16.8.5 删除安装包

1. 在 Dashboard 企业版顶部导航栏，单击平台设置。
2. 在左侧导航栏单击安装包管理，右侧会显示已有安装包的列表，找到需要删除的安装包。
3. 在目标安装包的操作列，单击删除并确认。

16.8.6 常见问题

上传安装包时报错 **`Request Entity Too Large`** 如何处理?

如果用户使用 Nginx 做反向代理，默认限制上传文件大小为 1 MB，请在 `nginx.conf` 中的 `http{}` 段中添加 `client_max_body_size 200m;` ，即允许上传最大 200 MB 的文件。

最后更新: June 20, 2023

2022 vesoft Inc.

16.9 全局配置

本文介绍使用 Dashboard 的全局配置，包括平台设置、帮助中心、个人信息。

16.9.1 平台设置

在 Dashboard 顶部导航栏中，单击平台设置，用户可以进行系统设置、通知设置和其他设置。

系统设置

在平台设置页面的左侧导航栏中，单击系统设置，可修改页面标题、Logo 图片和封面图。

通知设置

- 邮箱：Dashboard 支持通过邮箱发送和接收所有集群的告警消息。

  在平台设置页面的左侧导航栏中，单击通知设置->邮箱:

- 用户需设置以下参数来发送告警消息。

参数	说明
SMTP 服务器地址	用户邮箱对应的 SMTP 服务器地址。
端口号	邮箱对应的 SMTP 服务器的端口号。
Use SSL	勾选后启用 SSL 进行数据的加密传输。
SMTP 用户名	SMTP 服务器的用户名。
SMTP 密码	SMTP 服务器的密码。
发件人邮箱	发送告警消息的邮箱地址。

- 用户需设置接收人来接收告警消息。

参数	说明
接收人	接收告警消息的邮箱地址。该邮箱将接收 Dashboard 上所有集群的告警消息。

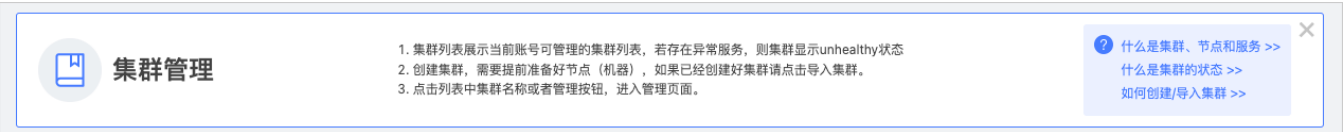
- **Webhook**：Dashboard 支持用户配置 Webhook 将所有集群的告警消息接入第三方项目中。

  在平台设置页面的左侧导航栏中，单击通知设置->**Webhook**，填入接收告警消息的 **Webhook URL**。用户也可在页面右上方开启和关闭 Webhook 功能。

其他设置

在平台设置页面的左侧导航栏中，单击其他设置，可进行以下操作：

- 更换显示语言，目前仅支持中文和英文。
- 开启和关闭帮助提示，提示示例如下：



2022 vesoft Inc.

16.9.2 帮助中心

在 Dashboard 顶部导航栏中，单击帮助中心查看 Dashboard 操作相关的文档、Nebula 官网及论坛等页面。

16.9.3 用户信息

在 Dashboard 顶部导航栏右上角，悬浮鼠标至账户名，例如 **nebula**：

- 单击个人信息查看用户信息及修改登录密码。

  Note

  如果是 LDAP 账户，则不支持修改登录密码。关于账户详情，参见权限管理。

- 单击登出退出当前账号。

最后更新: June 20, 2023

2022 vesoft Inc.

16.10 监控指标说明

本文介绍 Dashboard 中展示的 Nebula Graph 监控指标。

16.10.1 机器

Note

- 以下机器指标适用于 Linux 操作系统。
- 磁盘容量和流量的默认单位为字节（Byte），页面显示时单位会随着数据量级而变化，例如流量低于 1 KB/s 时单位为 Bytes/s。
- 对于高于 v1.0.2 版本的社区版 Dashboard，机器的 Buff 和 Cache 所占的内存没有被计算在内存使用率中。

CPU

参数	说明
`cpu_utilization`	CPU 已使用百分比
`cpu_idle`	CPU 空闲百分比
`cpu_wait`	等待 IO 操作的 CPU 百分比
`cpu_user`	用户空间（非 Nebula Graph 图空间）占用的 CPU 百分比
`cpu_system`	内核空间（非 Nebula Graph 内核空间）占用的 CPU 百分比

内存

参数	说明
`memory_utilization`	内存已使用百分比
`memory_used`	已使用内存（不包括缓存）。
`memory_free`	空闲内存

负载

参数	说明
`load_1m`	最近 1 分钟系统平均负载
`load_5m`	最近 5 分钟系统平均负载
`load_15m`	最近 15 分钟系统平均负载

2022 vesoft Inc.

磁盘

参数	说明
disk_used	磁盘已使用存储空间
disk_free	磁盘剩余存储空间
disk_readbytes	磁盘每秒读取的字节数
disk_writebytes	磁盘每秒写入的字节数
disk_readiops	磁盘每秒的读请求数量
disk_writeiops	磁盘每秒的写请求数量
inode_utilization	inode 已使用百分比

流量

参数	说明
network_in_rate	网卡每秒接收的字节数
network_out_rate	网卡每秒发送的字节数
network_in_errs	网卡每秒接收错误的字节数
network_out_errs	网卡每秒发送错误的字节数
network_in_packets	网卡每秒接收的数据包数量
network_out_packets	网卡每秒发送的数据包数量

16.10.2 服务

周期

指标统计的时间范围，当前支持 5 秒、60 秒、600 秒和 3600 秒，分别表示最近 5 秒、最近 1 分钟、最近 10 分钟和最近 1 小时。

聚合方式

参数	说明
rate	周期内平均每秒操作的速率
sum	周期内操作的总和
avg	周期内响应平均耗时
P75	周期内响应耗时从小到大排列，顺序处于 75%位置的分位数
P95	周期内响应耗时从小到大排列，顺序处于 95%位置的分位数
P99	周期内响应耗时从小到大排列，顺序处于 99%位置的分位数
P999	周期内响应耗时从小到大排列，顺序处于 99.9%位置的分位数

Note

以下为 Dashboard 获取内核的全量指标，但 Dashboard 仅展示重要的指标。

2022 vesoft Inc.

Graph

参数	说明
num_active_queries	当前正在执行的查询数。
num_active_sessions	当前活跃的会话数量。
num_aggregate_executors	聚合（Aggregate）算子的执行次数。
num_auth_failed_sessions_bad_username_password	因用户名密码错误导验证失败的会话数量。
num_auth_failed_sessions_out_of_max_allowed	因为超过 FLAG_OUT_OF_MAX_ALLOWED_CONNECTIONS 参数导致的验证登录的失败的 session 数量。
num_auth_failed_sessions	登录验证失败的会话数量。
num_indexscan_executors	索引扫描（IndexScan）算子的执行次数。
num_killed_queries	被终止的查询数量。
num_opened_sessions	服务端建立过的会话数量。
num_queries	查询次数。
num_query_errors_leader_changes	因查询错误而导致的 Leader 变更的次数。
num_query_errors	查询错误次数。
num_reclaimed_expired_sessions	服务端主动回收的过期的会话数量。
num_rpc_sent_to_metad_failed	Graphd 服务发给 Metad 的 RPC 请求失败的数量。
num_rpc_sent_to_metad	Graphd 服务发给 Metad 服务的 RPC 请求数量。
num_rpc_sent_to_storaged_failed	Graphd 服务发给 Storaged 服务的 RPC 请求失败的数量。
num_rpc_sent_to_storaged	Graphd 服务发给 Storaged 服务的 RPC 请求数量。
num_sentences	Graphd 服务接收的语句数。
num_slow_queries	慢查询次数。
num_sort_executors	排序（Sort）算子的执行次数。
optimizer_latency_us	优化器阶段延迟时间。
query_latency_us	查询平均延迟时间。
slow_query_latency_us	慢查询平均延迟时间。
num_queries_hit_memory_watermark	达到内存水位线的语句的数量。

2022 vesoft Inc.

Meta

参数	说明
commit_log_latency_us	Raft 协议中 Commit 日志的延迟时间。
commit_snapshot_latency_us	Raft 协议中 Commit 快照的延迟时间。
heartbeat_latency_us	心跳延迟时间。
num_heartbeats	心跳次数。
num_raft_votes	Raft 协议中投票的次数。
transfer_leader_latency_us	Raft 协议中转移 Leader 的延迟时间。
num_agent_heartbeats	AgentHBProcessor 心跳次数。
agent_heartbeat_latency_us	AgentHBProcessor 延迟时间。
replicate_log_latency_us	Raft 复制日志至大多数节点的延迟。
num_send_snapshot	Raft 发送快照至其他节点的次数。
append_log_latency_us	Raft 复制日志到单个节点的延迟。
append_wal_latency_us	Raft 写入单条 WAL 的延迟。
num_grant_votes	Raft 投票给其他节点的次数。
num_start_elect	Raft 发起投票的次数。

2022 vesoft Inc.

Storage

2022 vesoft Inc.

参数	说明
add_edges_atomic_latency_us	添加边单次延迟。
add_edges_latency_us	添加边的平均延迟时间。
add_vertices_latency_us	添加点的平均延迟时间。
commit_log_latency_us	Raft 协议中 Commit 日志的延迟时间。
commit_snapshot_latency_us	Raft 协议中 Commit 快照的延迟时间。
delete_edges_latency_us	删除边的平均延迟时间。
delete_vertices_latency_us	删除点的平均延迟时间。
get_neighbors_latency_us	查询邻居平均延迟时间。
num_get_prop	GetPropProcessor 执行的次数。
num_get_neighbors_errors	GetNeighborsProcessor 执行出错的次数。
get_prop_latency_us	GetPropProcessor 执行的延迟时间。
num_edges_deleted	删除的边数量。
num_edges_inserted	插入的边数量。
num_raft_votes	Raft 协议中投票的次数。
num_rpc_sent_to_metad_failed	Storaged 服务发给 Metad 服务的 RPC 请求失败的数量。
num_rpc_sent_to_metad	Storaged 服务发给 Metad 服务的 RPC 请求数量。
num_tags_deleted	删除的 Tag 数量。
num_vertices_deleted	删除的点数量。
num_vertices_inserted	插入的点数量。
transfer_leader_latency_us	Raft 协议中转移 Leader 的延迟时间。
lookup_latency_us	LookupProcessor 执行的延迟时间。
num_lookup_errors	LookupProcessor 执行时出错的次数。
num_scan_vertex	ScanVertexProcessor 执行的次数。
num_scan_vertex_errors	ScanVertexProcessor 执行时出错的次数。
update_edge_latency_us	UpdateEdgeProcessor 执行的延迟时间。
num_update_vertex	UpdateVertexProcessor 执行的次数。
num_update_vertex_errors	UpdateVertexProcessor 执行时出错的次数。
kv_get_latency_us	Getprocessor 的延迟时间。
kv_put_latency_us	PutProcessor 的延迟时间。
kv_remove_latency_us	RemoveProcessor 的延迟时间。
num_kv_get_errors	GetProcessor 执行出错次数。
num_kv_get	GetProcessor 执行次数。
num_kv_put_errors	PutProcessor 执行出错次数。
num_kv_put	PutProcessor 执行次数。
num_kv_remove_errors	RemoveProcessor 执行出错次数。

2022 vesoft Inc.

参数	说明
num_kv_remove	RemoveProcessor 执行次数。
forward_tranx_latency_us	传输平均延迟时间。
scan_edge_latency_us	ScanEdgeProcessor 执行的延迟时间。
num_scan_edge_errors	ScanEdgeProcessor 执行时出错的次数。
num_scan_edge	ScanEdgeProcessor 执行的次数。
scan_vertex_latency_us	ScanVertexProcessor 执行的延迟时间。
num_add_edges	添加边的次数。
num_add_edges_errors	添加边时出错的次数。
num_add_vertices	添加点的次数。
num_start_elect	Raft 发起投票的次数
num_add_vertices_errors	添加点时出错的次数。
num_delete_vertices_errors	删除点时出错的次数。
append_log_latency_us	Raft 复制日志到单个节点的延迟。
num_grant_votes	Raft 投票给其他节点的次数。
replicate_log_latency_us	Raft 复制日志到大多数节点的延迟。
num_delete_tags	删除 Tag 的次数。
num_delete_tags_errors	删除 Tag 时出错的次数。
num_delete_edges	删除边的次数。
num_delete_edges_errors	删除边时出错的次数。
num_send_snapshot	发送快照的次数。
update_vertex_latency_us	UpdateVertexProcessor 执行的延迟时间。
append_wal_latency_us	Raft 写入单条 WAL 的延迟。
num_update_edge	UpdateEdgeProcessor 执行的次数。
delete_tags_latency_us	删除 Tag 的平均延迟时间。
num_update_edge_errors	UpdateEdgeProcessor 执行时出错的次数。
num_get_neighbors	GetNeighborsProcessor 执行的次数。
num_get_prop_errors	GetPropProcessor 执行时出错的次数。
num_delete_vertices	删除点的次数。
num_lookup	LookupProcessor 执行的次数。

2022 vesoft Inc.

图空间级别监控指标

参数	说明
num_active_queries	当前正在执行的查询数。
num_queries	查询次数。
num_sentences	Graphd 服务接收的语句数。
optimizer_latency_us	优化器阶段延迟时间。
query_latency_us	查询平均延迟时间。
num_slow_queries	慢查询次数。
num_query_errors	查询报错语句数量。
num_query_errors_leader_changes	因查询错误而导致的 Leader 变更的次数。
num_killed_queries	被终止的查询数量。
num_aggregate_executors	聚合（Aggregate）算子的执行次数。
num_sort_executors	排序（Sort）算子的执行次数。
num_indexscan_executors	索引扫描（IndexScan）算子的执行次数。
num_oom_queries	导致内存耗尽的语句数量。
num_auth_failed_sessions_bad_username_password	因用户名密码错误导验证失败的会话数量。
num_auth_failed_sessions	登录验证失败的会话数量。
num_opened_sessions	服务端建立过的会话数量。
num_queries_hit_memory_watermark	达到内存水位线的语句的数量。
num_reclaimed_expired_sessions	服务端主动回收的过期的会话数量。
num_rpc_sent_to_metad_failed	Graphd 服务发给 Metad 的 RPC 请求失败的数量。
num_rpc_sent_to_metad	Graphd 服务发给 Metad 服务的 RPC 请求数量。
num_rpc_sent_to_storaged_failed	Graphd 服务发给 Storaged 服务的 RPC 请求失败的数量。
num_rpc_sent_to_storaged	Graphd 服务发给 Storaged 服务的 RPC 请求数量。
slow_query_latency_us	慢查询平均延迟时间。

最后更新: June 20, 2023

2022 vesoft Inc.

16.11 常见问题 FAQ

本文列出了使用 Dashboard 时可能遇到的常见问题，用户可以使用文档中心或者浏览器的搜索功能查找相应问题。

16.11.1 什么是集群、节点和服务?

- 集群：Dashboard 集群特指由多个 Nebula Graph 服务所在的节点组成的一组系统。
- 节点：特指承载运行 Nebula Graph 服务的物理或虚拟机。
- 服务：特指 Nebula 服务，包括 Metad，Storaged 和 Graphd。

16.11.2 什么是集群的状态?

集群的状态有以下几种：

- installing：集群正在创建中，过程大约需要等待 3～10 分钟。
- healthy：集群中所有服务正常运行。
- unhealthy：集群服务中存在非正常运行的服务。

16.11.3 为什么要授权节点?

由于集群的管理操作需要获取对应节点的 SSH 信息，因此 Dashboard 在执行操作前，需要先拥有最少可执行权限的 SSH 账号密码信息。

16.11.4 什么是扩缩容?

Nebula Graph 是分布式图数据库，可以支持运行时动态扩展和缩减服务。也可以通过 Dashboard 可视化的动态扩展或缩减 Storage 和 Graph 服务（不可以扩缩容 Metad）。

16.11.5 为什么不能操作 Meta 服务?

Meta 服务主要存储 Nebula Graph 数据库中的元数据。一旦 Meta 服务出现问题，整个集群会面临瘫痪风险。且 Meta 处理的数据量较少，一般不会出现扩缩容场景，因此我们直接在 Dashboard 中禁用了操作 Meta，防止出现用户误操作导致集群不可用的情况。

16.11.6 扩缩容之后对数据有什么影响?

- 扩容 Storage：Dashboard 会在指定的机器上创建并运行 Storage 服务，对已有数据不会造成影响，可以在左侧导航栏集群信息->信息总览中的 `Storage Leader 分布` 和 `Partition 分布` 区域，根据自身需求选择进行 `Balance Leader` 或者 `Balance Data` 操作。
- 缩容 Storage：如果 Storage 上有分片的话，无法直接缩容。用户需要先在左侧导航栏集群信息->信息总览中的 `Partition 分布` 区域单击 `Balance Data Remove`，将指定的 Storage 服务中的所有分片迁移至其他 Storage 服务中。
- 扩缩容 Graph 对数据不会有影响。

16.11.7 为什么无法启动 Dashboard（企业版）?

- 确保 License 已拷贝至 Dashboard 目录中，且执行了 `sudo ./dashboard.service start all` 命令。
- 确保 License 未过期。

用户也可以在 Dashboard 目录中执行 `cat logs/webserver.log` 查看各个模块的启动信息。如果满足以上条件，仍无法启动 Dashboard，请前往 Nebula Graph 官方论坛咨询。

16.11.8 是否可以手动添加 Nebula Graph 安装包?

Dashboard 支持手动添加安装包，可以点击如何获取 Nebula Graph 下载所需系统和版本的 RPM 包或 DEB 包，添加到 `nebula-dashboard-ent/download/nebula-graph` 路径下。在创建、扩容集群时，可选择添加的包进行部署。

2022 vesoft Inc.

16.11.9 为什么导入集群时，提示 “ssh 连接失败”?

如果在为节点授权时，节点**Host** 显示为 `127.0.0.1` 且 Dashboard 和 Nebula Graph 没有被部署在同一台机器上，系统会提示 “ssh 连接失败”。用户需要在 Nebula Graph 所有服务的配置文件中，修改各服务的 Host IP 为真实的机器 IP。更多信息，参见配置管理。

如果用户导入的是 Docker 部署的集群，也会显示 “ssh 连接失败”。Dashboard 不支持导入使用 Docker 部署的集群。

最后更新: June 20, 2023

2022 vesoft Inc.

17. Nebula Explorer

17.1 什么是 Nebula Explorer

Nebula Explorer （简称 Explorer）是一款可以通过 Web 访问的可视化图探索工具，搭配 Nebula Graph 内核使用，用于与图数据进行可视化交互。即使没有图数据操作经验，用户也可以快速成为图专家。

> **Enterpriseonly**
>
> Explorer 仅在企业版提供。

> **Note**
>
> 用户可以在线试用 Explorer 部分功能。

17.1.1 适用场景

如果有以下任一需求，都可以使用 Explorer：

- 从复杂关系中快速查找友邻关系、分析可疑目标，需要以可视化的方式展示图数据。
- 针对大规模数据集，需要以可视化的方式对数据进行过滤、分析和图探索。

17.1.2 产品优点

- 易于安装：简单步骤即可完成部署。
- 易于使用：可视化构造查询场景，无需构思 nGQL 语句，轻松实现图探索。
- 灵活性强：支持通过 VID、Tag、Subgraph 等方式查询数据。
- 探索拓展：支持对多个点进行拓展操作、查询多个点的共同邻居、查询起点到终点之间的路径等操作。
- 多样展示：支持修改画布中点的颜色和图标，突出关键节点；也支持使用不同布局模式展示数据。
- 数据存储：支持保存和导入画布数据。

17.1.3 身份验证

Nebula Graph 默认不启用身份验证，一般情况下用户可以使用 `root` 账号和任意密码登录 Explorer。

Nebula Graph 启用了身份验证后，用户只能使用指定的账号和密码登录 Explorer。

关于 Nebula Graph 的身份验证功能，参考 Nebula Graph 用户手册。

17.1.4 视频

- Nebula Explore Demo Show（2 分 54 秒）

最后更新: June 20, 2023

2022 vesoft Inc.

17.2 安装与登录

17.2.1 部署 Explorer

本文介绍如何在本地通过 RPM 和 TAR 包部署 Explorer。

Nebula Graph 版本支持

Nebula Graph 版本	Explorer 版本
3.1.0 ~ 3.1.0	3.1.0
3.0.0 ~ 3.1.0	3.0.0
2.5.x ~ 3.0.0	2.2.0
2.6.x	2.1.0
2.5.x	2.0.0

前提条件

在部署 Explorer 之前，用户需要确认以下信息：

- Nebula Graph 服务已经部署并启动。详细信息参考 Nebula Graph 安装部署。
- 以下端口未被使用。

端口号	说明
7002	Explorer 提供的 web 服务

Caution

Explorer 默认使用的端口号为 7002，用户可以在安装目录下的 `conf/app.conf` 文件中修改 `httpport`，并重启服务。

- 使用的 Linux 发行版为 CentOS。
- 准备 License。

Enterpriseonly

License 仅在企业版提供，申请 License 需填写 Nebula Explorer 试用申请。

RPM 部署

安装

1. 根据需要下载 RPM 包，建议选择最新版本。

Enterpriseonly

用户可以在线申请试用 Explorer 企业版；如需购买，通过邮箱(inquiry@vesoft.com)联系销售人员。点击定价查看更多。

2. 使用 `sudo rpm -i <rpm>` 命令安装 RPM 包。

2022 vesoft Inc.

例如，安装 Explorer 需要运行以下命令，默认安装路径为 `/usr/local/nebula-explorer`：

```
sudo rpm -i nebula-explorer-<version>.x86_64.rpm
```

也可以使用 `--prefix` 选项安装到指定路径：

```
sudo rpm -i nebula-explorer-<version>.x86_64.rpm --prefix=<path>
```

3. 拷贝 License 至安装路径下。

```
cp -r <license> <explorer_path>
```

例如：

```
cp -r nebula.license /usr/local/nebula-explorer
```

4. 执行以下命令启动服务。

```
systemctl start nebula-explorer
```

启停服务

支持使用 systemctl 服务控制项目启停。

```
systemctl status nebula-explorer #查看服务状态
systemctl stop nebula-explorer #停止服务
systemctl start nebula-explorer #启动服务
```

也可以在安装目录下使用以下命令，手动启动或停止服务。

```
cd ./scripts/rpm
bash ./start.sh #启动服务
bash ./stop.sh #停止服务
```

卸载

使用以下的命令卸载 Explorer。

```
sudo rpm -e nebula-explorer-<version>.x86_64
```

使用 **DEB** 包部署

安装

1. 下载 DEB 包。

> **...terpriseonly**
>
> 用户可以在线申请试用 Explorer 企业版；如需购买，通过邮箱(inquiry@vesoft.com)联系销售人员。点击定价查看更多。

2. 使用 `sudo dpkg -i <package_name>` 命令安装 DEB 包。

例如，安装 Explorer 需要运行以下命令，默认安装路径为 `/usr/local/nebula-explorer`：

```
sudo dpkg -i nebula-explorer-3.1.0.x86_64.deb
```

> **Note**
>
> 使用 DEB 包安装 Explorer 时不支持自定义安装路径。

3. 拷贝 License 至 `nebula-explorer` 目录下。

```
sudo cp -r <license> <explorer_path>
```


2022 vesoft Inc.

例如：

```
sudo cp -r nebula.license /usr/local/nebula-explorer
```

4. 执行以下命令启动服务。

```
sudo systemctl start nebula-explorer.service
```

或者在 `nebula-explorer/lib` 目录下执行以下命令，启动服务。

```
sudo bash ./start.sh
```

查看服务状态

```
sudo systemctl status nebula-explorer.service
```

停止服务

```
sudo systemctl stop nebula-explorer.service
```

卸载

使用以下的命令卸载 Explorer。

```
sudo dpkg -r nebula-explorer
```

TAR 包部署

安装及部署

1. 根据需要下载 TAR 包，建议选择最新版本。

> **Enterpriseonly**
>
> Explorer 仅在企业版提供，点击定价查看更多。

2. 使用 `tar -xvf` 解压 tar 包。

```
tar -xvf nebula-explorer-<version>.tar.gz
```

3. 拷贝 License 至 `nebula-explorer` 目录下。

```
cp -r <license> <explorer_path>
```

例如：

```
cp -r nebula.license /usr/local/nebula-explorer
```

4. 进入 `nebula-explorer` 文件夹，启动 Explorer。

```
cd nebula-explorer
nohup ./nebula-explorer-server &
```

停止服务

用户可以采用 `kill pid` 的方式来关停服务：

```
kill $(lsof -t -i :7002)
```

后续操作

启动成功后，在浏览器地址栏输入 `http://<ip_address>:7002`。

2022 vesoft Inc.

在浏览器窗口中看到以下登录界面表示已经成功部署并启动了 Explorer。



Note

首次登录 Nebula Explorer 的时候，页面显示最终用户许可协议的内容，请仔细阅读并单击同意。

进入 Explorer 登录界面后，用户需要连接 Nebula Graph。详细信息，参考连接数据库。

最后更新: June 20, 2023

2022 vesoft Inc.

17.2.2 连接数据库

在成功启动 Explorer 后，用户需要配置连接 Nebula Graph。本文主要描述 Explorer 如何连接 Nebula Graph 数据库。

前提条件

在连接 Nebula Graph 数据库前，用户需要确认以下信息：

- 已经安装部署了 Explorer。详细信息，参见部署 Explorer。
- Nebula Graph 的 Graph 服务本机 IP 地址以及服务所用端口。默认端口为 `9669`。
- Nebula Graph 数据库登录账号信息，包括用户名和密码。

连接数据库

按以下步骤连接 Nebula Graph 数据库：

1. 在 Explorer 的配置数据库页面上，输入以下信息：

- **Host**：填写 NebulaGraph 的 Graph 服务本机 IP 地址及端口。格式为 `ip:port`。如果端口未修改，则使用默认端口 `9669`。

> **Note**
>
> - 即使 NebulaGraph 数据库与 Explorer 部署在同一台机器上，用户也必须在 **Host** 字段填写这台机器的本机 IP 地址，而不是 `127.0.0.1` 或者 `localhost`。
> - 在新的标签页连接另一个 NebulaGraph 数据库时，会覆盖旧标签页的会话。如果需要同时登录多个 NebulaGraph 数据库，可以用不同的浏览器或者无痕模式。

- 用户名和密码：根据 NebulaGraph 的身份验证设置填写登录账号和密码。
- 如果未启用身份验证，可以填写默认用户名 `root` 和任意密码。
- 如果已启用身份验证，但是未创建账号信息，用户只能以 GOD 角色登录，必须填写 `root` 及对应的密码 `nebula`。
- 如果已启用身份验证，同时又创建了不同的用户并分配了角色，不同角色的用户使用自己的账号和密码登录。

2. 完成设置后，点击登录按钮。

> **Note**
>
> 一次连接会话持续 30 分钟。如果超过 30 分钟没有操作，会话即断开，用户需要重新登录数据库。

断开连接

在页面右上角，选择 图标 > 清空连接。

如果浏览器上显示配置数据库页面，表示 Explorer 已经成功断开了与 Nebula Graph 数据库的连接。

最后更新: June 20, 2023

2022 vesoft Inc.

17.2.3 Nebula Explorer 企业版 License

License 是为用户提供的软件授权证书。用户在部署 Nebula Explorer 企业版时，需要设置 License 才能正常启动 Nebula Explorer 企业版。本文介绍 Nebula Explorer 企业版的 License 相关信息。

注意事项

- 没有设置 License 时，Nebula Explorer 企业版无法启动。
- 请勿修改 License，否则会导致 License 失效。
- License 快过期时，请及时发送邮件至**inquiry@vesoft.com**申请续期。
- License 的过期缓冲为 14 天：
- 过期前 30 天和过期当天，服务启动时会打印日志进行提醒。
- 过期后仍可继续使用 14 天。
- 过期 14 天后，服务无法启动，并会打印日志进行提醒。

获取方式

用户需发送邮件至**inquiry@vesoft.com**申请 Nebula Explorer 企业版 License。

> **Note**
>
> 填写 Nebula Explorer 试用申请可获取 30 天免费试用的 License。

Nebula Explorer 企业版 License 说明

Nebula Explorer 企业版 License 是一个名为 `nebula.license` 的文件，内容示例如下：

```
----------License Content Start----------
{
  "vendor": "vesoft",
  "organization": "doc",
  "issuedDate": "2022-06-06T16:00:00.000Z",
  "expirationDate": "2023-05-31T15:59:59.000Z",
  "product": "nebula_graph_explorer",
  "version": ">=3.2.0",
  "licenseType": "enterprise",
  "gracePeriod": 14,
  "clusterCode": "BAIAEAiAQAAG"
}
----------License Content End----------

----------License Key Start----------
Rrjip5cxxxxxxxxxxxx5zKoQ==
----------License Key End----------
```


2022 vesoft Inc.

License 文件包含生效时间、过期时间等信息。说明如下。

参数	说明
`vendor`	发放渠道。
`organization`	用户名称。
`issuedDate`	License 生效时间。
`expirationDate`	License 过期时间。
`product`	产品类型。Nebula Explorer 企业版的产品类型为 `nebula_graph_explorer` 。
`version`	版本支持的信息。
`licenseType`	License 类型（预留参数）。包括 `enterprise` 、`samll_bussiness` 、`pro` 、`individual` 。
`gracePeriod`	证书过期后可继续使用服务的缓冲时间（单位天），超过缓冲期后停止服务。试用版的 License 过期后无缓冲期，默认值为 0。
`clusterCode`	用户的硬件信息，也是集群的唯一标识码。试用版的 License 中无此参数。

使用方式

关于如何使用 Nebula Explorer，参见部署 Nebula Explorer。

续期操作

按照以下操作续期 Nebula Explorer License。

1. 发送邮件至 `inquiry@vesoft.com` 申请新的 Nebula Explorer License 文件 `nebula.license` 。
2. 在 Nebula Explorer 的安装目录下（例如，`/usr/local/nebula-explorer`），使用新的 License 文件替换旧的 License 文件。

> **Note**
>
> Nebula Explorer license 过期后，用户无法登录 Explorer。为了保证业务正常运行，请及时更新 License。

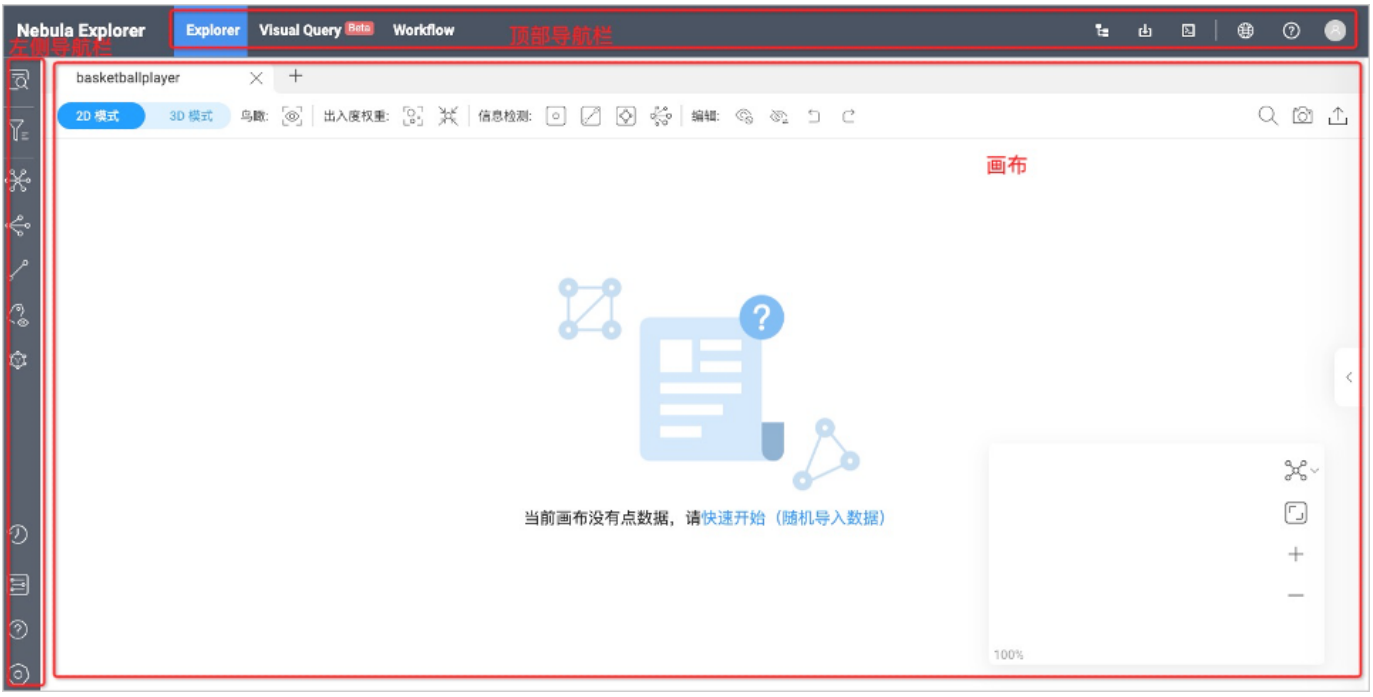
最后更新: June 20, 2023

2022 vesoft Inc.

17.3 页面概览

本文介绍 Nebula Explorer 的基本组成，帮忙用户了解 Nebula Explorer 的各功能。

Nebula Explorer 由顶部导航栏、左侧导航栏、和画布三大模块组成。



17.3.1 顶部导航栏

图标/元素	说明
Explorer	可视化探索及分析数据。详情参见开始探索、节点筛选、探索拓展、图计算。
Visual Query	可视化构造场景进行数据查询。详情参见可视化查询。
Workflow	可视化构造自定义工作流进行复杂图计算。详情参见工作流简介。
	对 Nebula Graph 图空间进行管理。详情参见创建 Schema。
	将数据批量导入 Nebula Graph。详情参见导入数据。
	对 Nebula Graph 内的数据使用 nGQL 进行查询操作。详情参见控制台。
	选择 Nebula Explorer 页面的语言，支持中文和英文。
	帮助页面，指导和帮忙用户使用 Nebula Graph。
	显示内核版本并可以断开与 Nebula Explorer 的连接。

2022 vesoft Inc.

17.3.2 左侧导航栏

> **Note**
>
> 登录 Explorer 后，用户需选择并单击一个目标图空间，然后才可解锁左侧导航栏的查询及分析功能。详情参见选择图空间。

用户可以通过单击 Explorer 页面左侧的图标进行图数据导入、图数据分析与探索等操作。左侧导航栏的图标及描述如下：

图标	说明
	输入 VID、Tag 和子图等信息后，匹配的数据会显示到画布上。更多信息，参见查询方式。
	对画布中显示的点进行过滤。更多信息，参见节点筛选。
	选择页面上的节点并进行自定义拓展，包括方向、步数、过滤条件等。更多信息，参见探索拓展。
	选择页面上至少两个点并查看它们的共同邻居。更多信息，参见探索拓展。
	查询起点到终点之间的所有路径、最短路径和非循环路径。更多信息，参见探索拓展。
	选择是否显画布中的点或边的属性值。更多信息，参见探索拓展。
	基于画布中的点边进行图计算。更多信息，参见图计算
	查看历史快照信息。更多信息，参见画布快照。
	查看所有图空间，单击图空间可创建相应图空间的画布。更多信息，参见选择图空间。
	查看 Explorer 操作文档、论坛内容等。
	查看用户名、版本、快捷键，限制查询最大返回数量。

17.3.3 画布

> **Note**
>
> 登录 Explorer 后，用户需选择并单击一个目标图空间，才可进入画布页面。详情参见选择图空间。

图数据可视化地展示在画布中。Explorer 的画布由以下部分组成：

- 顶部页签
- 可视化模式
- 数据存储
- 搜索框
- 多布局
- 缩略图
- 数据概览

更多信息，参见画布操作。

2022 vesoft Inc.

最后更新: June 20, 2023

2022 vesoft Inc.

17.4 数据库管理

17.4.1 Schema 管理

Explorer 支持界面化或者通过控制台命令方式管理 Schema。

在 Explorer 页面的右上方，单击  进入 Schema 创建页面。

Explorer 的 Schema 功能同 Studio。详情参见创建 Schema。

最后更新: June 20, 2023

2022 vesoft Inc.

17.4.2 数据导入

Explorer 支持界面化地将数据导入至 Nebula Graph 中。

在 Explorer 页面右上方，单击  进入导入数据页面。

Explorer 的导入数据功能同 Studio。详情参见导入数据。

最后更新: June 20, 2023

2022 vesoft Inc.

17.4.3 控制台

Explorer 的控制台功能允许用户手动输入 nGQL 语句，并可视化地呈现查询结果。

在 Explorer 页面的右上方，单击  进入控制台页面。

Explorer 控制台功能同 Studio。详情，参见控制台。

最后更新: June 20, 2023

2022 vesoft Inc.

17.5 图探索

17.5.1 选择图空间

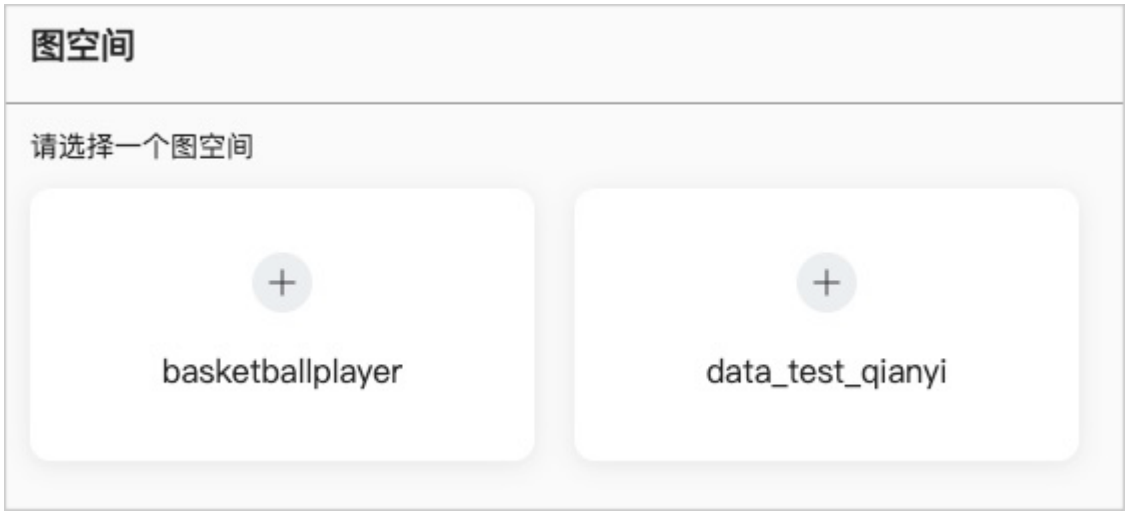
用户使用 Explorer 的功能前，需要先选择一个图空间，然后才能对图空间中的数据进行查询和分析。本文介绍如何选择图空间。

前提条件

已连接 Explorer。具体操作，参见连接 Explorer。

操作步骤

登录 Explorer 后，系统首先会显示图空间选择页面，用户只需选择目标图空间即可。



之后如果要再次选择图空间，可以通过以下方式：

1. 在 Explorer 页面左侧的导航栏中，单击图空间 图标。
2. 选择目标图空间。

> **Note**
>
> 用户可以多次选择相同或者不同的图空间，每次选择都会创建新的画布。

最后更新: June 20, 2023

2022 vesoft Inc.

17.5.2 开始探索

为了探索图数据，用户需要查询出一些初始数据，在这些初始数据的基础上，才能进一步的分析、过滤。本文介绍如何查询初始数据。

前提条件

已选择目标图空间。详情参见选择图空间。

> 版本兼容性

针对 3.0.0 以下版本的 Nebula Graph，需要创建索引后才可以使用查询功能。更多信息，参见创建索引。

操作步骤

在 Explorer 页面左侧单击开始图标 然后查询数据。查询到的数据会显示在画布中。查询方式如下：

- VID 查询
- Tag 查询
- 子图查询

VID 查询

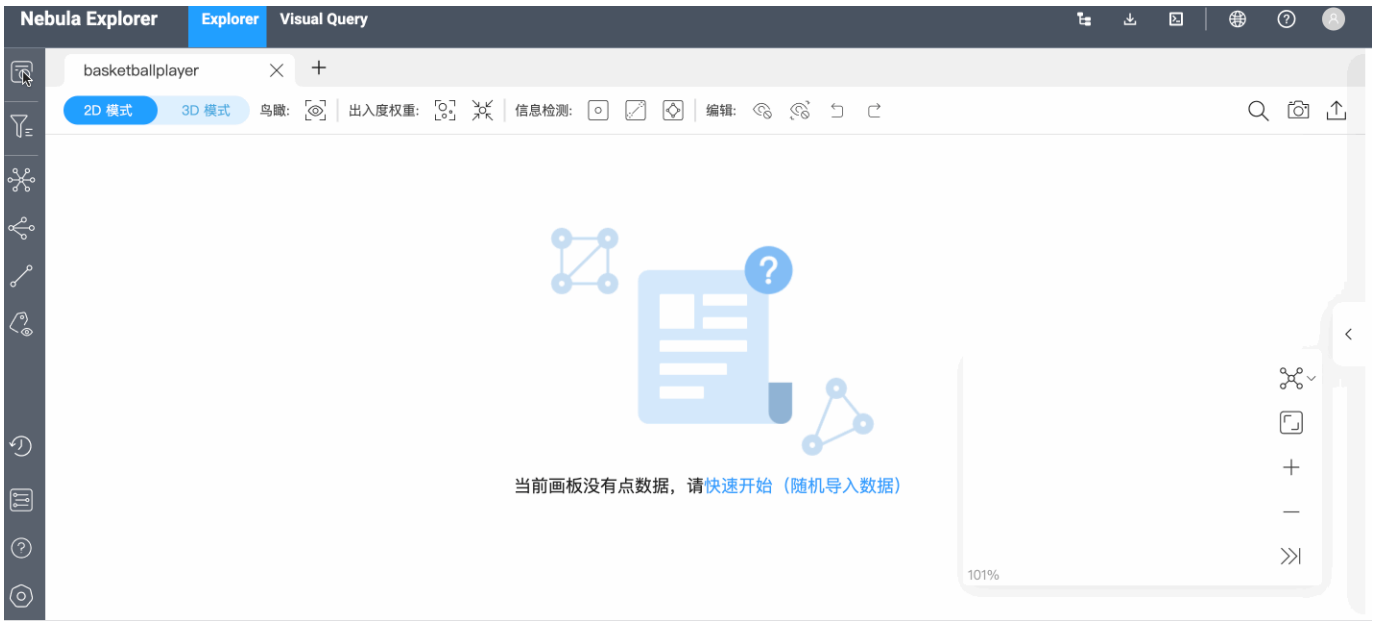
用户可输入点的 ID（VID）查询目标点。

支持手动输入、随机导入、文件导入 VID。

> **Note**

查询区域中每行仅支持填入一个 VID，用回车键分隔。

下图以图空间 `basketballplayer` 及相关数据为例演示如何查询数据。



TAG 查询

用户可以选择 Tag 和对应索引查询目标点，还可以设置结果数量限制和筛选条件。

2022 vesoft Inc.

> **Note**
>
> 请确保对应的图空间中已存在相应的 Tag 和索引，否则无法选择。详情参见创建 Tag 和创建索引。

下图为查询 10 个年龄大于 30 岁，且不等于 40 岁的球员的示例。



子图查询

用户可以输入一个或多个 VID，指定查询的步数、边类型及流入流出的方向查询子图数据。VID 为必选项，可选项步数默认值为 1，边类型默认值为全部。

> **Note**
>
> 当输入多个 VID 时，VID 之间以 `Enter` 键隔开。

下图为 VID 值为 `Kings` 和 `Suns`，步数为 `2` ，边类型为 `server` 和 `like` 的入边的示例。

2022 vesoft Inc.



最后更新: June 20, 2023

2022 vesoft Inc.

17.5.3 节点筛选

用户可以根据过滤条件筛选画布中的点和边。支持仅通过 Tag 或通过一组或多组筛选条件对画布中显示的点进行过滤。

前提条件

使用节点筛选功能前，确保画布中有点数据。具体操作，参见开始探索。

使用说明

- 使用 **Tag** （标签）过滤画布中的点及关联的边时：
- 筛选条件面板显示图空间中的所有 Tag。
- Tag 图例被过滤（置灰）时，画布中的对应 Tag 的点及关联的边会被隐藏。
- 对于多 Tag 的点，如果其任一 Tag 被过滤，则该点会被隐藏。
- 支持通过搜索框搜索 Tag。

- 使用筛选条件过滤画布中的点及关联的边时：
- 每一组筛选条件只针对带此 Tag 的数据，筛选条件包括：Tag、属性、运算符、值。如满足条件，画布中的点会被自动添加选中状态，不满足条件的点及关联的边可被设置为隐藏或置灰。其他 Tag 数据状态不受影响。
- 通过标签过滤（置灰）的优先级最高。如果筛选条件中配置了已通过标签过滤（置灰）的 Tag，画布中不会展示对应的数据。
- 每次只能选择一个 Tag。如果要过滤多个 Tag，需要多次进行添加筛选条件操作。
- 筛选条件不支持对同一个 Tag 进行多次过滤，只会显示第一次过滤的结果。

示例

示例**1** 过滤画布中 **TAG** 为 **PLAYER** 的所有点

1. 在左侧导航栏中，单击节点筛选图标 。
2. 在筛选条件面板中，单击**player**。
3. 画布中显示只有 Tag 为 team 的点。



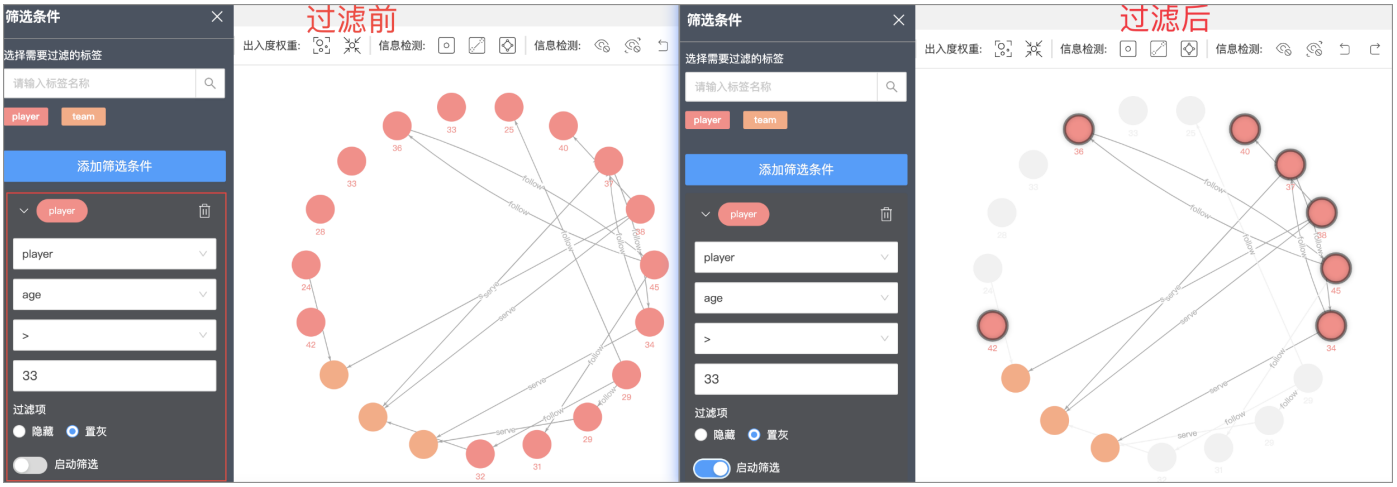
上图中被过滤出的橙色点即 Tag 为 team 的点。

示例**2** 筛选画布中年龄大于 **33** 岁的球员

1. 在左侧导航栏中，单击节点筛选图标 。
2. 单击添加筛选条件，然后设置筛选项（本示例设置的值分别为 `player` 、 `age` 、 `>` 、 `33` ）。
3. 单击置灰将不满足过滤条件的点显示为灰色。

2022 vesoft Inc.

4. 打开启动筛选开关。



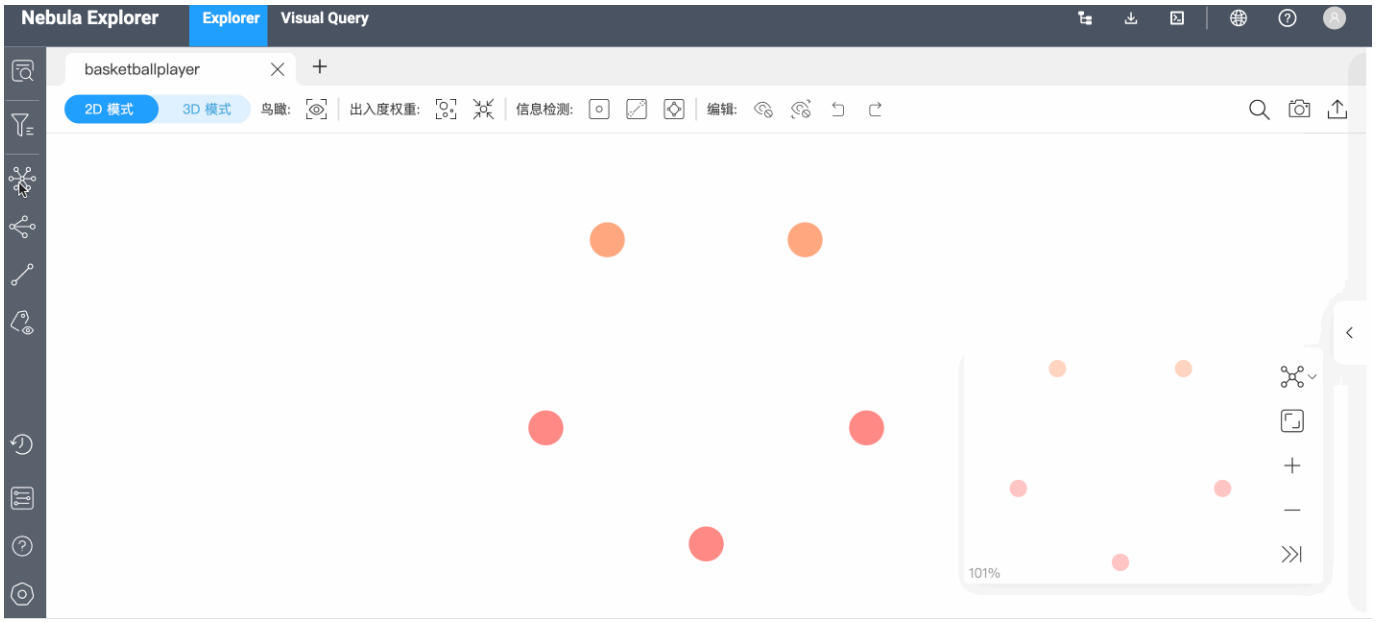
最后更新: June 20, 2023

2022 vesoft Inc.

17.5.4 探索拓展

图数据库的探索拓展功能分为以下四类：

- 拓展
- 共同邻居
- 路径查询
- 查看属性



前提条件

使用探索拓展的所有功能前，确保画布中有点数据。具体操作，参见开始探索。

拓展

1. 在左侧导航栏，单击 图标，打开拓展窗口。在窗口内设置拓展条件，包括边类型、方向、点样式、步数或更精确的条件，说明如下。

参数	描述
边类型	默认显示并选中图空间中的所有边。
方向	定义所有选中点的边的方向来查询目标点，包括 `流出` 、`流入` 、`双向` 三个选项。
点的样式	`按标签类型分类`：目标点的显示颜色和相应的 Tag 颜色一致。 `自定义样式`：自定义目标点的显示颜色。
步数	`单步`：自定义选中点至目标点的跳数。 `范围`：自定义选中点至目标点的范围。
筛选条件	根据筛选条件查询目标点。

2. 选择需要进行拓展的点，可以按住右键框选或按住 `Shift` 键单击左键选择画布中的多个点，然后在拓展窗口单击拓展。对于单个点，可以直接在点上双击左键拓展。

2022 vesoft Inc.

> **Note**
>
> 系统会在面板内的配置被修改后保存当前配置，当双击点或者右键对点快捷拓展时候会以当前配置进行拓展。

共同邻居

在左侧导航栏，单击  图标，打开共同邻居窗口。用户可以按住右键框选或按住 `Shift` 键单击左键选择画布中的两个点或多个点，查询它们的共同邻居。选中的点无共同邻居时，默认返回没有相应数据。

路径查询

1. 在左侧导航栏，单击  图标，打开路径查询窗口。在窗口内设置边类型、方向、查询类型等，说明如下。

参数	描述
边类型	默认显示并选中图数据库中的所有边。
方向	定义起始点的边的方向来查询和目标点的路径，包括 `流出` 、 `流入` 、 `双向` 三个选项。
查询类型	`全路径` ：显示从起点至终点所有路径的点和边。 `最短路径` ：显示从起点至终点的最短路径的点和边。 `非循环路径` ：显示从起点至终点的非循环路径的点和边。
选中步骤	起始点至终点的步数限制。
筛选条件	根据筛选条件筛选目标点。

2. 按住 `Shift` 键单击左键选中画布中的两个点，默认第一个点为起点，第二个点为终点。然后在路径查询窗口单击查询路径。

查看属性

在左侧导航栏，单击  图标，打开查看属性窗口。用户可以选择在画布中展示或隐藏点或边的属性。

> **Note**
>
> - 缩放比例大于 90% 时候在画布上才会显示点属性，小于 90% 的时候会自动隐藏。
> - 缩放比例大于 100% 时候在画布上才会显示边属性，小于 100% 的时候会自动隐藏。

最后更新: June 20, 2023

2022 vesoft Inc.

17.5.5 图计算

为了更好地挖掘、分析图数据，用户可以基于画布中的点边进行图计算，并直接查看图计算结果。

> **Note**
>
> 本功能仅对画布中现存的点进行图计算，如果需要进行复杂图计算，推荐使用 Workflow 进行复杂的可视化图计算。

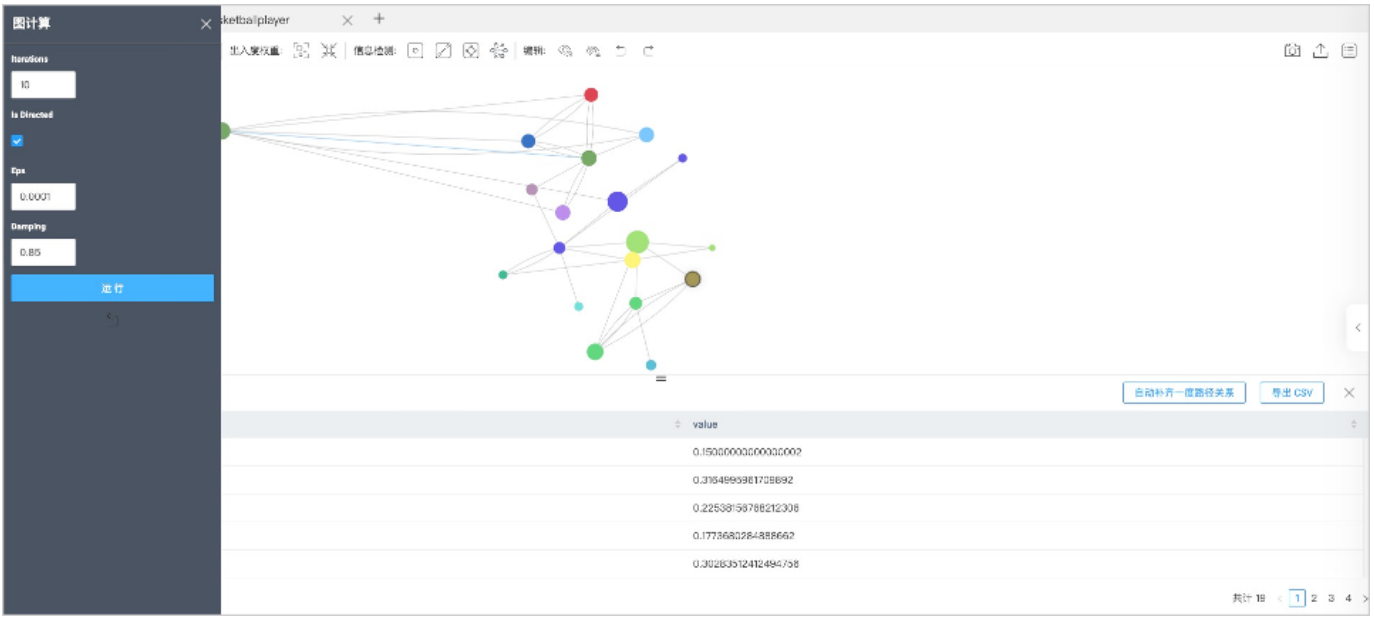
前提条件

确保画布中有图计算所需的点边数据。具体操作，参见开始探索。

操作步骤

1. 在左侧导航栏，单击 图标，打开图计算窗口。
2. 选择算法并设置相关参数。关于算法和参数的说明，请参见算法简介。
3. 单击运行，画布下方会弹出图计算结果。
4. 在结果页面可以：

- 单击自动补齐一度路径关系将画布中所有点之间的一度路径关系补齐。
- 单击导出**CSV**下载 CSV 格式的图计算结果文件。



最后更新: June 20, 2023

2022 vesoft Inc.

17.6 可视化查询

可视化查询帮助用户通过构造查询场景，查询目标数据并查看与之匹配的语句。用户只需拖拽和连线即可构造可视化查询语句，系统将查询结果展示在画布上。

mpatibility

不兼容 3.0.0 以下版本的 Nebula Graph。

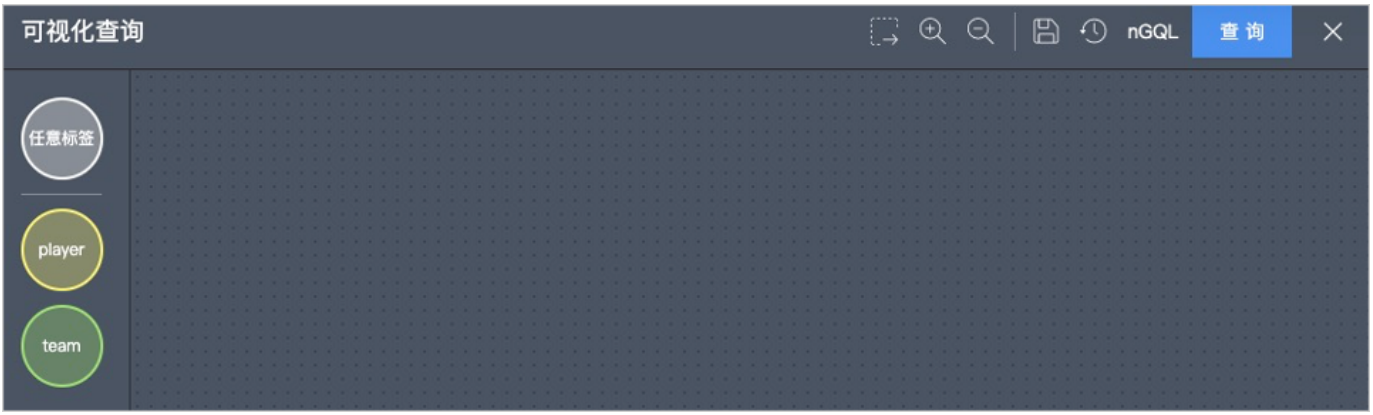
Note

Explorer 的可视化查询功能目前处于测试阶段。

17.6.1 前提条件

- 已选择图空间。具体操作，参见选择图空间。
- 部分查询需先创建索引。详情参见MATCH语句注意事项和创建索引。

17.6.2 页面元素



在 Explorer 页面顶部，单击 **Visual Query** 进入可视化查询页面。在可视化查询页面左侧可以看到图空间对应的所有 Tag (如 player 和 team) 和名为任意标签 Tag。

Note

任意标签的 Tag 也可以用于查询无 Tag 的点。

2022 vesoft Inc.

在可视化页面中，其他图标的描述如下表。

图标/元素	描述
	框选的点和边即为需要查询的结果，双击边框可限制查询数量（优先级高于全局设置中的最大返回数的值）。目前不支持只查询边。
	放大可视化查询页面。
	缩小可视化查询页面。
	保存当前的可视化查询图，保存的图被缓存在浏览器中。
	查看保存的可视化查询图，最多查看 10 份最近保存的可视化查询图。单击任一保存的图可将其显示在可视化查询页面中。
nGQL	单击 **nGQL** 为构图显示查询语句。
查询	单击查询将查询结果可视化地显示在画布中。

17.6.3 操作步骤

1. 在可视化查询页面左侧拖动若干个目标 Tag 到右侧创建对应 Tag 的点。
2. 单击点，在点边缘的锚点上按住鼠标左键拖动至另一个点，创建对应的边。
3. 单击点配置过滤条件。说明如下。



- 标签名：设置零、一个或多个 Tag。

2022 vesoft Inc.

> **Note**
>
> 一个点可以有 0 到多个 Tag：
>
> - 设置 0 个 Tag 时，表示查询无 Tag 的点。
> - 设置一个 Tag 时，表示查询该 Tag 的点。
> - 设置多个 Tag 时，表示查询同时包含所有设置的 Tag 的点。

- 筛选条件：设置一组或多组针对点的过滤条件，包括点的属性、运算符、属性值。

> **Note**
>
> 当配置多 Tag 的点时，不支持设置筛选条件查询数据。

4. 单击边配置过滤条件。说明如下。



- 边类型：设置一个或多个 Edge Type。

> **Note**
>
> 一条边有且仅有一个 Edge Type：
>
> - 设置一个 Edge Type 时，表示查询该 Edge Type 的边。
> - 设置多个 Edge Type 时，表示查询包含任一 Edge Type 的边。

- 方向：设置边的方向，包括流出、流入、双向。
- 单步：设置定长路径。
- 范围：设置变长路径。
- 筛选条件：设置一组或多组针对边的过滤条件，包括边的属性、运算符、属性值。

2022 vesoft Inc.

Note

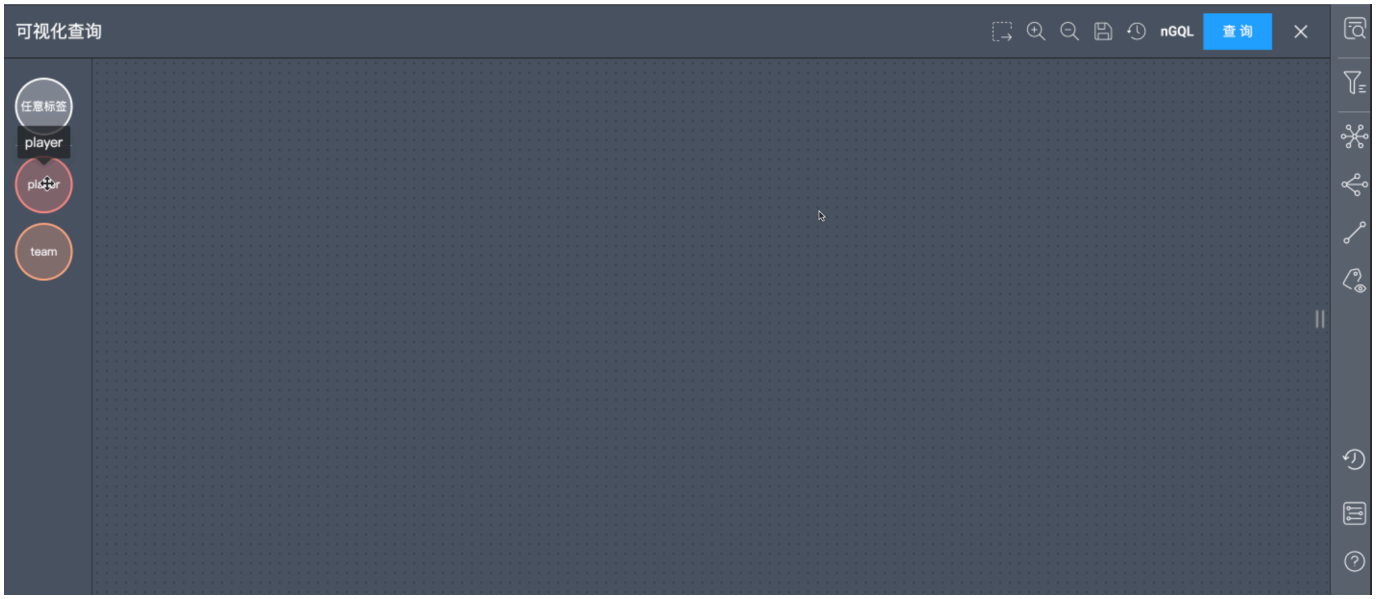
当配置多个 Edge Type 查询数据时，不支持设置筛选条件查询数据。

5. 查询场景(即模式)构建完成后，单击→并框选需要返回（RETURN）的结果。
6. 单击可视化查询页面右上角的查询，画布中会显示查询结果。

17.6.4 示例

示例 **1**

查询和球员 Yao Ming 相互关注，且年龄大于 35 岁的球员以及这些球员都效忠哪些球队，并限制最后查询结果中的效忠关系数为 6。



1. 构造查询模式，拖拽 Tag 至可视化查询面板中（2 个 player 和 1 个 team）。
2. 配置过滤条件。

a. 设置第一个球员的过滤条件为 player.name == Yao Ming。

b. 设置两个球员之间边的边类型为 follow、双向、步数为 1。

c. 设置第二个球员的过滤条件为 player.age > 35。

d. 设置第二个球员和球队之间边的边类型为 serve，方向为流出，步数为 1。

e. 单击→选中第二个球员、球队及它们之间的 serve 边。

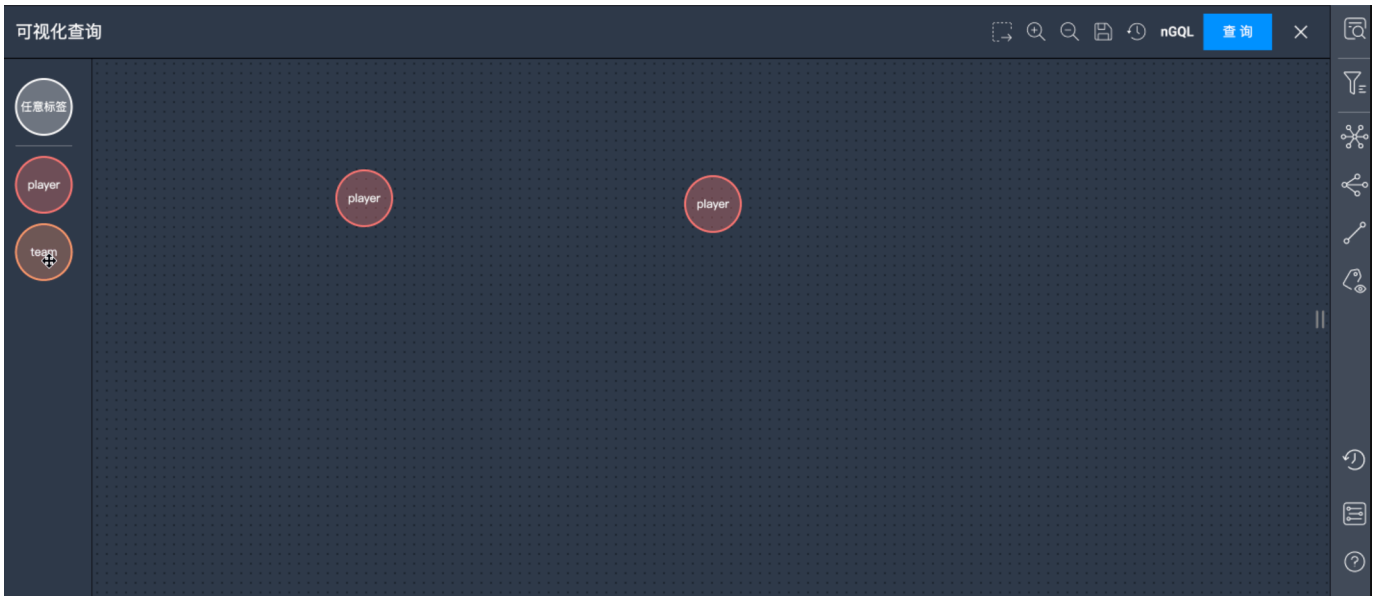
f. 单击模式查询框，然后设置结果数量限制为 6。

3. 单击查询。系统将 6 条查询关系结果显示至画布中。

示例 **2**

查询两个相互关注的球员共同效忠什么球队，并且查询此球队所有年龄大于 30 岁的球员。

2022 vesoft Inc.



1. 构造查询模式，拖拽 Tag 至面板中（3 个 player 和 1 个 team）。

2. 配置过滤条件。

a. 设置第一个和第二个球员之间边的边类型为 follow、双向、步数为 1。

b. 设置第一个球员和球队之间的边的边类型为 serve、方向为流出、步数为 1。

c. 设置第二个球员和球队之间的边的边类型为 serve、方向为流出、步数为 1。

d. 设置第三个球员的过滤条件为 player.age > 30。

e. 设置第三个球员和球队之间的边的边类型为 serve、方向为流出、步数为 1。

f. 单击 选中第三个球员、球队及它们之间的 serve 边。

3. 单击查询。

最后更新: June 20, 2023

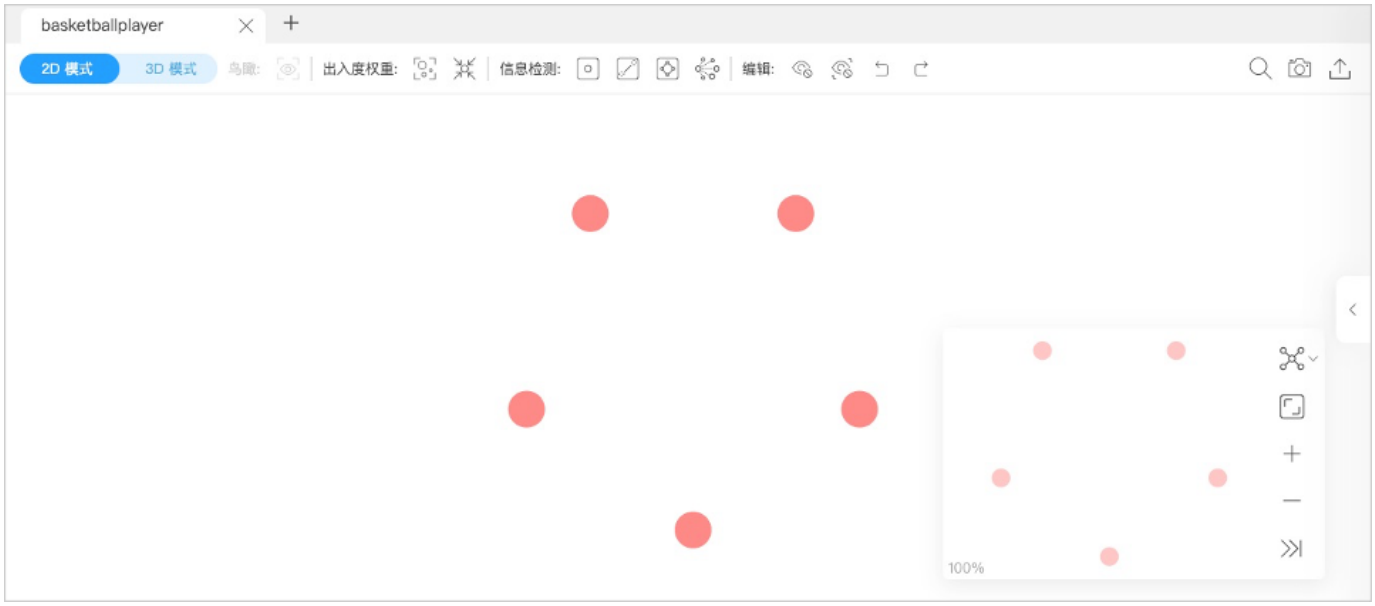
2022 vesoft Inc.

17.7 画布功能

17.7.1 画布概览

用户可在画布中可视化地探索图数据。本文主要介绍画布的组成及相关功能。

画布概览图如下：



顶部页签

用户可以通过单击画布顶部加号 + 图标同时操作多个画布。



- 不同画布上的数据可以来自同一个图空间也可以来自不同的图空间。
- 除了最左边页签的第一个画布外，用户可以自定义画布的名称。

可视化模式

用户可以选择 **2D** 模式和 **3D** 模式可视化地在画布上探索图数据。更多信息，参见可视化模式。

数据存储

用户可以通过创建快照，或以图片、CSV 文件的方式导出画布以存储当前画布数据。

2022 vesoft Inc.

在页面右上方：

- 单击 创建快照。更多信息，参见画布快照。
- 单击，然后单击导出 **CSV** 以 CSV 文件的方式存储画布数据。
- 单击，然后单击导出 **PNG** 以图片形式存储画布数据。

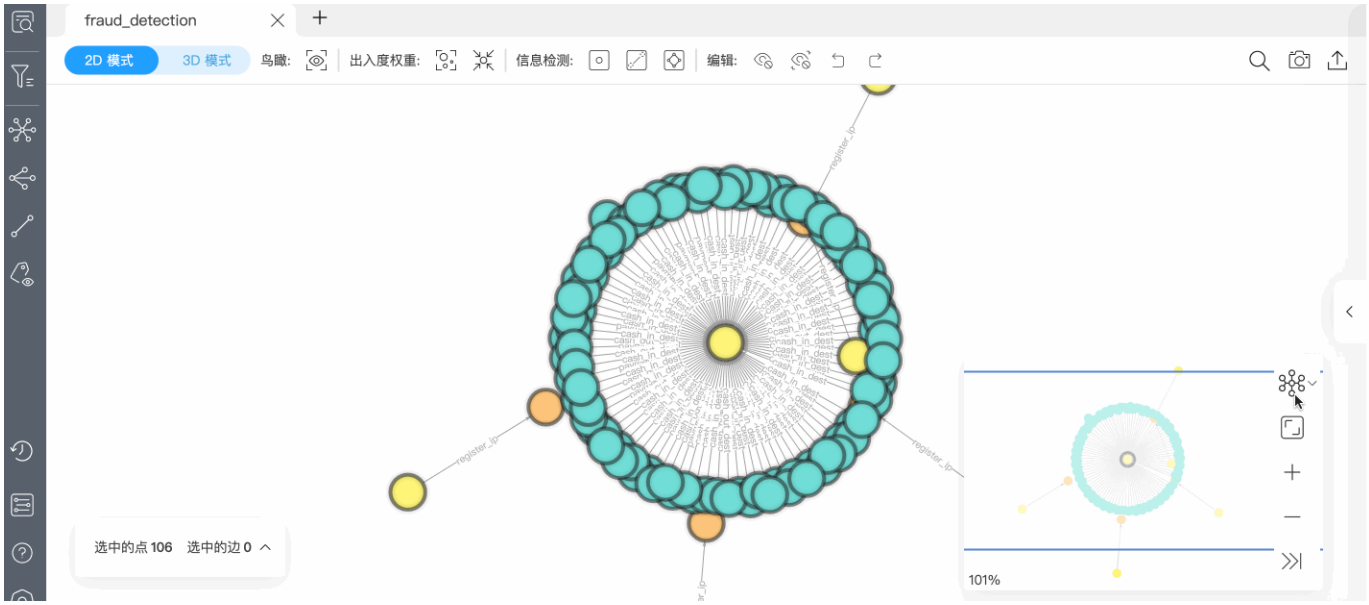
搜索框

在页面右上方，单击，然后在搜索框中输入 VID 或者 Tag 的属性值定位目标点。

多布局

Explorer 支持 6 种布局方式展示画布上的数据之间的关系。

力导向图 层次图 环形图 网格 神经网络 辐射



缩略图

全屏展示画布中的图，收起缩略图，缩小或放大画布中的图等。同时在缩略图的左下角显示了画布中的图占总图的百分比。

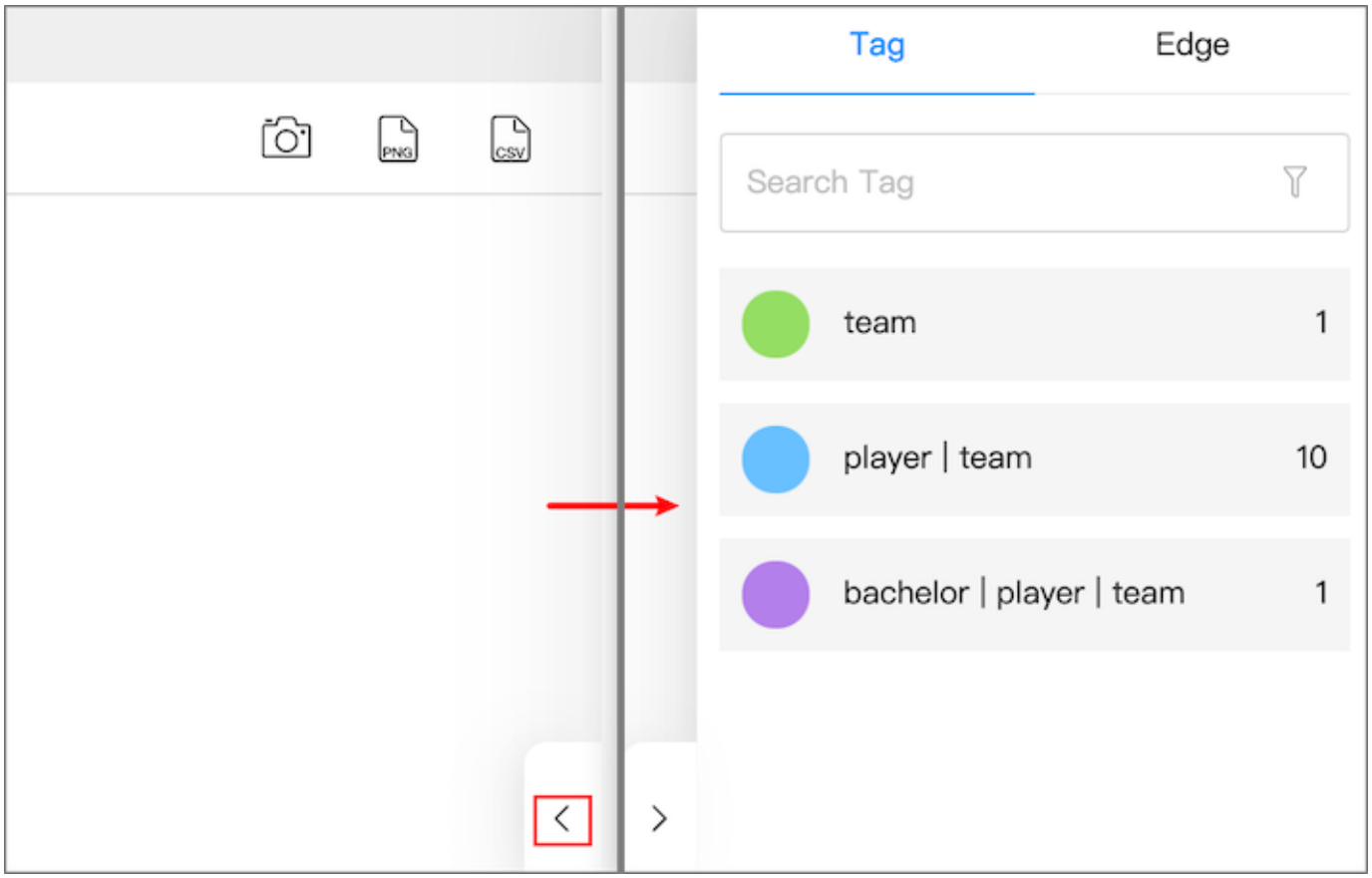


2022 vesoft Inc.

数据概览

在页面右侧，单击 展开数据概览面板。

2022 vesoft Inc.

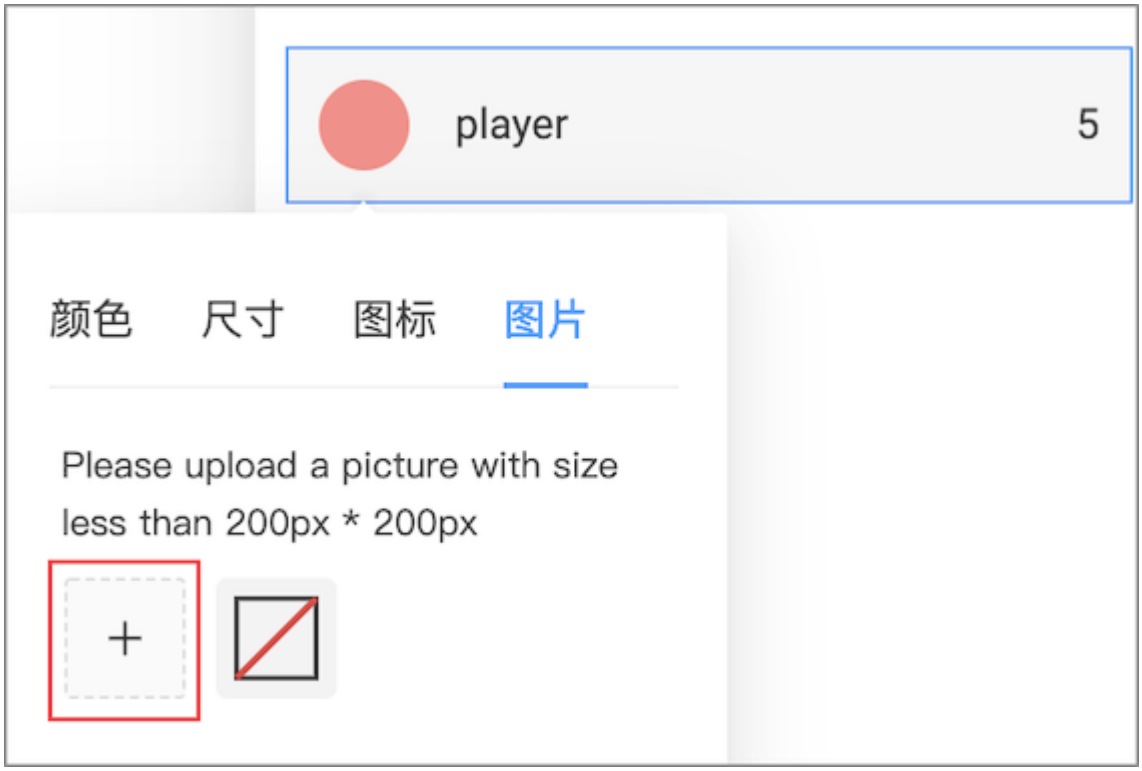


- 可以在数据概览面板中看到画布中的 Tag 数、Edge type 数，及对应的点和边的数量。
- 在数据概览面板，单击 Tag 颜色图标，可自定义 Tag 相同的点的颜色、尺寸、图标。

> **Note**
>
> Tag 完全相同的 VID 点的颜色相同。在画布中右键单击单个点可手动修改一个点的样式。

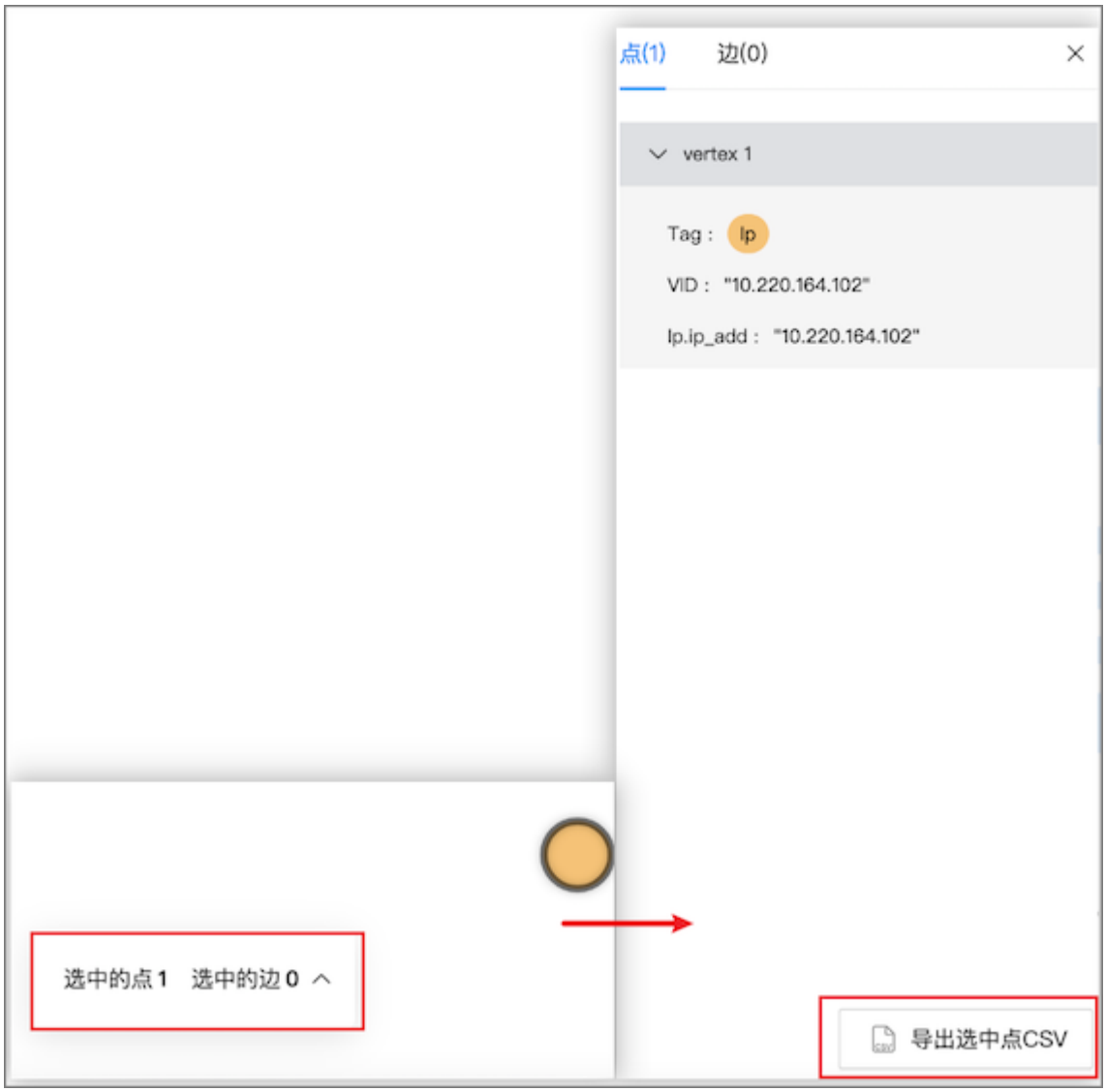
2022 vesoft Inc.

- 支持上传图片以个性化标记画布中的点的样式，上传的图片被存在浏览器中。如果需要永久保存上传的图片，可以将当前画布中的数据创建为快照，具体操作，参见数据快照。



选中画布中的点和边，然后在画布左下角单击选中的点 **{number}**，选中的边 **{number}**查看选中的点或边的数据信息并以 CSV 文件方式导出数据。

2022 vesoft Inc.



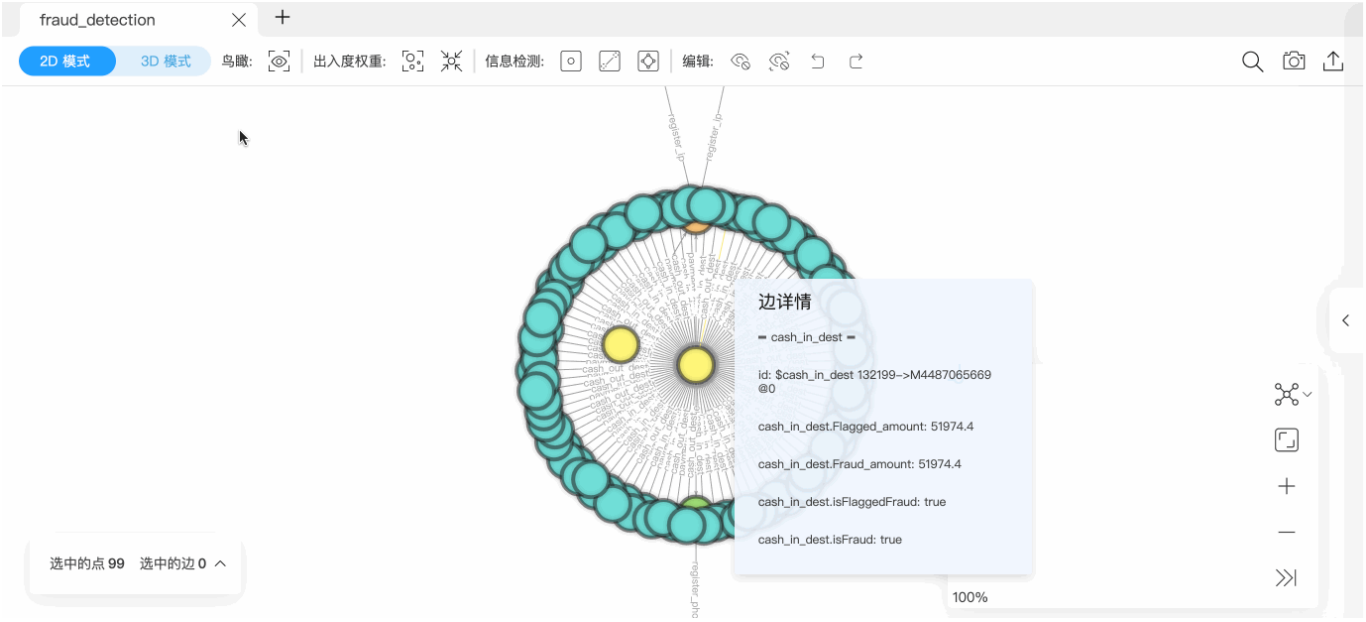
最后更新: June 20, 2023

2022 vesoft Inc.

17.7.2 可视化模式

Explorer 支持 **2D** 和 **3D** 两种可视化模式探索数据。2D 模式支持数据查看和图操作；3D 模式通过不同视角对图数据进行探索，适用于数据量级比较高或者需要演示的场合。

> **Note**
>
> 3D 模式下，不可进行图数据的操作。



2D 模式

2D 模式下可对画布上的数据进行探索操作：



参数	说明
出入度权重	出入度权重：根据画布中所有点的出边和入边的条数，自动调整点的大小。 重置出入度：重置画布中的点至原始大小。
信息检测	孤立点：检测画布中所有的无边点。 悬边：检测画布中与度数为1的点关联的边（包含关联点）。 环检测：检测由边和点连接在一起形成循环的路径。 N 步节点检测：从选中点开始，逐步显示出方向的点。
编辑	隐藏：隐藏画布中选中的点边。 隐藏其他：隐藏画布中未选择的所有点边。 撤销：撤销上一步新增或隐藏的操作。 重做：恢复上一步撤销的操作。

有关更多的 2D 模式下可进行的操作，参见画布功能。

2022 vesoft Inc.

3D 模式



在页面左上方，单击视图切换开关可切换至 3D 视图模式。3D 视图模式下，可切换回到 2D 图探索模式，并且不会对 2D 的探索产生任何副作用。

参数	说明
鸟瞰	以鸟瞰的方式展示当前图空间的所有数据。默认最多展示图空间中 20000 节点，2000 边。单击 可以调整，但是设置过大可能会导致浏览器崩溃。
画质	高：点以球的形式展示，光影效果更好。 低：点以圆的形式展示，可以支持更大数据量。
重新受力	当不同的点重叠时，分散点与点之间的距离。

版本兼容性

针对 3.0.0 以下版本的 Nebula Graph，需要创建索引后才可以使用鸟瞰功能。更多信息，参见创建索引。

最后更新: June 20, 2023

2022 vesoft Inc.

17.7.3 管理快照

Explorer 的快照功能可保存图探索得到的可视化结果，以便再次打开浏览器时还原保存的图数据。

创建快照

1. 在画布右上角，单击相机 图标。
2. 填写快照名称和备注（选填）。
3. 单击提交。

> **Note**
>
> 已创建的快照被存放在快照历史中。更多信息，参见下文。

快照历史

> **Note**
>
> - 目前最多存储 50 个历史快照。
> - 历史快照存储在浏览器本地，清理浏览器可能造成快照数据丢失。

在 Explorer 页面左侧导航栏，单击 图标进入快照列表页面。用户可切换图空间查看相应图空间的历史快照，还可以导入历史快照至画布，下载历史快照至本地，以及删除快照。

在目标历史快照右侧的操作列下：

- 单击 导入历史快照至新画布中。
- 单击 以 JSON 的格式下载历史快照至本地。
- 单击 删除历史快照。

在快照列表左上方，单击导入快照可导入之前下载的 JSON 格式快照文件至快照列表中，以便用户之间离线共享快照数据。系统会根据 JSON 文件中记录的图空间信息将导入的快照自动放置于相应的图空间中。

最后更新: June 20, 2023

2022 vesoft Inc.

17.8 工作流

17.8.1 工作流简介

Nebula Explorer 支持用自定义工作流进行可视化的复杂图计算。

背景信息

Nebula Explorer 提供多种组件（Component），包括图查询组件和图计算组件，用户可以基于编排调度工具 Dag Controller 自由组合这些组件，例如将图查询组件的输出作为图计算组件的输入，最终形成一个有向无环的工作流（Workflow）。



需要进行图计算时再将工作流实例化，组件实例化后称为任务（Task），工作流实例化后称为作业（Job）。一个作业可以由多个任务构成。Explorer 将作业发送给 Nebula Analytics 进行图计算，完成后可以在作业列表查看对应的图计算结果。

功能说明

- 支持新增、查看、修改、删除、对比、克隆、重命名工作流。
- 支持查询组件和多种算法组件，可以搜索、新增、配置、重命名组件。
- 支持查看作业列表、进度、结果、日志，以及重跑作业。
- 支持搜索工作流或作业。

注意事项

- 使用工作流需要额外部署 Dag Controller 和 Nebula Analytics。详情参见部署依赖服务。
- 图查询组件的输入只能是查询语言。
- 图查询组件的结果仅支持保存在 HDFS，方便被多个算法调用。
- 图计算组件的输入可以是 Nebula Graph 或 HDFS 的指定数据，也可以依赖于图查询组件的结果。 如果依赖于前一个图查询组件的结果，必须和该图查询组件全连接，即该图计算组件的白色输入锚点全都和上一个图查询组件的白色输出锚点连接。
- 部分算法的参数配置也可以依赖于上游节点。
- 图计算组件结果允许保存在 HDFS、Nebula Graph 中，但是并非所有算法的结果都适合存入到 Nebula Graph 里，在设置结果保存页面时，部分算法只能选择 HDFS。

算法参数配置说明

请参见算法简介。

2022 vesoft Inc.

最后更新: June 20, 2023

2022 vesoft Inc.

17.8.2 资源配置

使用工作流之前，需要进行资源配置，包括 Nebula Graph 配置、HDFS 配置和 Nebula Analytics 配置。

前提条件

已部署 3.2.0 或以上版本的 Nebula Analytics 和 3.2.0 或以上版本的 Dag Controller。详情参见部署依赖服务。

操作步骤

1. 在 Explorer 页面顶部的导航栏中，单击 **Workflow**。
2. 在工作流标签页单击配置。
3. 配置如下三种资源:

类型	说明
Nebula Graph 配置	执行图查询或写入图计算结果的 Graph 服务地址。默认为登录 Explorer 使用的 Graph 服务，无法修改。可以设置三种服务的超时时间。
HDFS 配置	存储结果的 HDFS 地址。单击添加新增地址，设置 HDFS 名称、HDFS 路径、HDFS 用户名（可选）。
Nebula Analytics 配置	执行图计算的 Nebula Analytics 地址。单击添加新增地址。

4. 配置完成后，单击确认。

最后更新: June 20, 2023

2022 vesoft Inc.

17.8.3 工作流示例

本文介绍如何创建一个简单的工作流。

前提条件

- 已有数据源。数据源可以是 Nebula Graph 中的数据或者 HDFS 上的 CSV 文件。
- 已完成资源配置。

2022 vesoft Inc.

新增工作流

以 MATCH 语句 `MATCH (v1:player)--(v2) RETURN id(v1), id(v2);` 的结果做为 PageRank 算法的输入，介绍如何创建一个简单的工作流。

2022 vesoft Inc.

1. 在 Explorer 页面顶部的导航栏中，单击 **Workflow**。
2. 在工作流标签页单击新建工作流，进入流程画布页面。
3. 在左侧组件库列表里选择查询->**Query**，按住左键拖拽至画布中，单击该图查询组件，在右侧弹出的工作流配置面板中进行如下设置。



配置项	说明
Query	单击 可以修改组件名称，方别识别。
查询语言	选择执行 nGQL 语句的图空间，以及填写nGQL语句。填写语句后单击解析参数，会在输出里展示返回的列名。
输入	设置自定义参数，可以用于参数化查询。单击添加参数可以增加更多自定义参数。
输出	解析查询语言得到的返回结果列名。可以修改名称，相当于用 `AS` 设置列的别名。
结果	设置结果的保存位置。为方便其他算法调用结果，图查询组件的结果只支持保存在 HDFS 上。

Note

连接锚点显示为黄色，表示可选，可以自行设置或者由其他任意组件提供输入。

4. 在左侧组件库列表里选择节点重要度->**PageRank**，按住左键拖拽至画布中，将图查询组件的 `output0` 锚点连接至图计算组件的 `input0` 锚点， `output1` 锚点连接至 `input1` 锚点。

2022 vesoft Inc.



5. 单击该图计算组件，在右侧弹出的工作流配置面板中进行如下设置。

2022 vesoft Inc.



配置项	说明
PageRank	单击 可以修改组件名称，方别识别。
输入	支持三种数据源作为输入。**Nebula Graph** 可以选择图空间和对应的边类型；**Dependence** 会根据锚点的连接情况自动识别；**HDFS**可以选择 HDFS 和数据源文件的相对路径。
参数配置	设置图算法特有的参数。不同算法的参数不同，部分参数可以从任意上游组件获取，此时锚点显示为黄色。
输出	显示图计算结果的列名，无法修改。
执行配置	机器数量表示将要执行算法的机器数量。 进程数量表示将要执行算法的总进程数，会根据机器数量平均分配到各个机器。 线程数量表示每个进程会启动多少个线程。
结果	设置结果的保存位置。可以保存在 HDFS 或 Nebula Graph 中。 保存为 HDFS 时只需要选择 HDFS ，保存路径无需修改，会根据作业和任务 ID 自动生成。 保存为 Nebula Graph 时，需要预先在相应图空间中创建 Tag 用于保存结果。Tag 的属性说明请参见算法简介。 部分算法只支持保存在 HDFS 上。

6. 在画布左上角自动生成的工作流名称旁单击 修改工作流名称，然后在画布右上角单击运行，会自动跳转至作业页面显示作业进度，耐心等待作业完成即可查看结果。详情参见作业管理。

2022 vesoft Inc.

Note

单击运行时，会自动进行保存，如果不执行图计算，仅仅修改，修改完成后请单击  进行保存，或者单击  另存为新的工作流。

常见问题

如何删除组件?

单击鼠标左键选择要删除的组件，单击 `Backspace` 即可删除该组件。

最后更新: June 20, 2023

2022 vesoft Inc.

17.8.4 工作流管理

本文介绍如何管理工作流，包括查看、修改、重命名、克隆、删除、对比。

操作步骤

1. 在 Explorer 页面顶部的导航栏中，单击 **Workflow**。
2. 在工作流标签页可以查看所有已保存的工作流，页面显示工作流名称、创建时间、最近一次修改时间、包含的图算法。

- 在列表上方单击对比，可以选择两个工作流或者相同工作流的不同历史版本进行代码的对比。
- 在列表上方的搜索框，可以通过关键字搜索工作流。
- 在列表的操作列，可以执行如下操作：
- 运行：直接将该工作流实例化为作业执行。
- 打开：打开工作流查看，可以修改工作流，修改完后请单击  进行保存，或者单击  另存为新的工作流。
- 查看作业：跳转至作业列表查看该工作流实例化的所有作业。
-  ：可以查看工作流的代码、重命名工作流、克隆工作流和删除工作流。

最后更新: June 20, 2023

2022 vesoft Inc.

17.8.5 作业管理

本文介绍如何查看作业列表、进度、结果、日志，以及重跑作业。

操作步骤

1. 在 Explorer 页面顶部的导航栏中，单击 **Workflow**。
2. 在作业标签页可以查看所有执行的作业，页面显示作业 ID、作业名称、状态、创建时间、结束时间、工作流版本。

- 在列表上方单击对比，可以选择两个工作流或者相同工作流的不同作业进行代码的对比。
- 在列表上方的筛选框，可以筛选工作流和版本。
- 在列表上方的搜索框，可以通过关键字搜索作业。
- 在列表的操作列，可以执行如下操作：
- 在**Explorer**中打开：对于执行成功的作业，可以选择图空间和组件，查看组件的输出结果。支持将结果导出为 CSV 文件。
- 重跑：对于执行失败的作业，可以重跑该作业。
- 打开：可以重跑作业、查看作业结果、日志，以及跳转到工作流中进行编辑（工作流为最新版本）。

最后更新: June 20, 2023

2022 vesoft Inc.

17.8.6 工作流 API

工作流 **API** 概览

Nebula Explorer 提供 API 接口使用工作流的部分功能。

当前支持的 API 接口如下：

- 新增作业
- 获取所有作业列表
- 获取指定工作流的作业列表
- 查询指定作业详情
- 取消作业运行
- 获取指定任务的运行结果数据

请求方式

支持使用 curl 调用 API 接口实现对应的功能。

格式如下：

```
curl <options> http://<explorer_address>:<explorer_port>/<api_path>?{<body>}
```

- `<options>`：curl 支持大量选项，工作流使用较多的是 `-X` 、`-H` 、`-d` 。关于选项的详细说明，参见 curl 官方文档。
- `<explorer_address>`：Nebula Explorer 访问地址。
- `<explorer_port>`：Nebula Explorer 访问端口。
- `<api_path>`：API 的调用路径。例如 `api-open/v1/jobs` 。
- `<body>`：调用 API 时传入的 Body 参数。

获取授权 **TOKEN**

使用 API 时，需要做 Token 信息校验。请使用如下命令获取 Token 信息。

```
curl -i -X POST -H "Content-Type: application/json" -H "Authorization: Bearer <account_base64_encode>" -d '{"address":"<nebula_address>","port":<nebula_port>}' http://
<explorer_address>:<explorer_port>/api-open/v1/connect
```

- `<account_base64_encode>`：Base64 编码后的 Nebula Graph 账号和密码。编码前格式为 `账号:密码` ，下文示例为 `root:123` ，编码后为 `cm9vdDoxMjM=` 。
- `<nebula_address>`：Nebula Graph 访问地址。
- `<nebula_port>`：Nebula Graph 访问端口。
- `<explorer_address>`：Nebula Explorer 访问地址。
- `<explorer_port>`：Nebula Explorer 访问端口。

示例：

```
curl -i -X POST -H "Content-Type: application/json" -H "Authorization: Bearer cm9vdDoxMjM=" -d '{"address":"192.168.8.111","port":9669}' http://192.168.8.145:7002/api-open/v1/connect
```

返回结果：

```
HTTP/1.1 200 OK
Content-Type: application/json
Set-Cookie: explorer_token=eyJhbxxx; Path=/; # Max-Age=259200; HttpOnly
Traceparent: 00-1c3f55cdbf81e13a2331ed88155ce0bf-2b97474943563f20-# 00
Date: Thu, 14 Jul 2022 06:47:01 GMT
Content-Length: 54

{
  "code": 0,
  "data": {
    "success": true
```


2022 vesoft Inc.

```
  },
  "message": "Success"
}
```

需要关注的参数如下：

- `explorer_token`： Token 信息。
- `Max-Age`： Token 有效时间。单位：秒。默认为 259200 秒，即 3 天。可以在安装目录内的 `config/app-config.yaml` 文件内修改默认有效时间。

请求结果

- API 如果调用成功，会返回如下信息：

```
{
  code: 0,
  message: 'Success',
  data: <ResponseData> //根据接口返回相应结果。
}
```

- API如果调用失败，会返回对应的通用错误码，例如：

```
{
  code: 40004000,
  message: '<ErrBadRequest>', //返回具体报错信息。
}
```

通用错误码的说明参见下文。

通用错误码

错误码	信息	说明
40004000	`ErrBadRequest`	请求异常
40004001	`ErrParam`	请求参数异常
40104000	`ErrUnauthorized`	请求未授权
40104001	`ErrSession`	登录会话异常
40304000	`ErrForbidden`	请求被拒绝
40404000	`ErrNotFound`	请求资源不存在
50004000	`ErrInternalServer`	内部服务异常
50004001	`ErrInternalDatabase`	数据库异常
50004002	`ErrInternalController`	控制器异常
50004003	`ErrInternalLicense`	证书校验异常
90004000	`ErrUnknown`	未知错误

任务状态码

状态码	说明
0	准备中
1	执行中
2	执行成功
3	执行失败
4	已中断
5	暂停中

2022 vesoft Inc.

最后更新: June 20, 2023

2022 vesoft Inc.

新增作业

本文介绍如何使用工作流的 API 新增作业。

API 路径

`api-open/v1/workflows/<workflow_id>/jobs`

`<workflow_id>`：工作流 ID。参见下文的请求参数。

请求参数

路径参数

参数	类型	是否必填	默认值	示例	说明
`workflow_id`	number	必填	-	`4216617528`	工作流 ID。需要指定工作流将其实例化为作业。可以在指定的工作流页面左上角查看 ID。

Headers 参数

参数	类型	是否必填	默认值	示例	说明
`Content-Type`	string	必填	-	`application/json`	内容类型。
`explorer_token`	string	必填	-	`eyJhbxxx`	授权 Token，用于验证账号信息。如何获取授权 Token 请参见工作流 API 概览。

Body 参数

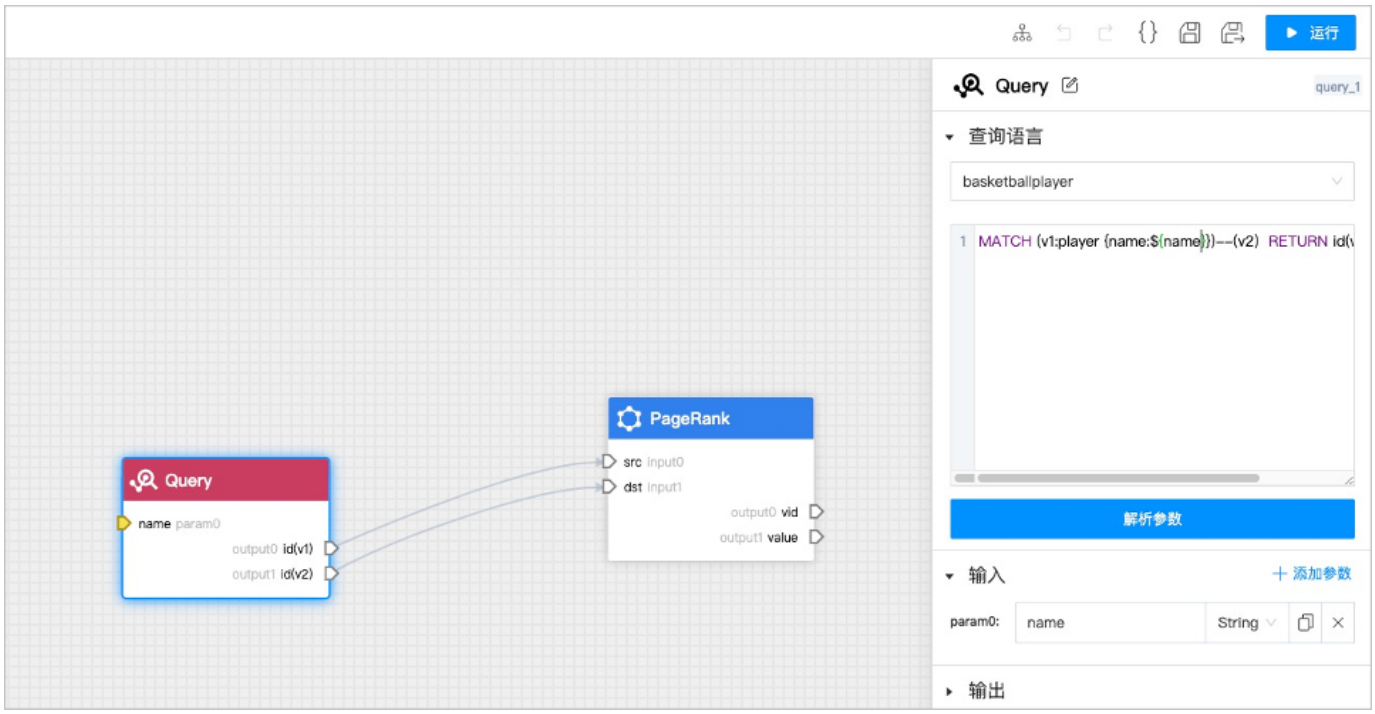
> **Note**
>
> 自定义的传入参数需要用户自行保证参数的合理性和正确性，否则作业会执行失败。

参数	类型	是否必填	默认值	示例	说明
`input`	object	可选	-	-	自定义的传入参数。
- `task_id`	object	可选	-	`query_1`	任务 ID。可以在组件设置页面右上角查看。一个任务可以设置多个由键值对表示的参数。
- `param_name: param_value`	string: {string 或 number}	可选	-	`param0: player100`	`param_name` 为参数的键，即参数名。`param_value` 为参数的值。

请求示例

以下图为例，在 nGQL 语句使用自定义参数 `name`。在创建作业时传入参数值 `Tim Duncan`。

2022 vesoft Inc.




```
curl -i -X POST -H "Content-Type: application/json" -H "Cookie: "explorer_token=eyJhbxxx"" -d '{"input":{"query_1":{"name":"Tim Duncan"}}}' http://192.168.8.145:7002/api-open/v1/workflows/4216617528/jobs
```

返回参数

参数	类型	示例	说明
`code`	number	`0`	请求结果码。请求成功返回 `0`，请求不成功返回对应的错误码。详情参见工作流 API 概览。
`message`	string	`Success`	执行结果信息。
`data`	object	-	返回的数据列表。
- `id`	string	`107`	新增作业的 ID。

返回示例

```
{
  "cookie": [],
  "Content-Type": "application/json",
  "Traceparent": "00-1ba128615cdc2226c921973a689e9f1b-7630b12963494672-00",
  "Date": "Fri, 15 Jul 2022 07:19:25 GMT",
  "Content-Length": "48"
}

{
  "code": 0,
  "data": {
    "id": 107
  },
  "message": "Success"
}
```

最后更新: June 20, 2023

2022 vesoft Inc.

获取所有作业列表

本文介绍如何使用工作流的 API 获取所有作业列表。

API 路径

`api-open/v1/jobs`

请求参数

路径参数

无。

Headers 参数

参数	类型	是否必填	默认值	示例	说明
`Content-Type`	string	必填	-	`application/json`	内容类型。
`explorer_token`	string	必填	-	`eyJhbxxx`	授权 Token，用于验证账号信息。如何获取授权 Token 请参见工作流 API 概览。

Body 参数

参数	类型	是否必填	默认值	示例	说明
`filter`	object	可选	-	-	过滤器的设置。
- `name`	string	可选	-	`workflow_q745a_20220715092236`	作业名称。
- `status`	number	可选	-	`2`	任务状态码。详情参见工作流 API 概览。
- `fromCreateTime`	number	可选	-	`1657848036000`	起始时间戳。基于作业的创建时间进行过滤。
- `toCreateTime`	number	可选	-	`1657848157000`	结束时间戳。基于作业的创建时间进行过滤。
- `orderByCreateTime`	string	可选	`desc`	-	排序方式。支持 `desc` 和 `asc` 。
`pageSize`	number	可选	`10`	-	每页记录数。
`page`	number	可选	`1`	-	页码。

请求示例

> **Note**
>
> `jobs?` 后的内容为 Body 参数，`filter` 的内容是进过 URL 编码的结果。原始内容为：`{ "status": 2, "orderByCreateTime": "asc"}` 。

```
curl -i -X GET -H "Content-Type: application/json" -H "Cookie: "explorer_token=eyJhbxxx"" http://192.168.8.145:7002/api-open/v1/jobs?
filter=%7B%20%22status%22%3A%202%2C%20%20%22orderByCreateTime%22%3A%20%22asc%22%7D&pageSize=10&page=1
```


2022 vesoft Inc.

返回参数

参数	类型	示例	说明
code	number	0	请求结果码。请求成功返回 0，请求不成功返回对应的错误码。详情参见工作流 API 概览。
message	string	Success	执行结果信息。
data	object	-	返回的数据列表。
- total	number	2	记录总数。
- Page	number	1	页码。
- PageSize	number	10	每页记录数。
- items	object	-	记录详情列表。
- id	number	105	作业 ID。
- name	string	workflow_q745a_20220715090915	作业名称。
- workflowId	string	4216617528	工作流 ID。
- workflowName	string	workflow_q745a	工作流名称。
- status	number	2	作业状态码。详情参见工作流 API 概览。
- runBeginTime	number	1657847358000	作业开始执行时间。
- runEndTime	number	1657847364000	作业执行结束时间。
- createTime	number	1657847355906	作业创建时间。

返回示例

```
{
  "cookie": [],
  "Content-Type": "application/json",
  "Traceparent": "00-d3a1943f5baf46771e9afc629e0b5d40-920db2f06142f5ff-00",
  "Date": "Fri, 15 Jul 2022 06:17:21 GMT",
  "Content-Length": "512"
}

{
  "code": 0,
  "data": {
    "items": [
      {
        "id": 105,
        "name": "workflow_q745a_20220715090915",
        "workflowId": "4216617528",
        "workflowName": "workflow_q745a",
        "status": 2,
        "runBeginTime": 1657847358000,
        "runEndTime": 1657847364000,
        "createTime": 1657847355906
      },
      {
        "id": 106,
        "name": "workflow_q745a_20220715092236",
        "workflowId": "4216617528",
        "workflowName": "workflow_q745a",
        "status": 2,
        "runBeginTime": 1657848157000,
        "runEndTime": 1657848163000,
        "createTime": 1657848156290
      }
    ],
    "total": 2,
    "Page": 1,
    "PageSize": 10
  },
```


2022 vesoft Inc.

```
  "message": "Success"
}
```

最后更新: June 20, 2023

2022 vesoft Inc.

获取指定工作流的作业列表

本文介绍如何使用工作流的 API 获取指定工作流的作业列表。

API 路径

`api-open/v1/workflows/<workflow_id>/jobs`

`<workflow_id>`：工作流 ID。参见下文的请求参数。

请求参数

路径参数

参数	类型	是否必填	默认值	示例	说明
`workflow_id`	number	必填	-	`4216617528`	工作流 ID。需要指定工作流将其实例化为作业。可以在指定的工作流页面左上角查看 ID。

Headers 参数

参数	类型	是否必填	默认值	示例	说明
`Content-Type`	string	必填	-	`application/json`	内容类型。
`explorer_token`	string	必填	-	`eyJhbxxx`	授权 Token，用于验证账号信息。如何获取授权 Token 请参见工作流 API 概览。

Body 参数

参数	类型	是否必填	默认值	示例	说明
`filter`	object	可选	-	-	过滤器的设置。
- `name`	string	可选	-	`workflow_q745a_20220715092236`	作业名称。
- `status`	number	可选	-	`2`	任务状态码。详情参见工作流 API 概览。
- `fromCreateTime`	number	可选	-	`1657848036000`	起始时间戳。基于作业的创建时间进行过滤。
- `toCreateTime`	number	可选	-	`1657848157000`	结束时间戳。基于作业的创建时间进行过滤。
- `orderByCreateTime`	string	可选	`desc`	-	排序方式。支持 `desc` 和 `asc` 。
`pageSize`	number	可选	`10`	-	每页记录数。
`page`	number	可选	`1`	-	页码。

请求示例

> **Note**
>
> `jobs?` 后的内容为 Body 参数，`filter` 的内容是进过 URL 编码的结果。原始内容为：`{"status": 2, "fromCreateTime": 1657874100000}` 。

```
curl -i -X GET -H "Content-Type: application/json" -H "Cookie: "explorer_token=eyJhbxxx"" http://192.168.8.145:7002/api-open/v1/workflows/4216617528/jobs?
filter=%7B%22status%22%3A%202%2C%20%20%22fromCreateTime%22%3A%201657874100000%7D&pageSize=10&page=1
```


2022 vesoft Inc.

返回参数

参数	类型	示例	说明
code	number	0	请求结果码。请求成功返回 0，请求不成功返回对应的错误码。详情参见工作流 API 概览。
message	string	Success	执行结果信息。
data	object	-	返回的数据列表。
- total	number	2	记录总数。
- Page	number	1	页码。
- PageSize	number	10	每页记录数。
- items	object	-	记录详情列表。
- id	number	105	作业 ID。
- name	string	workflow_q745a_20220715090915	作业名称。
- workflowId	string	4216617528	工作流 ID。
- workflowName	string	workflow_q745a	工作流名称。
- status	number	2	作业状态码。详情参见工作流 API 概览。
- runBeginTime	number	1657847358000	作业开始执行时间。
- runEndTime	number	1657847364000	作业执行结束时间。
- createTime	number	1657847355906	作业创建时间。

返回示例


```
{
  "cookie": [],
  "Content-Type": "application/json",
  "Traceparent": "00-008c3056686dd3f3be38b8eda42a917e-b5616e30434cb803-00",
  "Date": "Fri, 15 Jul 2022 08:44:06 GMT",
  "Content-Length": "297"
}
{
  "code": 0,
  "data": {
    "items": [
      {
        "id": 115,
        "name": "workflow_q745a_20220715163650",
        "workflowId": "4216617528",
        "workflowName": "workflow_q745a",
        "status": 2,
        "runBeginTime": 1657874212000,
        "runEndTime": 1657874218000,
        "createTime": 1657874210088
      }
    ],
    "total": 1,
    "Page": 1,
    "PageSize": 10
  },
  "message": "Success"
}
```


最后更新: June 20, 2023

2022 vesoft Inc.

查询指定作业详情

本文介绍如何使用工作流的 API 查询指定作业详情。

API 路径

`api-open/v1/jobs/<job_id>`

`<job_id>`：作业 ID。参见下文的请求参数。

请求参数

路径参数

参数	类型	是否必填	默认值	示例	说明
`job_id`	number	必填	-	`1964`	作业 ID。可以通过 API 获取所有作业列表查询，或者在作业列表页面查看。

Headers 参数

参数	类型	是否必填	默认值	示例	说明
`Content-Type`	string	必填	-	`application/json`	内容类型。
`explorer_token`	string	必填	-	`eyJhbxxx`	授权 Token，用于验证账号信息。如何获取授权 Token 请参见工作流 API 概览。

Body 参数

无。

请求示例

```
curl -i -X GET -H "Content-Type: application/json" -H "Cookie: "explorer_token=eyJhbxxx"" http://192.168.8.145:7002/api-open/v1/jobs/1964
```


2022 vesoft Inc.

返回参数

参数	类型	示例	说明
`code`	number	`0`	请求结果码。请求成功返回 `0`，请求不成功返回对应的错误码。详情参见工作流 API 概览。
`message`	string	`Success`	执行结果信息。
`data`	object	-	返回的数据列表。
- `id`	number	`1964`	作业 ID。
- `name`	string	`workflow_xkkjf_20220712103332`	作业名称。
- `workflowId`	string	`3992429968`	工作流 ID。
- `workflowName`	string	`workflow_xkkjf`	工作流名称。
- `status`	number	`2`	作业状态码。详情参见工作流 API 概览。
- `tasks`	object	-	任务详情。
- `id`	string	`f93dea90fc3a11ecac7e6da0662c195b`	任务 ID。
- `name`	string	`BFS`	任务名称。
- `runBeginTime`	datetime	`2022-07-12T10:33:35+08:00`	任务开始执行时间。
- `runEndTime`	datetime	`2022-07-12T10:33:38+08:00`	任务执行结束时间。
- `status`	number	`2`	任务状态。

返回示例


```
{
  "cookie": [],
  "Content-Type": "application/json",
  "Traceparent": "00-3db17c9fd9e0a4c3824973471523d214-4384705e523dce83-00",
  "Date": "Fri, 15 Jul 2022 09:08:20 GMT",
  "Content-Length": "400"
}
{
  "code": 0,
  "data": {
    "id": 1964,
    "name": "workflow_xkkjf_20220712103332",
    "workflowId": "3992429968",
    "workflowName": "workflow_xkkjf",
    "status": 2,
    "tasks": [
      {
        "id": "f93dea90fc3a11ecac7e6da0662c195b",
        "name": "BFS",
        "runBeginTime": "2022-07-12T10:33:35+08:00",
        "runEndTime": "2022-07-12T10:33:38+08:00",
        "status": 2
      }
    ],
    "runBeginTime": 1657593215000,
    "runEndTime": 1657593218000,
    "createTime": 1657593212505
  },
  "message": "Success"
}
```


最后更新: June 20, 2023

2022 vesoft Inc.

终止执行指定作业

本文介绍如何使用工作流的 API 终止执行指定作业。

API 路径

`api-open/v1/jobs/<job_id>/cancel`

`<job_id>`：作业 ID。参见下文的请求参数。

请求参数

路径参数

参数	类型	是否必填	默认值	示例	说明
`job_id`	number	必填	-	`1964`	作业 ID。可以通过 API 获取所有作业列表查询，或者在作业列表页面查看。

Headers 参数

参数	类型	是否必填	默认值	示例	说明
`Content-Type`	string	必填	-	`application/x-www-form-urlencoded`	内容类型。
`explorer_token`	string	必填	-	`eyJhbxxx`	授权 Token，用于验证账号信息。如何获取授权 Token 请参见工作流 API 概览。

Body 参数

无。

请求示例

```
curl -i -X PUT -H "Content-Type: application/x-www-form-urlencoded" -H "Cookie: "explorer_token=eyJhbxxx"" http://192.168.8.145:7002/api-open/v1/jobs/30600/cancel
```

返回参数

参数	类型	示例	说明
`code`	number	`0`	请求结果码。请求成功返回 `0`，请求不成功返回对应的错误码。详情参见工作流 API 概览。
`message`	string	`Success`	执行结果信息。
`data`	object	-	返回的数据列表。
- `success`	bool	`true`	是否成功终止。

返回示例

```
{
  "cookie": [],
  "Content-Type": "application/json",
  "Traceparent": "00-8b4b47413a211d9b5e0839aadc712052-4a98bae37fe5948a-00",
  "Date": "Mon, 18 Jul 2022 01:45:08 GMT",
  "Content-Length": "54"
}
{
  "code": 0,
  "data": {
    "success": true
  },
  "message": "Success"
}
```


2022 vesoft Inc.

最后更新: June 20, 2023

2022 vesoft Inc.

获取指定任务的运行结果

本文介绍如何使用工作流的 API 获取指定任务的运行结果。

API 路径

`api-open/v1/jobs/<job_id>/tasks/<task_id>/sample_result`

- `<job_id>`：作业 ID。参见下文的请求参数。
- `<task_id>`：任务 ID。参见下文的请求参数。

请求参数

路径参数

参数	类型	是否必填	默认值	示例	说明
`job_id`	number	必填	-	`29987`	作业 ID。可以通过 API 获取所有作业列表查询，或者在作业列表页面查看。
`task_id`	number	必填	-	`8c171f70fb6f11ecac7e6da0662c195b`	作业 ID。可以通过其他 API 查询到，或者在指定作业页面单击组件，在右上角查看。

Headers 参数

参数	类型	是否必填	默认值	示例	说明
`Content-Type`	string	必填	-	`application/x-www-form-urlencoded`	内容类型。
`explorer_token`	string	必填	-	`eyJhbxxx`	授权 Token，用于验证账号信息。如何获取授权 Token 请参见工作流 API 概览。

Body 参数

参数	类型	是否必填	默认值	示例	说明
`limit`	number	必填	`10`	-	返回结果行数限制。

请求示例

```
curl -i -X GET -H "Content-Type: application/x-www-form-urlencoded" -H "Cookie: "explorer_token=eyJhbxxx"" http://192.168.8.145:7002/api-open/v1/jobs/29987/tasks/
8c171f70fb6f11ecac7e6da0662c195b/sample_result?limit=1000
```

返回参数

参数	类型	示例	说明
`code`	number	`0`	请求结果码。请求成功返回 `0`，请求不成功返回对应的错误码。详情参见工作流 API 概览。
`message`	string	`Success`	执行结果信息。
`data`	object	-	返回的数据列表。
- `items`	list	-	详细结果列表。
- `result`	string	`"player110","0.150000"`	根据算法不同，结果可能是 2 列或 3 列。

2022 vesoft Inc.

返回示例


```
{
  "cookie": [],
  "Content-Type": "application/json",
  "Traceparent": "00-14047b04b6810be06be22e010f500506-4c310a844b824a7f-00",
  "Date": "Fri, 15 Jul 2022 09:36:56 GMT",
  "Content-Length": "2014"
}
{
  "code": 0,
  "data": {
    "items": [
      [
        "player110",
        "0.150000"
      ],
      [
        "team219",
        "0.452126"
      ],
      ......
      [
        "player121",
        "0.262148"
      ]
    ]
  },
  "message": "Success"
}
```


最后更新: June 20, 2023

2022 vesoft Inc.

17.9 基本操作和快捷键

本文列出了 Explorer 的基本操作及快捷键。

17.9.1 基本操作

操作	说明
移动画布	在画布空白处鼠标左键拖拽移动。
放大或缩小画布	鼠标滚轮向下放大画布，向上缩小画布。
选中单个点或单条边	鼠标左键单击节点或边。
选中多个点或多条边	同时按住 Shift 和鼠标左键单击点或边。
批量选中数据	按住鼠标右键并框选数据；或者同时按住 Shift 和鼠标左键，并框选数据。
移动单个点	选中点按住鼠标左键拖拽。

17.9.2 快捷键

操作	说明
Enter	拓展
Shift + '-'	缩小
Shift + '+'	放大
Shift + 'l'	显示
Ctrl/Cmd + 'z'	撤销
Ctrl/Cmd + Shift + 'z'	回退
Ctrl/Cmd + 'a'	选中所有点
选中 + 'del'	隐藏
选中 + Shift + 'del'	隐藏其他

最后更新: June 20, 2023

2022 vesoft Inc.

18. Nebula Importer

18.1 Nebula Importer

Nebula Importer（简称 Importer）是一款 Nebula Graph 的 CSV 文件单机导入工具。Importer 可以读取本地的 CSV 文件，然后导入数据至 Nebula Graph 图数据库中。

18.1.1 适用场景

Importer 适用于将本地 CSV 文件的内容导入至 Nebula Graph 中。

18.1.2 优势

- 轻量快捷：不需要复杂环境即可使用，快速导入数据。
- 灵活筛选：通过配置文件可以实现对 CSV 文件数据的灵活筛选。

18.1.3 更新说明

Release notes

18.1.4 前提条件

在使用 Nebula Importer 之前，请确保：

- 已部署 Nebula Graph 服务。目前有三种部署方式：
- Docker Compose 部署
- RPM/DEB 包安装
- 源码编译安装

- Nebula Graph 中已创建 Schema，包括图空间、Tag 和 Edge type，或者通过参数 `clientSettings.postStart.commands` 设置。
- 运行 Importer 的机器已部署 Golang 环境。详情请参见 Golang 环境搭建。

18.1.5 操作步骤

配置 yaml 文件并准备好待导入的 CSV 文件，即可使用本工具向 Nebula Graph 批量写入数据。

下载二进制包运行

1. 在Release页面下载二进制包，并添加执行权限。
2. 启动服务。

```
$ ./<binary_package_name> --config <yaml_config_file_path>
```

源码编译运行

1. 克隆仓库。

```
$ git clone -b release-3.1 https://github.com/vesoft-inc/nebula-importer.git
```


2022 vesoft Inc.

> **Note**
>
> 请使用正确的分支。 Nebula Graph 2.x 和 3.x 的 rpc 协议不同。

2. 进入目录 `nebula-importer`。

```
$ cd nebula-importer
```

3. 编译源码。

```
$ make build
```

4. 启动服务。

```
$ ./nebula-importer --config <yaml_config_file_path>
```

> **Note**
>
> yaml 配置文件说明请参见下文的配置文件说明。

无网络编译方式

如果服务器不能联网，建议在能联网的机器上将源码和各种依赖打包上传到对应的服务器上编译即可，操作步骤如下：

1. 克隆仓库。

```
$ git clone -b release-3.1 https://github.com/vesoft-inc/nebula-importer.git
```

2. 使用如下的命令下载并打包依赖的源码。

```
$ cd nebula-importer
$ go mod vendor
$ cd .. && tar -zcvf nebula-importer.tar.gz nebula-importer
```

3. 将压缩包上传到不能联网的服务器上。
4. 解压并编译。

```
$ tar -zxvf nebula-importer.tar.gz
$ cd nebula-importer
$ go build -mod vendor cmd/importer.go
```

Docker 方式运行

使用 Docker 可以不必在本地安装 Go 语言环境，只需要拉取 Nebula Importer 的镜像，并将本地配置文件和 CSV 数据文件挂载到容器中。命令如下：

```
$ docker run --rm -ti \
    --network=host \
    -v <config_file>:<config_file> \
    -v <csv_data_dir>:<csv_data_dir> \
    vesoft/nebula-importer:<version>
    --config <config_file>
```

- `<config_file>`：本地 yaml 配置文件的绝对路径。
- `<csv_data_dir>`：本地 CSV 数据文件的绝对路径。
- `<version>`：Nebula Graph 3.x 请填写 `v3`。

2022 vesoft Inc.

> **Note**
>
> 建议使用相对路径。如果使用本地绝对路径，请检查路径映射到 Docker 中的路径。

18.1.6 配置文件说明

Nebula Importer 通过 `nebula-importer/examples/v2/example.yaml` 配置文件来描述待导入文件信息、Nebula Graph 服务器信息等。用户可以参考示例配置文件：无表头配置/有表头配置。下文将分类介绍配置文件内的字段。

> **Note**
>
> 如果用户下载的是二进制包，请手动创建配置文件。

基本配置

示例配置如下：

```
version: v3
description: example
removeTempFiles: false
```

参数	默认值	是否必须	说明
`version`	v3	是	目标 Nebula Graph 的版本。
`description`	example	否	配置文件的描述。
`removeTempFiles`	false	否	是否删除临时生成的日志和错误数据文件。

客户端配置

客户端配置存储客户端连接 Nebula Graph 相关的配置。

示例配置如下：

```
clientSettings:
  retry: 3
  concurrency: 10
  channelBufferSize: 128
  space: test
  connection:
    user: user
    password: password
    address: 192.168.11.13:9669,192.168.11.14:9669
  # # 只有 local_config 是 false 的情况下，才可以通过 UPDATE CONFIGS 更新配置
  # postStart:
  #   commands: |
  #     UPDATE CONFIGS storage:wal_ttl=3600;
  #     UPDATE CONFIGS storage:rocksdb_column_family_options = { disable_auto_compactions = true };
  #   afterPeriod: 8s
  # preStop:
  #   commands: |
```


2022 vesoft Inc.

```
#     UPDATE CONFIGS storage:wal_ttl=86400;
#     UPDATE CONFIGS storage:rocksdb_column_family_options = { disable_auto_compactions = false };
```

参数	默认值	是否必须	说明
`clientSettings.retry`	3	否	nGQL 语句执行失败的重试次数。
`clientSettings.concurrency`	10	否	Nebula Graph 客户端并发数。
`clientSettings.channelBufferSize`	128	否	每个 Nebula Graph 客户端的缓存队列大小。
`clientSettings.space`	-	是	指定数据要导入的 Nebula Graph 图空间。不要同时导入多个空间，以免影响性能。
`clientSettings.connection.user`	-	是	Nebula Graph 的用户名。
`clientSettings.connection.password`	-	是	Nebula Graph 用户名对应的密码。
`clientSettings.connection.address`	-	是	所有 Graph 服务的地址和端口。
`clientSettings.postStart.commands`	-	否	配置连接 Nebula Graph 服务器之后，在插入数据之前执行的一些操作。
`clientSettings.postStart.afterPeriod`	-	否	执行上述 `commands` 命令后到执行插入数据命令之间的间隔，例如 `8s` 。
`clientSettings.preStop.commands`	-	否	配置断开 Nebula Graph 服务器连接之前执行的一些操作。

文件配置

文件配置存储数据文件和日志的相关配置，以及 Schema 的具体信息。

文件和日志配置

示例配置如下：

```
logPath: ./err/test.log
files:
  - path: ./student_without_header.csv
    failDataPath: ./err/studenterr.csv
    batchSize: 128
    limit: 10
    inOrder: false
    type: csv
    csv:
      withHeader: false
```


2022 vesoft Inc.

```
    withLabel: false
    delimiter: ","
```

参数	默认值	是否必须	说明
`logPath`	-	否	导入过程中的错误等日志信息输出的文件路径。
`files.path`	-	是	数据文件的存放路径，如果使用相对路径，则会将路径和当前配置文件的目录拼接。可以使用星号（*）进行模糊匹配，导入多个名称相似的文件，但是文件的结构需要相同。
`files.failDataPath`	-	是	插入失败的数据文件存放路径，以便后面补写数据。
`files.batchSize`	128	否	单批次插入数据的语句数量。
`files.limit`	-	否	读取数据的行数限制。
`files.inOrder`	-	否	是否按顺序在文件中插入数据行。如果为 `false`，可以避免数据倾斜导致的导入速率降低。
`files.type`	-	是	文件类型。
`files.csv.withHeader`	`false`	是	是否有表头。详情请参见关于 CSV 文件表头。
`files.csv.withLabel`	`false`	是	是否有 LABEL。详情请参见有表头配置说明。
`files.csv.delimiter`	`","`	是	指定 csv 文件的分隔符。只支持一个字符的字符串分隔符。

SCHEMA 配置

Schema 配置描述当前数据文件的 Meta 信息，Schema 的类型分为点和边两类，可以同时配置多个点或边。

- 点配置

示例配置如下：

```
schema:
  type: vertex
  vertex:
    vid:
      type: string
      index: 0
    tags:
      - name: student
        props:
          - name: name
            type: string
            index: 1
          - name: age
            type: int
            index: 2
          - name: gender
            type: string
            index: 3
```

参数	默认值	是否必须	说明
`files.schema.type`	-	是	Schema 的类型，可选值为 `vertex` 和 `edge`。
`files.schema.vertex.vid.type`	-	否	点 ID 的数据类型，可选值为 `int` 和 `string`。
`files.schema.vertex.vid.index`	-	否	点 ID 对应 CSV 文件中列的序号。
`files.schema.vertex.tags.name`	-	是	Tag 名称。
`files.schema.vertex.tags.props.name`	-	是	Tag 属性名称，必须和 Nebula Graph 中的 Tag 属性一致。
`files.schema.vertex.tags.props.type`	-	是	属性数据类型，支持 `bool`、`int`、`float`、`double`、`timestamp` 和 `string`。
`files.schema.vertex.tags.props.index`	-	否	属性对应 CSV 文件中列的序号。

2022 vesoft Inc.

Note

CSV 文件中列的序号从 0 开始，即第一列的序号为 0，第二列的序号为 1。

- 边配置

示例配置如下：

```
schema:
  type: edge
  edge:
    name: follow
    withRanking: true
    srcVID:
      type: string
      index: 0
    dstVID:
      type: string
      index: 1
    rank:
      index: 2
    props:
      - name: degree
        type: double
        index: 3
```

参数	默认值	是否必须	说明
`files.schema.type`	-	是	Schema 的类型，可选值为 `vertex` 和 `edge` 。
`files.schema.edge.name`	-	是	Edge type 名称。
`files.schema.edge.srcVID.type`	-	否	边的起始点 ID 的数据类型。
`files.schema.edge.srcVID.index`	-	否	边的起始点 ID 对应 CSV 文件中列的序号。
`files.schema.edge.dstVID.type`	-	否	边的目的点 ID 的数据类型。
`files.schema.edge.dstVID.index`	-	否	边的目的点 ID 对应 CSV 文件中列的序号。
`files.schema.edge.rank.index`	-	否	边的 rank 值对应 CSV 文件中列的序号。
`files.schema.edge.props.name`	-	是	Edge type 属性名称，必须和 Nebula Graph 中的 Edge type 属性一致。
`files.schema.edge.props.type`	-	是	属性类型，支持 `bool` 、 `int` 、 `float` 、 `double` 、 `timestamp` 和 `string` 。
`files.schema.edge.props.index`	-	否	属性对应 CSV 文件中列的序号。

18.1.7 关于 CSV 文件表头（header）

Importer 根据 CSV 文件有无表头，需要对配置文件进行不同的设置，相关示例和说明请参见：

- 无表头配置说明
- 有表头配置说明

18.1.8 视频

- 数据库导入工具——Nebula Importer 简介（3 分 09 秒）

最后更新: June 20, 2023

2022 vesoft Inc.

18.2 有表头配置说明

对于有表头（header）的 CSV 文件，需要在配置文件里设置 `withHeader` 为 `true`，表示 CSV 文件中第一行为表头，表头内容具有特殊含义。

Caution

如果 CSV 文件中含有 header，Importer 就会按照 header 来解析每行数据的 Schema，并忽略 yaml 文件中的点或边设置。

18.2.1 示例文件

有表头的 CSV 文件示例如下：

- 点示例

  `student_with_header.csv` 的示例数据：

  ```
  :VID(string),student.name:string,student.age:int,student.gender:string
  student100,Monica,16,female
  student101,Mike,18,male
  student102,Jane,17,female
  ```

  第一列为点 ID，后面三列为属性 `name`、`age` 和 `gender`。

- 边示例

  `follow_with_header.csv` 的示例数据：

  ```
  :SRC_VID(string),:DST_VID(string),:RANK,follow.degree:double
  student100,student101,0,92.5
  student101,student100,1,85.6
  student101,student102,2,93.2
  student100,student102,1,96.2
  ```

  前两列的数据分别为起始点 ID 和目的点 ID，第三列为 rank，第四列为属性 `degree`。

18.2.2 表头格式说明

表头通过一些关键词定义起始点、目的点、rank 以及一些特殊功能，说明如下：

- `:VID`（必填）：点 ID。需要用 `:VID(type)` 形式设置数据类型，例如 `:VID(string)` 或 `:VID(int)`。
- `:SRC_VID`（必填）：边的起始点 ID。需要用 `:SRC_VID(type)` 形式设置数据类型。
- `:DST_VID`（必填）：边的目的点 ID。需要用 `:DST_VID(type)` 形式设置数据类型。
- `:RANK`（可选）：边的 rank 值。
- `:IGNORE`（可选）：插入数据时忽略这一列。
- `:LABEL`（可选）：表示对该行进行插入（+）或删除（-）操作。必须为第一列。例如：

  ```
  :LABEL,
  +,
  -,
  ```

Note

除了 `:LABEL` 列之外的所有列都可以按任何顺序排序，因此针对较大的 CSV 文件，用户可以灵活地设置 header 来选择需要的列。

2022 vesoft Inc.

对于 Tag 或 Edge type 的属性，格式为 `<tag_name/edge_name>.<prop_name>:<prop_type>`，说明如下：

- `<tag_name/edge_name>`：Tag 或者 Edge type 的名称。
- `<prop_name>`：属性名称。
- `<prop_type>`：属性类型。支持 `bool`、`int`、`float`、`double`、`timestamp` 和 `string`，默认为 `string`。

例如 `student.name:string`、`follow.degree:double`。

18.2.3 配置示例

```
# 连接的 Nebula Graph 版本，连接 3.x 时设置为 v3。
version: v3

description: example

# 是否删除临时生成的日志和错误数据文件。
removeTempFiles: false

clientSettings:

  # nGQL 语句执行失败的重试次数。
  retry: 3

  # Nebula Graph 客户端并发数。
  concurrency: 10

  # 每个 Nebula Graph 客户端的缓存队列大小。
  channelBufferSize: 128

  # 指定数据要导入的 Nebula Graph 图空间。
  space: student

  # 连接信息。
  connection:
    user: root
    password: nebula
    address: 192.168.11.13:9669

  postStart:
    # 配置连接 Nebula Graph 服务器之后，在插入数据之前执行的一些操作。
    commands: |
      DROP SPACE IF EXISTS student;
      CREATE SPACE IF NOT EXISTS student(partition_num=5, replica_factor=1, vid_type=FIXED_STRING(20));
      USE student;
      CREATE TAG student(name string, age int,gender string);
      CREATE EDGE follow(degree int);

    # 执行上述命令后到执行插入数据命令之间的间隔。
    afterPeriod: 15s

  preStop:
    # 配置断开 Nebula Graph 服务器连接之前执行的一些操作。
    commands: |

# 错误等日志信息输出的文件路径。
logPath: ./err/test.log

# CSV 文件相关设置。
files:

    # 数据文件的存放路径，如果使用相对路径，则会将路径和当前配置文件的目录拼接。本示例第一个数据文件为点的数据。
  - path: ./student_with_header.csv

    # 插入失败的数据文件存放路径，以便后面补写数据。
    failDataPath: ./err/studenterr.csv

    # 单批次插入数据的语句数量。
    batchSize: 10

    # 读取数据的行数限制。
    limit: 10

    # 是否按顺序在文件中插入数据行。如果为 false，可以避免数据倾斜导致的导入速率降低。
    inOrder: true

    # 文件类型，当前仅支持 csv。
    type: csv

    csv:
      # 是否有表头。
      withHeader: true

      # 是否有 LABEL。
      withLabel: false

      # 指定 csv 文件的分隔符。只支持一个字符的字符串分隔符。
```


2022 vesoft Inc.

```
    delimiter: ","

  schema:
    # Schema 的类型，可选值为 vertex 和 edge。
    type: vertex

  # 本示例第二个数据文件为边的数据。
- path: ./follow_with_header.csv
  failDataPath: ./err/followerr.csv
  batchSize: 10
  limit: 10
  inOrder: true
  type: csv
  csv:
    withHeader: true
    withLabel: false
  schema:
    # Schema 的类型为 edge。
    type: edge
    edge:
      # Edge type 名称。
      name: follow

      # 是否包含 rank。
      withRanking: true
```



点 ID 的数据类型需要和 `clientSettings.postStart.commands` 中的创建图空间语句的数据类型一致。

最后更新: June 20, 2023

2022 vesoft Inc.

18.3 无表头配置说明

对于无表头（header）的 CSV 文件，需要在配置文件里设置 `withHeader` 为 `false`，表示 CSV 文件中只含有数据（不含第一行表头），同时可能还需要设置数据类型、对应的列等。

18.3.1 示例文件

无表头的 CSV 文件示例如下：

- 点示例

  `student_without_header.csv` 的示例数据：

  ```
  student100,Monica,16,female
  student101,Mike,18,male
  student102,Jane,17,female
  ```

  第一列为点 ID，后面三列为属性 `name` 、`age` 和 `gender` 。

- 边示例

  `follow_without_header.csv` 的示例数据：

  ```
  student100,student101,0,92.5
  student101,student100,1,85.6
  student101,student102,2,93.2
  student100,student102,1,96.2
  ```

  前两列的数据分别为起始点 ID 和目的点 ID，第三列为 rank，第四列为属性 `degree` 。

18.3.2 配置示例

```
# 连接的 Nebula Graph 版本，连接 3.x 时设置为 v3。
version: v3

description: example

# 是否删除临时生成的日志和错误数据文件。
removeTempFiles: false

clientSettings:

  # nGQL 语句执行失败的重试次数。
  retry: 3

  # Nebula Graph 客户端并发数。
  concurrency: 10

  # 每个 Nebula Graph 客户端的缓存队列大小。
  channelBufferSize: 128

  # 指定数据要导入的 Nebula Graph 图空间。
  space: student

  # 连接信息。
  connection:
    user: root
    password: nebula
    address: 192.168.11.13:9669

  postStart:
    # 配置连接 Nebula Graph 服务器之后，在插入数据之前执行的一些操作。
    commands: |
      DROP SPACE IF EXISTS student;
      CREATE SPACE IF NOT EXISTS student(partition_num=5, replica_factor=1, vid_type=FIXED_STRING(20));
      USE student;
      CREATE TAG student(name string, age int,gender string);
      CREATE EDGE follow(degree int);

    # 执行上述命令后到执行插入数据命令之间的间隔。
    afterPeriod: 15s

  preStop:
    # 配置断开 Nebula Graph 服务器连接之前执行的一些操作。
    commands: |
```


2022 vesoft Inc.

```
# 错误等日志信息输出的文件路径。
logPath: ./err/test.log

# CSV 文件相关设置。
files:

    # 数据文件的存放路径，如果使用相对路径，则会将路径和当前配置文件的目录拼接。本示例第一个数据文件为点的数据。
  - path: ./student_without_header.csv

    # 插入失败的数据文件存放路径，以便后面补写数据。
    failDataPath: ./err/studenterr.csv

    # 单批次插入数据的语句数量。
    batchSize: 10

    # 读取数据的行数限制。
    limit: 10

    # 是否按顺序在文件中插入数据行。如果为 false，可以避免数据倾斜导致的导入速率降低。
    inOrder: true

    # 文件类型，当前仅支持 csv。
    type: csv

    csv:
      # 是否有表头。
      withHeader: false

      # 是否有 LABEL。
      withLabel: false

      # 指定 csv 文件的分隔符。只支持一个字符的字符串分隔符。
      delimiter: ","

    schema:
      # Schema 的类型，可选值为 vertex 和 edge。
      type: vertex

      vertex:

        # 点 ID 设置。
        vid:
           # 点 ID 对应 CSV 文件中列的序号。CSV 文件中列的序号从 0 开始。
           index: 0

           # 点 ID 的数据类型，可选值为 int 和 string，分别对应 Nebula Graph 中的 INT64 和 FIXED_STRING。
           type: string

        # Tag 设置。
        tags:
            # Tag 名称。
          - name: student

            # Tag 内的属性设置。
            props:
                # 属性名称。
              - name: name

                # 属性数据类型。
                type: string

                # 属性对应 CSV 文件中列的序号。
                index: 1

              - name: age
                type: int
                index: 2
              - name: gender
                type: string
                index: 3

    # 本示例第二个数据文件为边的数据。
  - path: ./follow_without_header.csv
    failDataPath: ./err/followerr.csv
    batchSize: 10
    limit: 10
    inOrder: true
    type: csv
    csv:
      withHeader: false
      withLabel: false
    schema:
      # Schema 的类型为 edge。
      type: edge
      edge:
        # Edge type 名称。
        name: follow

        # 是否包含 rank。
        withRanking: true

        # 起始点 ID 设置。
        srcVID:
           # 数据类型。
```


2022 vesoft Inc.

```
      type: string

      # 起始点 ID 对应 CSV 文件中列的序号。
      index: 0

  # 目的点 ID 设置。
  dstVID:
      type: string
      index: 1

  # rank 设置。
  rank:
      # rank 值对应 CSV 文件中列的序号。如果没有设置 index，请务必在第三列设置 rank 的值。之后的列依次设置各属性。
      index: 2

  # Edge type 内的属性设置。
  props:
       # 属性名称。
      - name: degree

        # 属性数据类型。
        type: double

        # 属性对应 CSV 文件中列的序号。
        index: 3
```

Note

- CSV 文件中列的序号从 0 开始，即第一列的序号为 0，第二列的序号为 1。
- 点 ID 的数据类型需要和 `clientSettings.postStart.commands` 中的创建图空间语句的数据类型一致。
- 如果没有设置 index 字段指定列的序号，CSV 文件必须遵守如下规则：
- 在点数据文件中，第一列必须为点 ID，后面的列为属性，且需要和配置文件内的顺序一一对应。
- 在边数据文件中，第一列必须为起始点 ID，第二列必须为目的点 ID，如果 `withRanking` 为 `true`，第三列必须为 rank 值，后面的列为属性，且需要和配置文件内的顺序一一对应。

最后更新: June 20, 2023

2022 vesoft Inc.

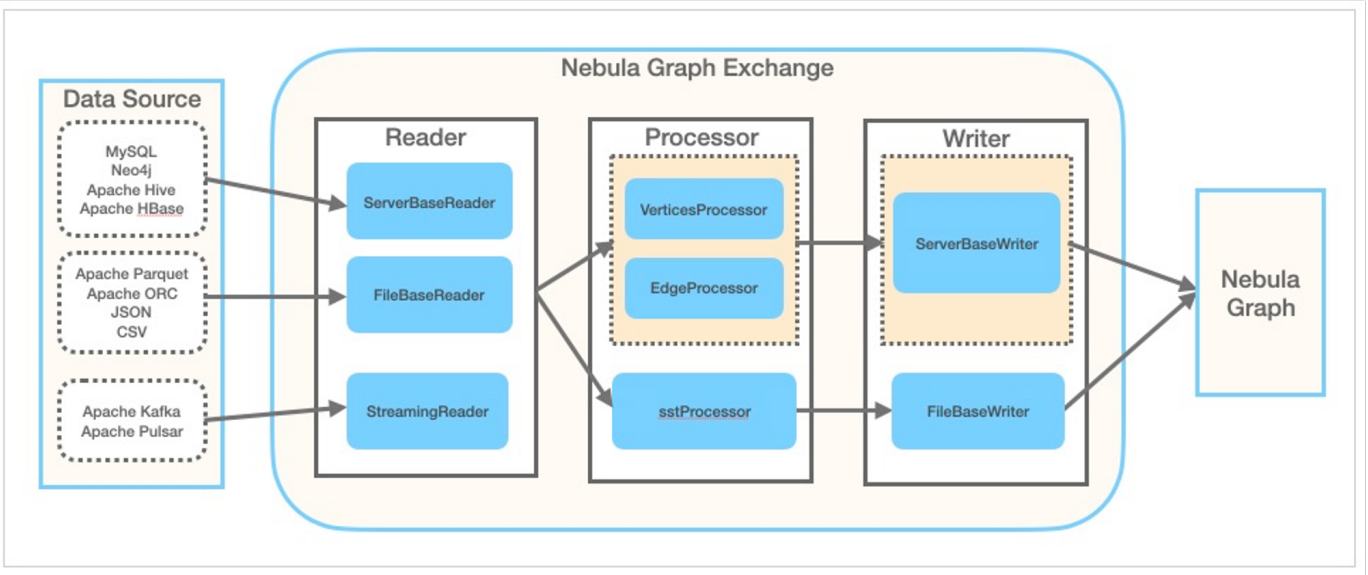
19. Nebula Exchange

19.1 认识 Nebula Exchange

19.1.1 什么是 Nebula Exchange

Nebula Exchange（简称 Exchange）是一款 Apache Spark™ 应用，用于在分布式环境中将集群中的数据批量迁移到 Nebula Graph 中，能支持多种不同格式的批式数据和流式数据的迁移。

Exchange 由 Reader、Processor 和 Writer 三部分组成。Reader 读取不同来源的数据返回 DataFrame 后，Processor 遍历 DataFrame 的每一行，根据配置文件中 `fields` 的映射关系，按列名获取对应的值。在遍历指定批处理的行数后，Writer 会将获取的数据一次性写入到 Nebula Graph 中。下图描述了 Exchange 完成数据转换和迁移的过程。



版本系列

Exchange 有社区版和企业版两个系列，二者功能不同。社区版在 GitHub 开源开发，企业版属于 Nebula Graph 企业套餐，详情参见版本对比。

适用场景

Exchange 适用于以下场景：

- 需要将来自 Kafka、Pulsar 平台的流式数据，如日志文件、网购数据、游戏内玩家活动、社交网站信息、金融交易或地理空间服务，以及来自数据中心内所连接设备或仪器的遥测数据等转化为属性图的点或边数据，并导入 Nebula Graph 数据库。
- 需要从关系型数据库（如 MySQL）或者分布式文件系统（如 HDFS）中读取批式数据，如某个时间段内的数据，将它们转化为属性图的点或边数据，并导入 Nebula Graph 数据库。
- 需要将大批量数据生成 Nebula Graph 能识别的 SST 文件，再导入 Nebula Graph 数据库。
- 需要导出 Nebula Graph 中保存的数据。

> **nterpriseonly**
>
> 仅企业版 Exchange 支持从 Nebula Graph 中导出数据。

2022 vesoft Inc.

产品优点

Exchange 具有以下优点：

- 适应性强：支持将多种不同格式或不同来源的数据导入 Nebula Graph 数据库，便于迁移数据。
- 支持导入 SST：支持将不同来源的数据转换为 SST 文件，用于数据导入。
- 支持 SSL 加密：支持在 Exchange 与 Nebula Graph 之间建立 SSL 加密传输通道，保障数据安全。
- 支持断点续传：导入数据时支持断点续传，有助于节省时间，提高数据导入效率。

> **Note**
>
> 目前仅迁移 Neo4j 数据时支持断点续传。

- 异步操作：会在源数据中生成一条插入语句，发送给 Graph 服务，最后再执行插入操作。
- 灵活性强：支持同时导入多个 Tag 和 Edge type，不同 Tag 和 Edge type 可以是不同的数据来源或格式。
- 统计功能：使用 Apache Spark™ 中的累加器统计插入操作的成功和失败次数。
- 易于使用：采用 HOCON（Human-Optimized Config Object Notation）配置文件格式，具有面向对象风格，便于理解和操作。

数据源

Exchange 3.0.0 支持将以下格式或来源的数据转换为 Nebula Graph 能识别的点和边数据，然后通过 nGQL 语句的形式导入 Nebula Graph：

- 存储在 HDFS 或本地的数据：
- Apache Parquet
- Apache ORC
- JSON
- CSV
- Apache HBase™
- 数据仓库：
- Hive
- MaxCompute
- 图数据库：Neo4j（Client 版本 2.4.5-M1）
- 关系型数据库：
- MySQL
- PostgreSQL
- 列式数据库：ClickHouse
- 流处理软件平台：Apache Kafka®
- 发布/订阅消息平台：Apache Pulsar 2.4.5

除了用 nGQL 语句的形式导入数据，Exchange 还支持将数据源的数据生成 SST 文件，然后通过 Console 导入 SST 文件。

此外，企业版 Exchange 支持以 Nebula Graph 为源，将数据导出到 CSV 文件。

2022 vesoft Inc.

更新说明

Release

视频

- 图数据库 Nebula Graph 数据导入工具——Exchange（3 分 08 秒）

最后更新: June 20, 2023

2022 vesoft Inc.

19.1.2 使用限制

本文描述 Exchange 3.0.0 的一些使用限制。

版本兼容性

Nebula Exchange 版本（即 JAR 包版本）和 Nebula Graph 内核的版本对应关系如下。

Exchange client 版本	**Nebula Graph** 版本
3.0-SNAPSHOT	nightly
3.0.0	3.1.0
2.6.x	2.6.x
2.5.x	2.5.x
2.1.0	2.0.0、2.0.1
2.0.1	2.0.0、2.0.1
2.0.0	2.0.0、2.0.1

JAR 包有两种获取方式：自行编译或者从 maven 仓库下载。

使用环境

Exchange 3.x 支持以下操作系统：

- CentOS 7
- macOS

2022 vesoft Inc.

软件依赖

为保证 Exchange 正常工作，请确认机器上已经安装如下软件：

- Java 1.8 版本
- Scala 2.10.7、2.11.12 或 2.12.10 版本
- Apache Spark。使用 Exchange 从不同数据源导出数据对 Spark 版本的要求如下：

> **Note**
>
> 使用 Exchange 时，需根据 Spark 版本选择相应的 JAR 文件。例如，当 Spark 版本为 2.4 时，选择 nebula-exchange_spark_2.4-3.0.0.jar。

数据源	**Spark 2.2**	**Spark 2.4**	**Spark 3**
CSV 文件	支持	支持	支持
JSON 文件	支持	支持	支持
ORC 文件	支持	支持	支持
Parquet 文件	支持	支持	支持
HBase	支持	支持	支持
MySQL	支持	支持	支持
PostgreSQL	支持	支持	支持
ClickHouse	支持	支持	支持
Neo4j	不支持	支持	不支持
Hive	支持	支持	支持
MaxCompute	不支持	支持	不支持
Pulsar	不支持	支持	未测试
Kafka	不支持	支持	未测试
Nebula Graph	不支持	支持	不支持

在以下使用场景，还需要部署 Hadoop Distributed File System (HDFS)：

- 迁移 HDFS 的数据
- 生成 SST 文件

最后更新: June 20, 2023

2022 vesoft Inc.

19.2 获取 Nebula Exchange

本文介绍如何获取 Nebula Exchange 的 JAR 文件。

19.2.1 直接下载 JAR 文件

社区版 Exchange 的 JAR 文件可以直接下载。

要下载企业版 Exchange，需先获取 Nebula Graph 企业版套餐。

19.2.2 编译源代码获取 JAR 文件

社区版 Exchange 的 JAR 文件还可以通过编译源代码获取。下文介绍如何编译 Exchange 源代码。

Enterpriseonly

企业版 Exchange 仅能在 Nebula Graph 企业版套餐中获取。

前提条件

- 安装 Maven。
- 根据数据源安装需要的 Spark 版本，从各数据源导出数据支持的 Spark 版本参见软件依赖。

操作步骤

1. 在根目录克隆仓库 `nebula-exchange`。

```
git clone -b v3.0.0 https://github.com/vesoft-inc/nebula-exchange.git
```

2. 切换到目录 `nebula-exchange`。

```
cd nebula-exchange
```

3. 根据 Exchange 使用环境中的 Spark 版本打包 Exchange。

- Spark 2.2：

```
mvn clean package -Dmaven.test.skip=true -Dgpg.skip -Dmaven.javadoc.skip=true \
-pl nebula-exchange_spark_2.2 -am -Pscala-2.11 -Pspark-2.2
```

- Spark 2.4：

```
mvn clean package -Dmaven.test.skip=true -Dgpg.skip -Dmaven.javadoc.skip=true \
-pl nebula-exchange_spark_2.4 -am -Pscala-2.11 -Pspark-2.4
```

- Spark 3.0：

```
mvn clean package -Dmaven.test.skip=true -Dgpg.skip -Dmaven.javadoc.skip=true \
-pl nebula-exchange_spark_3.0 -am -Pscala-2.12 -Pspark-3.0
```

编译成功后，可以在 `nebula-exchange_spark_x.x/target/` 目录里找到 `nebula-exchange_spark_x.x-v3.0.0.jar` 文件。`x.x` 代表 Spark 版本，例如 `2.4`。

Note

JAR 文件版本号会因 Nebula Java Client 的发布版本而变化。用户可以在 Releases 页面查看最新版本。

2022 vesoft Inc.

迁移数据时，用户可以参考配置文件 `target/classes/application.conf`。

下载依赖包失败

如果编译时下载依赖包失败：

- 检查网络设置，确认网络正常。
- 修改 Maven 安装目录下 `libexec/conf/settings.xml` 文件的 `mirror` 部分：

```
<mirror>
 <id>alimaven</id>
 <mirrorOf>central</mirrorOf>
 <name>aliyun maven</name>
 <url>http://maven.aliyun.com/nexus/content/repositories/central/</url>
</mirror>
```

最后更新: June 20, 2023

2022 vesoft Inc.

19.3 参数说明

19.3.1 导入命令参数

完成配置文件修改后，可以运行以下命令将指定来源的数据导入 Nebula Graph 数据库。

- 首次导入

```
<spark_install_path>/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange <nebula-exchange-2.x.y.jar_path> -c <application.conf_path>
```

- 导入 reload 文件

  如果首次导入时有一些数据导入失败，会将导入失败的数据存入 reload 文件，可以用参数 `-r` 尝试导入 reload 文件。

```
<spark_install_path>/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange <nebula-exchange-2.x.y.jar_path> -c <application.conf_path> -r "<reload_file_path>"
```

> **Note**
>
> JAR 文件版本号以实际编译得到的 JAR 文件名称为准。

> **Note**
>
> 如果使用 yarn-cluster 模式提交任务，请参考如下示例：

```
$SPARK_HOME/bin/spark-submit     --master yarn-cluster \
--class com.vesoft.nebula.exchange.Exchange \
--files application.conf \
--conf spark.driver.extraClassPath=./ \
--conf spark.executor.extraClassPath=./ \
nebula-exchange-3.0.0.jar \
-c application.conf
```

下表列出了命令的相关参数。

参数	是否必需	默认值	说明
`--class`	是	无	指定驱动的主类。
`--master`	是	无	指定 Spark 集群中 master 进程的 URL。详情请参见 master-urls。
`-c` / `--config`	是	无	指定配置文件的路径。
`-h` / `--hive`	否	`false`	添加这个参数表示支持从 Hive 中导入数据。
`-D` / `--dry`	否	`false`	添加这个参数表示检查配置文件的格式是否符合要求，但不会校验 `tags` 和 `edges` 的配置项是否正确。正式导入数据时不能添加这个参数。
-r / --reload	否	无	指定需要重新加载的 reload 文件路径。

更多 Spark 的参数配置说明请参见 Spark Configuration。

最后更新: June 20, 2023

2022 vesoft Inc.

19.3.2 配置说明

本文介绍使用 Nebula Exchange 时如何修改配置文件 `application.conf`。

修改配置文件之前，建议根据数据源复制并修改文件名称，便于区分。例如数据源为 CSV 文件，可以复制为 `csv_application.conf`。

配置文件的内容主要分为如下几类：

- Spark 相关配置
- Hive 配置（可选）
- Nebula Graph 相关配置
- 点配置
- 边配置

Spark 相关配置

本文只列出部分 Spark 参数，更多参数请参见官方文档。

参数	数据类型	默认值	是否必须	说明
`spark.app.name`	string	-	否	Spark 驱动程序名称。
`spark.driver.cores`	int	`1`	否	驱动程序使用的 CPU 核数，仅适用于集群模式。
`spark.driver.maxResultSize`	string	`1G`	否	单个 Spark 操作（例如 collect）时，所有分区的序列化结果的总大小限制（字节为单位）。最小值为 1M，0 表示无限制。
`spark.executor.memory`	string	`1G`	否	Spark 驱动程序使用的内存量，可以指定单位，例如 512M、1G。
`spark.cores.max`	int	`16`	否	当驱动程序以“粗粒度”共享模式在独立部署集群或 Mesos 集群上运行时，跨集群（而非从每台计算机）请求应用程序的最大 CPU 核数。如果未设置，则值为 Spark 的独立集群管理器上的 `spark.deploy.defaultCores` 或 Mesos 上的 infinite（所有可用的内核）。

Hive 配置（可选）

如果 Spark 和 Hive 部署在不同集群，才需要配置连接 Hive 的参数，否则请忽略这些配置。

参数	数据类型	默认值	是否必须	说明
`hive.warehouse`	string	-	是	HDFS 中的 warehouse 路径。用双引号括起路径，以 `hdfs://` 开头。
`hive.connectionURL`	string	-	是	JDBC 连接的 URL。例如 `"jdbc:mysql://127.0.0.1:3306/hive_spark?characterEncoding=UTF-8"`。
`hive.connectionDriverName`	string	`"com.mysql.jdbc.Driver"`	是	驱动名称。
`hive.connectionUserName`	list[string]	-	是	连接的用户名。
`hive.connectionPassword`	list[string]	-	是	用户名对应的密码。

2022 vesoft Inc.

Nebula Graph 相关配置

2022 vesoft Inc.

参数	数据类型	默认值	是否必须	说明
`nebula.address.graph`	list[string]	`["127.0.0.1:9669"]`	是	所有 Graph 服务的地址，包括 IP 和端口，多个地址用英文逗号（,）分隔。格式为 `["ip1:port1","ip2:port2","ip3:port3"]`。
`nebula.address.meta`	list[string]	`["127.0.0.1:9559"]`	是	所有 Meta 服务的地址，包括 IP 和端口，多个地址用英文逗号（,）分隔。格式为 `["ip1:port1","ip2:port2","ip3:port3"]`。
`nebula.user`	string	-	是	拥有 Nebula Graph 写权限的用户名。
`nebula.pswd`	string	-	是	用户名对应的密码。
`nebula.space`	string	-	是	需要导入数据的的图空间名称。
`nebula.ssl.enable.graph`	bool	`false`	是	开启 Exchange 与 Graph 服务之间的 SSL 加密传输。当值为 `true` 时开启，下方的 SSL 相关参数生效。如果 Exchange 运行在多机集群上，在设置以下 SSL 相关路径时，需要在每台机器的相同路径都存储相应的文件。
`nebula.ssl.enable.meta`	bool	`false`	是	开启 Exchange 与 Meta 服务之间的 SSL 加密传输。当值为 `true` 时开启，下方的 SSL 相关参数生效。如果 Exchange 运行在多机集群上，在设置以下 SSL 相关路径时，需要在每台机器的相同路径都存储相应的文件。
`nebula.ssl.sign`	string	`ca`	是	签名方式，可选值：`ca`（CA 签名）或 `self`（自签名）。
`nebula.ssl.ca.param.caCrtFilePath`	string	`"/path/caCrtFilePath"`	是	`nebula.ssl.sign` 的值为 `ca` 时生效，用于指定 CA 证书的存储路径。
`nebula.ssl.ca.param.crtFilePath`	string	`"/path/crtFilePath"`	是	`nebula.ssl.sign` 的值为 `ca` 时生效，用于指定 CRT 证书的存储路径。
`nebula.ssl.ca.param.keyFilePath`	string	`"/path/keyFilePath"`	是	`nebula.ssl.sign` 的值为 `ca` 时生效，用于指定私钥文件的存储路径。
`nebula.ssl.self.param.crtFilePath`	string	`"/path/crtFilePath"`	是	`nebula.ssl.sign` 的值为 `self` 时生效，用于指定 CRT 证书的存储路径。
`nebula.ssl.self.param.keyFilePath`	string	`"/path/keyFilePath"`	是	`nebula.ssl.sign` 的值为 `self` 时生效，用于指定私钥文件的存储路径。
`nebula.ssl.self.param.password`	string	`"nebula"`	是	`nebula.ssl.sign` 的值为 `self` 时生效，用于指定密码文件的存储路径。
`nebula.path.local`	string	`"/tmp"`	否	导入 SST 文件时需要设置本地 SST 文件路径。
`nebula.path.remote`	string	`"/sst"`	否	导入 SST 文件时需要设置远端 SST 文件路径。
`nebula.path.hdfs.namenode`	string	`"hdfs://name_node:9000"`	否	导入 SST 文件时需要设置 HDFS 的 namenode。
`nebula.connection.timeout`	int	`3000`	否	Thrift 连接的超时时间，单位为 ms。
`nebula.connection.retry`	int	`3`	否	Thrift 连接重试次数。
`nebula.execution.retry`	int	`3`	否	nGQL 语句执行重试次数。

2022 vesoft Inc.

参数	数据类型	默认值	是否必须	说明
`nebula.error.max`	int	`32`	否	导入过程中的最大失败次数。当失败次数达到最大值时，提交的 Spark 作业将自动停止。
`nebula.error.output`	string	`/tmp/errors`	否	输出错误日志的路径。错误日志保存执行失败的 nGQL 语句。
`nebula.rate.limit`	int	`1024`	否	导入数据时令牌桶的令牌数量限制。
`nebula.rate.timeout`	int	`1000`	否	令牌桶中拿取令牌的超时时间，单位：毫秒。

点配置

对于不同的数据源，点的配置也有所不同，有很多通用参数，也有部分特有参数，配置时需要配置通用参数和不同数据源的特有参数。

通用参数

参数	数据类型	默认值	是否必须	说明
`tags.name`	string	-	是	Nebula Graph 中定义的 Tag 名称。
`tags.type.source`	string	-	是	指定数据源。例如 `csv` 。
`tags.type.sink`	string	`client`	是	指定导入方式，可选值为 `client` 和 `SST` 。
`tags.fields`	list[string]	-	是	属性对应的列的表头或列名。如果有表头或列名，请直接使用该名称。如果 CSV 文件没有表头，用 `[_c0, _c1, _c2]` 的形式表示第一列、第二列、第三列，以此类推。
`tags.nebula.fields`	list[string]	-	是	Nebula Graph 中定义的属性名称，顺序必须和 `tags.fields` 一一对应。例如 `[_c1, _c2]` 对应 `[name, age]` ，表示第二列为属性 name 的值，第三列为属性 age 的值。
`tags.vertex.field`	string	-	是	点 ID 的列。例如 CSV 文件没有表头时，可以用 `_c0` 表示第一列的值作为点 ID。
`tags.batch`	int	`256`	是	单批次写入 Nebula Graph 的最大点数量。
`tags.partition`	int	`32`	是	Spark 分片数量。

PARQUET/JSON/ORC 源特有参数

参数	数据类型	默认值	是否必须	说明
`tags.path`	string	-	是	HDFS 中点数据文件的路径。用双引号括起路径，以 `hdfs://` 开头。

CSV 源特有参数

参数	数据类型	默认值	是否必须	说明
`tags.path`	string	-	是	HDFS 中点数据文件的路径。用双引号括起路径，以 `hdfs://` 开头。
`tags.separator`	string	`,`	是	分隔符。默认值为英文逗号（,）。
`tags.header`	bool	`true`	是	文件是否有表头。

2022 vesoft Inc.

HIVE 源特有参数

参数	数据类型	默认值	是否必须	说明
`tags.exec`	string	-	是	查询数据源的语句。例如 `select name,age from mooc.users` 。

MAXCOMPUTE 源特有参数

参数	数据类型	默认值	是否必须	说明
`tags.table`	string	-	是	MaxCompute 的表名。
`tags.project`	string	-	是	MaxCompute 的项目名。
`tags.odpsUrl`	string	-	是	MaxCompute 服务的 odpsUrl。地址可根据阿里云文档查看。
`tags.tunnelUrl`	string	-	是	MaxCompute 服务的 tunnelUrl。地址可根据阿里云文档查看。
`tags.accessKeyId`	string	-	是	MaxCompute 服务的 accessKeyId。
`tags.accessKeySecret`	string	-	是	MaxCompute 服务的 accessKeySecret。
`tags.partitionSpec`	string	-	否	MaxCompute 表的分区描述。
`tags.numPartitions`	int	`1`	否	MaxCompute 的 Spark 连接器在读取 MaxCompute 数据时使用的分区数。
`tags.sentence`	string	-	否	查询数据源的语句。SQL 语句中的表名和上方 table 的值相同。

NEO4J 源特有参数

参数	数据类型	默认值	是否必须	说明
`tags.exec`	string	-	是	查询数据源的语句。例如 `match (n:label) return n.neo4j-field-0` 。
`tags.server`	string	`"bolt://127.0.0.1:7687"`	是	Neo4j 服务器地址。
`tags.user`	string	-	是	拥有读取权限的 Neo4j 用户名。
`tags.password`	string	-	是	用户名对应密码。
`tags.database`	string	-	是	Neo4j 中保存源数据的数据库名。
`tags.check_point_path`	string	`/tmp/test`	否	设置保存导入进度信息的目录，用于断点续传。如果未设置，表示不启用断点续传。

2022 vesoft Inc.

MYSQL/POSTGRESQL 源特有参数

参数	数据类型	默认值	是否必须	说明
`tags.host`	string	-	是	MySQL/PostgreSQL 服务器地址。
`tags.port`	string	-	是	MySQL/PostgreSQL 服务器端口。
`tags.database`	string	-	是	数据库名称。
`tags.table`	string	-	是	需要作为数据源的表名称。
`tags.user`	string	-	是	拥有读取权限的 MySQL/PostgreSQL 用户名。
`tags.password`	string	-	是	用户名对应密码。
`tags.sentence`	string	-	是	查询数据源的语句。例如 `"select teamid, name from team order by teamid"` 。

CLICKHOUSE 源特有参数

参数	数据类型	默认值	是否必须	说明
`tags.url`	string	-	是	ClickHouse 的 JDBC URL。
`tags.user`	string	-	是	有读取权限的 ClickHouse 用户名。
`tags.password`	string	-	是	用户名对应密码。
`tags.numPartition`	string	-	是	ClickHouse 分区数。
`tags.sentence`	string	-	是	查询数据源的语句。

HBASE 源特有参数

参数	数据类型	默认值	是否必须	说明
`tags.host`	string	`127.0.0.1`	是	Hbase 服务器地址。
`tags.port`	string	`2181`	是	Hbase 服务器端口。
`tags.table`	string	-	是	需要作为数据源的表名称。
`tags.columnFamily`	string	-	是	表所属的列族（column family）。

PULSAR 源特有参数

参数	数据类型	默认值	是否必须	说明
`tags.service`	string	`"pulsar://localhost:6650"`	是	Pulsar 服务器地址。
`tags.admin`	string	`"http://localhost:8081"`	是	连接 pulsar 的 admin.url。
`tags.options.<topic|topics|topicsPattern>`	string	-	是	Pulsar 的选项，可以从 `topic` 、 `topics` 和 `topicsPattern` 选择一个进行配置。
`tags.interval.seconds`	int	`10`	是	读取消息的间隔。单位：秒。

2022 vesoft Inc.

KAFKA 源特有参数

参数	数据类型	默认值	是否必须	说明
tags.service	string	-	是	Kafka 服务器地址。
tags.topic	string	-	是	消息类别。
tags.interval.seconds	int	10	是	读取消息的间隔。单位：秒。

生成 SST 时的特有参数

参数	数据类型	默认值	是否必须	说明
tags.path	string	-	是	指定需要生成 SST 文件的源文件的路径。
tags.repartitionWithNebula	bool	false	否	生成 SST 文件时是否要基于 Nebula Graph 中图空间的 partition 进行数据重分区。开启该功能可减少 DOWNLOAD 和 INGEST SST 文件需要的时间。

NEBULA GRAPH 源特有参数

nterpriseonly

Nebula Graph 源特有参数用于导出 Nebula Graph 数据，仅企业版 Exchange 支持。

参数	数据类型	默认值	是否必须	说明
tags.path	string	"hdfs://namenode:9000/path/vertex"	是	指定 CSV 文件的存储路径。设置的路径必须不存在，Exchange 会自动创建该路径。存储到 HDFS 服务器时路径格式同默认值，例如 "hdfs://192.168.8.177:9000/vertex/player" 。存储到本地时路径格式为 "file:///path/vertex" ，例如 "file:///home/nebula/vertex/player" 。有多个 Tag 时必须为每个 Tag 设置不同的目录。
tags.noField	bool	false	是	当值为 true 时，仅导出 VID 而不导出属性数据；当值为 false 时导出 VID 和属性数据。
tags.return.fields	list	[]	是	指定要导出的属性。例如，要导出 name 和 age 属性，需将参数值设置为 ["name","age"] 。该参数仅在 tags.noField 的值为 false 时生效。

边配置

对于不同的数据源，边的配置也有所不同，有很多通用参数，也有部分特有参数，配置时需要配置通用参数和不同数据源的特有参数。

边配置的不同数据源特有参数请参见上方点配置内的特有参数介绍，注意区分 tags 和 edges 即可。

2022 vesoft Inc.

通用参数

参数	数据类型	默认值	是否必须	说明
`edges.name`	string	-	是	Nebula Graph 中定义的 Edge type 名称。
`edges.type.source`	string	-	是	指定数据源。例如 `csv`。
`edges.type.sink`	string	`client`	是	指定导入方式，可选值为 `client` 和 `SST`。
`edges.fields`	list[string]	-	是	属性对应的列的表头或列名。如果有表头或列名，请直接使用该名称。如果 CSV 文件没有表头，用 `[_c0, _c1, _c2]` 的形式表示第一列、第二列、第三列，以此类推。
`edges.nebula.fields`	list[string]	-	是	Nebula Graph 中定义的属性名称，顺序必须和 `edges.fields` 一一对应。例如 `[_c2, _c3]` 对应 `[start_year, end_year]`，表示第三列为开始年份的值，第四列为结束年份的值。
`edges.source.field`	string	-	是	边的起始点的列。例如 `_c0` 表示第一列的值作为边的起始点。
`edges.target.field`	string	-	是	边的目的点的列。例如 `_c1` 表示第二列的值作为边的目的点。
`edges.ranking`	int	-	否	rank 值的列。没有指定时，默认所有 rank 值为 `0`。
`edges.batch`	int	`256`	是	单批次写入 Nebula Graph 的最大边数量。
`edges.partition`	int	`32`	是	Spark 分片数量。

生成 **SST** 时的特有参数

参数	数据类型	默认值	是否必须	说明
`edges.path`	string	-	是	指定需要生成 SST 文件的源文件的路径。
`edges.repartitionWithNebula`	bool	`false`	否	生成 SST 文件时是否要基于 Nebula Graph 中图空间的 partition 进行数据重分区。开启该功能可减少 DOWNLOAD 和 INGEST SST 文件需要的时间。

NEBULA GRAPH 源特有参数

参数	数据类型	默认值	是否必须	说明
`edges.path`	string	`"hdfs://namenode:9000/path/edge"`	是	指定 CSV 文件的存储路径。设置的路径必须不存在，Exchange 会自动创建该路径。存储到 HDFS 服务器时路径格式同默认值，例如 `"hdfs://192.168.8.177:9000/edge/follow"`。存储到本地时路径格式为 `"file:///path/edge"`，例如 `"file:///home/nebula/edge/follow"`。有多个 Edge 时必须为每个 Edge 设置不同的目录。
`edges.noField`	bool	`false`	是	当值为 `true` 时，仅导出起始点 VID、目的点 VID 和 Rank，而不导出属性数据；当值为 `false` 时导出起始点 VID、目的点 VID、Rank 和属性数据。
`edges.return.fields`	list	`[]`	是	指定要导出的属性。例如，要导出 `start_year` 和 `end_year` 属性，需将参数值设置为 `["start_year","end_year"]`。该参数仅在 `edges.noField` 的值为 `false` 时生效。

最后更新: June 20, 2023

2022 vesoft Inc.

19.4 使用 Nebula Exchange

19.4.1 导入 CSV 文件数据

本文以一个示例说明如何使用 Exchange 将存储在 HDFS 或本地的 CSV 文件数据导入 Nebula Graph。

如果要向 Nebula Graph 导入本地 CSV 文件，请参见 Nebula Importer。

数据集

本文以 basketballplayer 数据集为例。

环境配置

本文示例在 MacOS 下完成，以下是相关的环境配置信息：

- 硬件规格：
- CPU：1.7 GHz Quad-Core Intel Core i7
- 内存：16 GB
- Spark：2.4.7 单机版
- Hadoop：2.9.2 伪分布式部署
- Nebula Graph：3.1.0。使用 Docker Compose 部署。

前提条件

开始导入数据之前，用户需要确认以下信息：

- 已经安装部署 Nebula Graph 并获取如下信息：
- Graph 服务和 Meta 服务的的 IP 地址和端口。
- 拥有 Nebula Graph 写权限的用户名和密码。
- 已经编译 Exchange。详情请参见编译 Exchange。本示例中使用 Exchange 3.0.0。
- 已经安装 Spark。
- 了解 Nebula Graph 中创建 Schema 的信息，包括 Tag 和 Edge type 的名称、属性等。
- 如果文件存储在 HDFS 上，需要确认 Hadoop 服务运行正常。
- 如果文件存储在本地且 Nebula Graph 是集群架构，需要在集群每台机器本地相同目录下放置文件。

2022 vesoft Inc.

操作步骤

步骤 1：在 NEBULA GRAPH 中创建 SCHEMA

分析 CSV 文件中的数据，按以下步骤在 Nebula Graph 中创建 Schema：

1. 确认 Schema 要素。Nebula Graph 中的 Schema 要素如下表所示。

要素	名称	属性
Tag	`player`	`name string, age int`
Tag	`team`	`name string`
Edge Type	`follow`	`degree int`
Edge Type	`serve`	`start_year int, end_year int`

2. 使用 Nebula Console 创建一个图空间 **basketballplayer**，并创建一个 Schema，如下所示。

```
## 创建图空间
nebula> CREATE SPACE basketballplayer \
        (partition_num = 10, \
        replica_factor = 1, \
        vid_type = FIXED_STRING(30));

## 选择图空间 basketballplayer
nebula> USE basketballplayer;

## 创建 Tag player
nebula> CREATE TAG player(name string, age int);

## 创建 Tag team
nebula> CREATE TAG team(name string);

## 创建 Edge type follow
nebula> CREATE EDGE follow(degree int);

## 创建 Edge type serve
nebula> CREATE EDGE serve(start_year int, end_year int);
```

更多信息，请参见快速开始。

步骤 2：处理 CSV 文件

确认以下信息：

1. 处理 CSV 文件以满足 Schema 的要求。

> **Note**

Exchange 支持上传有表头或者无表头的 CSV 文件。

2. 获取 CSV 文件存储路径。

步骤 3：修改配置文件

编译 Exchange 后，复制 `target/classes/application.conf` 文件设置 CSV 数据源相关的配置。在本示例中，复制的文件名为 `csv_application.conf`。各个配置项的详细说明请参见配置说明。

```
{
  # Spark 相关配置
  spark: {
    app: {
      name: Nebula Exchange 3.0.0
    }
    driver: {
      cores: 1
      maxResultSize: 1G
    }
    executor: {
        memory:1G
    }

    cores: {
```


2022 vesoft Inc.

```
    max: 16
  }
}

# Nebula Graph 相关配置
nebula: {
  address:{
    # 指定 Graph 服务和所有 Meta 服务的 IP 地址和端口。
    # 如果有多台服务器，地址之间用英文逗号（,）分隔。
    # 格式："ip1:port","ip2:port","ip3:port"
    graph:["127.0.0.1:9669"]
    meta:["127.0.0.1:9559"]
  }

  # 指定拥有 Nebula Graph 写权限的用户名和密码。
  user: root
  pswd: nebula

  # 指定图空间名称。
  space: basketballplayer
  connection: {
    timeout: 3000
    retry: 3
  }
  execution: {
    retry: 3
  }
  error: {
    max: 32
    output: /tmp/errors
  }
  rate: {
    limit: 1024
    timeout: 1000
  }
}

# 处理点
tags: [
  # 设置 Tag player 相关信息。
  {
    # 指定 Nebula Graph 中定义的 Tag 名称。
    name: player
    type: {
      # 指定数据源，使用 CSV。
      source: csv

      # 指定如何将点数据导入 Nebula Graph：Client 或 SST。
      sink: client
    }

    # 指定 CSV 文件的路径。
    # 如果文件存储在 HDFS 上，用双引号括起路径，以 hdfs://开头，例如"hdfs://ip:port/xx/xx"。
    # 如果文件存储在本地，用双引号括起路径，以 file://开头，例如"file:///tmp/xx.csv"。
    path: "hdfs://192.168.*.*:9000/data/vertex_player.csv"

    # 如果 CSV 文件没有表头，使用 [_c0, _c1, _c2, ..., _cn] 表示其表头，并将列指示为属性值的源。
    # 如果 CSV 文件有表头，则使用实际的列名。
    fields: [_c1, _c2]

    # 指定 Nebula Graph 中定义的属性名称。
    # fields 与 nebula.fields 的顺序必须一一对应。
    nebula.fields: [age, name]

    # 指定一个列作为 VID 的源。
    # vertex 的值必须与上述 fields 或者 csv.fields 中的列名保持一致。
    # 目前，Nebula Graph 3.1.0仅支持字符串或整数类型的 VID。
    vertex: {
      field:_c0
      # policy:hash
    }

    # 指定的分隔符。默认值为英文逗号（,）。
    separator: ","

    # 如果 CSV 文件有表头，请将 header 设置为 true。
    # 如果 CSV 文件没有表头，请将 header 设置为 false。默认值为 false。
    header: false

    # 指定单批次写入 Nebula Graph 的最大点数量。
    batch: 256

    # 指定 Spark 分片数量。
    partition: 32
  }

  # 设置 Tag team 相关信息。
  {
    # 指定 Nebula Graph 中定义的 Tag 名称。
    name: team
    type: {
      # 指定数据源，使用 CSV。
      source: csv
```


2022 vesoft Inc.

```
      # 指定如何将点数据导入 Nebula Graph：Client 或 SST。
      sink: client
    }

    # 指定 CSV 文件的路径。
    # 如果文件存储在 HDFS 上，用双引号括起路径，以 hdfs://开头，例如"hdfs://ip:port/xx/xx"。
    # 如果文件存储在本地，用双引号括起路径，以 file://开头，例如"file:///tmp/xx.csv"。
    path: "hdfs://192.168.*.*:9000/data/vertex_team.csv"

    # 如果 CSV 文件没有表头，使用 [_c0, _c1, _c2, ..., _cn] 表示其表头，并将列指示为属性值的源。
    # 如果 CSV 文件有表头，则使用实际的列名。
    fields: [_c1]

    # 指定 Nebula Graph 中定义的属性名称。
    # fields 与 nebula.fields 的顺序必须一一对应。
    nebula.fields: [name]

    # 指定一个列作为 VID 的源。
    # vertex 的值必须与上述 fields 或者 csv.fields 中的列名保持一致。
    # 目前，Nebula Graph 3.1.0仅支持字符串或整数类型的 VID。
    vertex: {
      field:_c0
      # policy:hash
    }

    # 指定的分隔符。默认值为英文逗号（,）。
    separator: ","

    # 如果 CSV 文件有表头，请将 header 设置为 true。
    # 如果 CSV 文件没有表头，请将 header 设置为 false。默认值为 false。
    header: false

    # 指定单批次写入 Nebula Graph 的最大点数量。
    batch: 256

    # 指定 Spark 分片数量。
    partition: 32
  }

  # 如果需要添加更多点，请参考前面的配置进行添加。
]
# 处理边
edges: [
  # 设置 Edge type follow 相关信息。
  {
    # 指定 Nebula Graph 中定义的 Edge type 名称。
    name: follow
    type: {
      # 指定数据源，使用 CSV。
      source: csv

      # 指定如何将点数据导入 Nebula Graph：Client 或 SST。
      sink: client
    }

    # 指定 CSV 文件的路径。
    # 如果文件存储在 HDFS 上，用双引号括起路径，以 hdfs://开头，例如"hdfs://ip:port/xx/xx"。
    # 如果文件存储在本地，用双引号括起路径，以 file://开头，例如"file:///tmp/xx.csv"。
    path: "hdfs://192.168.*.*:9000/data/edge_follow.csv"

    # 如果 CSV 文件没有表头，使用 [_c0, _c1, _c2, ..., _cn] 表示其表头，并将列指示为属性值的源。
    # 如果 CSV 文件有表头，则使用实际的列名。
    fields: [_c2]

    # 指定 Nebula Graph 中定义的属性名称。
    # fields 与 nebula.fields 的顺序必须一一对应。
    nebula.fields: [degree]

    # 指定一个列作为起始点和目的点的源。
    # vertex 的值必须与上述 fields 或者 csv.fields 中的列名保持一致。
    # 目前，Nebula Graph 3.1.0仅支持字符串或整数类型的 VID。
    source: {
      field: _c0
    }
    target: {
      field: _c1
    }

    # 指定的分隔符。默认值为英文逗号（,）。
    separator: ","

    # 指定一个列作为 rank 的源（可选）。

    #ranking: rank

    # 如果 CSV 文件有表头，请将 header 设置为 true。
    # 如果 CSV 文件没有表头，请将 header 设置为 false。默认值为 false。
    header: false

    # 指定单批次写入 Nebula Graph 的最大边数量。
    batch: 256

    # 指定 Spark 分片数量。
    partition: 32
```


2022 vesoft Inc.

```
    }

    # 设置 Edge type serve 相关信息。
    {
      # 指定 Nebula Graph 中定义的 Edge type 名称。
      name: serve
      type: {
        # 指定数据源，使用 CSV。
        source: csv

        # 指定如何将点数据导入 Nebula Graph：Client 或 SST。
        sink: client
      }

      # 指定 CSV 文件的路径。
      # 如果文件存储在 HDFS 上，用双引号括起路径，以 hdfs://开头，例如"hdfs://ip:port/xx/xx"。
      # 如果文件存储在本地，用双引号括起路径，以 file://开头，例如"file:///tmp/xx.csv"。
      path: "hdfs://192.168.*.*:9000/data/edge_serve.csv"

      # 如果 CSV 文件没有表头，使用 [_c0, _c1, _c2, ..., _cn] 表示其表头，并将列指示为属性值的源。
      # 如果 CSV 文件有表头，则使用实际的列名。
      fields: [_c2,_c3]

      # 指定 Nebula Graph 中定义的属性名称。
      # fields 与 nebula.fields 的顺序必须一一对应。
      nebula.fields: [start_year, end_year]

      # 指定一个列作为起始点和目的点的源。
      # vertex 的值必须与上述 fields 或者 csv.fields 中的列名保持一致。
      # 目前，Nebula Graph 3.1.0仅支持字符串或整数类型的 VID。
      source: {
        field: _c0
      }
      target: {
        field: _c1
      }

      # 指定的分隔符。默认值为英文逗号（,）。
      separator: ","

      # 指定一个列作为 rank 的源（可选）。
      #ranking: _c5

      # 如果 CSV 文件有表头，请将 header 设置为 true。
      # 如果 CSV 文件没有表头，请将 header 设置为 false。默认值为 false。
      header: false

      # 指定单批次写入 Nebula Graph 的最大边数量。
      batch: 256

      # 指定 Spark 分片数量。
      partition: 32
    }

  ]
  # 如果需要添加更多边，请参考前面的配置进行添加。
}
```

步骤 **4**：向 **NEBULA GRAPH** 导入数据

运行如下命令将 CSV 文件数据导入到 Nebula Graph 中。关于参数的说明，请参见导入命令参数。

```
${SPARK_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange <nebula-exchange-3.0.0.jar_path> -c <csv_application.conf_path>
```

> **Note**
>
> JAR 包有两种获取方式：自行编译或者从 maven 仓库下载。

示例：

```
${SPARK_HOME}/bin/spark-submit  --master "local" --class com.vesoft.nebula.exchange.Exchange  /root/nebula-exchange/nebula-exchange/target/nebula-exchange-3.0.0.jar  -c /root/nebula-exchange/nebula-exchange/target/classes/csv_application.conf
```

用户可以在返回信息中搜索 `batchSuccess.<tag_name/edge_name>`，确认成功的数量。例如 `batchSuccess.follow: 300` 。

步骤 **5**：（可选）验证数据

用户可以在 Nebula Graph 客户端（例如 Nebula Studio）中执行查询语句，确认数据是否已导入。例如：

```
GO FROM "player100" OVER follow;
```


2022 vesoft Inc.

用户也可以使用命令 `SHOW STATS` 查看统计数据。

步骤 **6**: （如有）在 **NEBULA GRAPH** 中重建索引

导入数据后，用户可以在 Nebula Graph 中重新创建并重建索引。详情请参见索引介绍。

最后更新: June 20, 2023

2022 vesoft Inc.

19.4.2 导入 JSON 文件数据

本文以一个示例说明如何使用 Exchange 将存储在 HDFS 或本地的 JSON 文件数据导入 Nebula Graph。

数据集

本文以 basketballplayer 数据集为例。部分示例数据如下：

- player

```
{"id":"player100","age":42,"name":"Tim Duncan"}
{"id":"player101","age":36,"name":"Tony Parker"}
{"id":"player102","age":33,"name":"LaMarcus Aldridge"}
{"id":"player103","age":32,"name":"Rudy Gay"}
...
```

- team

```
{"id":"team200","name":"Warriors"}
{"id":"team201","name":"Nuggets"}
...
```

- follow

```
{"src":"player100","dst":"player101","degree":95}
{"src":"player101","dst":"player102","degree":90}
...
```

- serve

```
{"src":"player100","dst":"team204","start_year":"1997","end_year":"2016"}
{"src":"player101","dst":"team204","start_year":"1999","end_year":"2018"}
...
```

环境配置

本文示例在 MacOS 下完成，以下是相关的环境配置信息：

- 硬件规格：
- CPU：1.7 GHz Quad-Core Intel Core i7
- 内存：16 GB
- Spark：2.3.0，单机版
- Hadoop：2.9.2，伪分布式部署
- Nebula Graph：3.1.0。使用 Docker Compose 部署。

2022 vesoft Inc.

前提条件

开始导入数据之前，用户需要确认以下信息：

- 已经安装部署 Nebula Graph 并获取如下信息：
- Graph 服务和 Meta 服务的的 IP 地址和端口。
- 拥有 Nebula Graph 写权限的用户名和密码。
- 已经编译 Exchange。详情请参见编译 Exchange。本示例中使用 Exchange 3.0.0。
- 已经安装 Spark。
- 了解 Nebula Graph 中创建 Schema 的信息，包括 Tag 和 Edge type 的名称、属性等。
- 如果文件存储在 HDFS 上，需要确认 Hadoop 服务运行正常。
- 如果文件存储在本地且 Nebula Graph 是集群架构，需要在集群每台机器本地相同目录下放置文件。

操作步骤

步骤 1：在 NEBULA GRAPH 中创建 SCHEMA

分析文件中的数据，按以下步骤在 Nebula Graph 中创建 Schema：

1. 确认 Schema 要素。Nebula Graph 中的 Schema 要素如下表所示。

要素	名称	属性
Tag	`player`	`name string, age int`
Tag	`team`	`name string`
Edge Type	`follow`	`degree int`
Edge Type	`serve`	`start_year int, end_year int`

2. 使用 Nebula Console 创建一个图空间 **basketballplayer**，并创建一个 Schema，如下所示。

```
## 创建图空间
nebula> CREATE SPACE basketballplayer \
        (partition_num = 10, \
        replica_factor = 1, \
        vid_type = FIXED_STRING(30));

## 选择图空间 basketballplayer
nebula> USE basketballplayer;

## 创建 Tag player
nebula> CREATE TAG player(name string, age int);

## 创建 Tag team
nebula> CREATE TAG team(name string);

## 创建 Edge type follow
nebula> CREATE EDGE follow(degree int);

## 创建 Edge type serve
nebula> CREATE EDGE serve(start_year int, end_year int);
```

更多信息，请参见快速开始。

步骤 2：处理 JSON 文件

确认以下信息：

1. 处理 JSON 文件以满足 Schema 的要求。
2. 获取 JSON 文件存储路径。

2022 vesoft Inc.

步骤 **3.** 修改配置文件

编译 Exchange 后，复制 `target/classes/application.conf` 文件设置 JSON 数据源相关的配置。在本示例中，复制的文件名为 `json_application.conf` 。各个配置项的详细说明请参见配置说明。

```
{
  # Spark 相关配置
  spark: {
    app: {
      name: Nebula Exchange 3.0.0
    }
    driver: {
      cores: 1
      maxResultSize: 1G
    }
    executor: {
        memory:1G
    }

    cores: {
      max: 16
    }
  }

  # Nebula Graph 相关配置
  nebula: {
    address:{
      # 指定 Graph 服务和所有 Meta 服务的 IP 地址和端口。
      # 如果有多台服务器，地址之间用英文逗号（,）分隔。
      # 格式："ip1:port","ip2:port","ip3:port"
      graph:["127.0.0.1:9669"]
      meta:["127.0.0.1:9559"]
    }

    # 指定拥有 Nebula Graph 写权限的用户名和密码。
    user: root
    pswd: nebula

    # 指定图空间名称。
    space: basketballplayer
    connection: {
      timeout: 3000
      retry: 3
    }
    execution: {
      retry: 3
    }
    error: {
      max: 32
      output: /tmp/errors
    }
    rate: {
      limit: 1024
      timeout: 1000
    }
  }

  # 处理点
  tags: [
    # 设置 Tag player 相关信息。
    {
      # 指定 Nebula Graph 中定义的 Tag 名称。
      name: player
      type: {
        # 指定数据源，使用 JSON。
        source: json

        # 指定如何将点数据导入 Nebula Graph：Client 或 SST。
        sink: client
      }

      # 指定 JSON 文件的路径。
      # 如果文件存储在 HDFS 上，用双引号括起路径，以 hdfs://开头，例如"hdfs://ip:port/xx/xx"。
      # 如果文件存储在本地，用双引号括起路径，以 file://开头，例如"file:///tmp/xx.json"。
      path: "hdfs://192.168.*.*:9000/data/vertex_player.json"

      # 在 fields 里指定 JSON 文件中 key 名称，其对应的 value 会作为 Nebula Graph 中指定属性的数据源。
      # 如果需要指定多个值，用英文逗号（,）隔开。
      fields: [age,name]

      # 指定 Nebula Graph 中定义的属性名称。
      # fields 与 nebula.fields 的顺序必须一一对应。
      nebula.fields: [age, name]

      # 指定一个列作为 VID 的源。
      # vertex 的值必须与 JSON 文件中的字段保持一致。
      # 目前，Nebula Graph 3.1.0仅支持字符串或整数类型的 VID。
      vertex: {
        field:id
      }

      # 指定单批次写入 Nebula Graph 的最大点数量。
```


2022 vesoft Inc.

```
    batch: 256

    # 指定 Spark 分片数量。
    partition: 32
  }

  # 设置 Tag team 相关信息。
  {
    # 指定 Nebula Graph 中定义的 Tag 名称。
    name: team
    type: {
      # 指定数据源，使用 JSON。
      source: json

      # 指定如何将点数据导入 Nebula Graph：Client 或 SST。
      sink: client
    }

    # 指定 JSON 文件的路径。
    # 如果文件存储在 HDFS 上，用双引号括起路径，以 hdfs://开头，例如"hdfs://ip:port/xx/xx"。
    # 如果文件存储在本地，用双引号括起路径，以 file://开头，例如"file:///tmp/xx.json"。
    path: "hdfs://192.168.*.*:9000/data/vertex_team.json"

    # 在 fields 里指定 JSON 文件中 key 名称，其对应的 value 会作为 Nebula Graph 中指定属性的数据源。
    # 如果需要指定多个值，用英文逗号（,）隔开。
    fields: [name]

    # 指定 Nebula Graph 中定义的属性名称。
    # fields 与 nebula.fields 的顺序必须一一对应。
    nebula.fields: [name]

    # 指定一个列作为 VID 的源。
    # vertex 的值必须与 JSON 文件中的字段保持一致。
    # 目前，Nebula Graph 3.1.0仅支持字符串或整数类型的 VID。
    vertex: {
      field:id
    }

    # 指定单批次写入 Nebula Graph 的最大点数量。
    batch: 256

    # 指定 Spark 分片数量。
    partition: 32
  }

  # 如果需要添加更多点，请参考前面的配置进行添加。
]
# 处理边
edges: [
  # 设置 Edge type follow 相关信息。
  {
    # 指定 Nebula Graph 中定义的 Edge type 名称。
    name: follow
    type: {
      # 指定数据源，使用 JSON。
      source: json

      # 指定如何将点数据导入 Nebula Graph：Client 或 SST。
      sink: client
    }

    # 指定 JSON 文件的路径。
    # 如果文件存储在 HDFS 上，用双引号括起路径，以 hdfs://开头，例如"hdfs://ip:port/xx/xx"。
    # 如果文件存储在本地，用双引号括起路径，以 file://开头，例如"file:///tmp/xx.json"。
    path: "hdfs://192.168.*.*:9000/data/edge_follow.json"

    # 在 fields 里指定 JSON 文件中 key 名称，其对应的 value 会作为 Nebula Graph 中指定属性的数据源。
    # 如果需要指定多个值，用英文逗号（,）隔开。
    fields: [degree]

    # 指定 Nebula Graph 中定义的属性名称。
    # fields 与 nebula.fields 的顺序必须一一对应。
    nebula.fields: [degree]

    # 指定一个列作为起始点和目的点的源。
    # vertex 的值必须与 JSON 文件中的字段保持一致。
    # 目前，Nebula Graph 3.1.0仅支持字符串或整数类型的 VID。
    source: {
      field: src
    }
    target: {
      field: dst
    }

    # 指定一个列作为 rank 的源（可选）。
    #ranking: rank

    # 指定单批次写入 Nebula Graph 的最大边数量。
    batch: 256

    # 指定 Spark 分片数量。
    partition: 32
  }
```


2022 vesoft Inc.

```
    # 设置 Edge type serve 相关信息。
    {
      # 指定 Nebula Graph 中定义的 Edge type 名称。
      name: serve
      type: {
        # 指定数据源，使用 JSON。
        source: json

        # 指定如何将点数据导入 Nebula Graph：Client 或 SST。
        sink: client
      }

      # 指定 JSON 文件的路径。
      # 如果文件存储在 HDFS 上，用双引号括起路径，以 hdfs://开头，例如"hdfs://ip:port/xx/xx"。
      # 如果文件存储在本地，用双引号括起路径，以 file://开头，例如"file:///tmp/xx.json"。
      path: "hdfs://192.168.*.*:9000/data/edge_serve.json"

      # 在 fields 里指定 JSON 文件中 key 名称，其对应的 value 会作为 Nebula Graph 中指定属性的数据源。
      # 如果需要指定多个值，用英文逗号（,）隔开。
      fields: [start_year,end_year]

      # 指定 Nebula Graph 中定义的属性名称。
      # fields 与 nebula.fields 的顺序必须一一对应。
      nebula.fields: [start_year, end_year]

      # 指定一个列作为起始点和目的点的源。
      # vertex 的值必须与 JSON 文件中的字段保持一致。
      # 目前，Nebula Graph 3.1.0仅支持字符串或整数类型的 VID。
      source: {
        field: src
      }
      target: {
        field: dst
      }

      # 指定一个列作为 rank 的源（可选）。
      #ranking: _c5

      # 指定单批次写入 Nebula Graph 的最大边数量。
      batch: 256

      # 指定 Spark 分片数量。
      partition: 32
    }

  ]
  # 如果需要添加更多边，请参考前面的配置进行添加。
}
```

步骤 **4**：向 **NEBULA GRAPH** 导入数据

运行如下命令将 JSON 文件数据导入到 Nebula Graph 中。关于参数的说明，请参见导入命令参数。

```
${SPARK_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange <nebula-exchange-3.0.0.jar_path> -c <json_application.conf_path>
```

> **Note**
>
> JAR 包有两种获取方式：自行编译或者从 maven 仓库下载。

示例：

```
${SPARK_HOME}/bin/spark-submit  --master "local" --class com.vesoft.nebula.exchange.Exchange  /root/nebula-echange/nebula-exchange/target/nebula-exchange-3.0.0.jar  -c /root/nebula-exchange/nebula-exchange/target/classes/json_application.conf
```

用户可以在返回信息中搜索 `batchSuccess.<tag_name/edge_name>`，确认成功的数量。例如 `batchSuccess.follow: 300` 。

步骤 **5**：（可选）验证数据

用户可以在 Nebula Graph 客户端（例如 Nebula Studio）中执行查询语句，确认数据是否已导入。例如：

```
GO FROM "player100" OVER follow;
```

用户也可以使用命令 `SHOW STATS` 查看统计数据。

步骤 **6**：（如有）在 **NEBULA GRAPH** 中重建索引

导入数据后，用户可以在 Nebula Graph 中重新创建并重建索引。详情请参见索引介绍。

2022 vesoft Inc.

最后更新: June 20, 2023

2022 vesoft Inc.

19.4.3 导入 ORC 文件数据

本文以一个示例说明如何使用 Exchange 将存储在 HDFS 或本地的 ORC 文件数据导入 Nebula Graph。

如果要向 Nebula Graph 导入本地 ORC 文件，请参见 Nebula Importer。

数据集

本文以 basketballplayer 数据集为例。

环境配置

本文示例在 MacOS 下完成，以下是相关的环境配置信息：

- 硬件规格：
- CPU：1.7 GHz Quad-Core Intel Core i7
- 内存：16 GB
- Spark：2.4.7 单机版
- Hadoop：2.9.2 伪分布式部署
- Nebula Graph：3.1.0。使用 Docker Compose 部署。

前提条件

开始导入数据之前，用户需要确认以下信息：

- 已经安装部署 Nebula Graph 并获取如下信息：
- Graph 服务和 Meta 服务的的 IP 地址和端口。
- 拥有 Nebula Graph 写权限的用户名和密码。
- 已经编译 Exchange。详情请参见编译 Exchange。本示例中使用 Exchange 3.0.0。
- 已经安装 Spark。
- 了解 Nebula Graph 中创建 Schema 的信息，包括 Tag 和 Edge type 的名称、属性等。
- 如果文件存储在 HDFS 上，需要确认 Hadoop 服务运行正常。
- 如果文件存储在本地且 Nebula Graph 是集群架构，需要在集群每台机器本地相同目录下放置文件。

2022 vesoft Inc.

操作步骤

步骤 1：在 NEBULA GRAPH 中创建 SCHEMA

分析 ORC 文件中的数据，按以下步骤在 Nebula Graph 中创建 Schema：

1. 确认 Schema 要素。Nebula Graph 中的 Schema 要素如下表所示。

要素	名称	属性
Tag	`player`	`name string, age int`
Tag	`team`	`name string`
Edge Type	`follow`	`degree int`
Edge Type	`serve`	`start_year int, end_year int`

2. 使用 Nebula Console 创建一个图空间 **basketballplayer**，并创建一个 Schema，如下所示。

```
## 创建图空间
nebula> CREATE SPACE basketballplayer \
        (partition_num = 10, \
        replica_factor = 1, \
        vid_type = FIXED_STRING(30));

## 选择图空间 basketballplayer
nebula> USE basketballplayer;

## 创建 Tag player
nebula> CREATE TAG player(name string, age int);

## 创建 Tag team
nebula> CREATE TAG team(name string);

## 创建 Edge type follow
nebula> CREATE EDGE follow(degree int);

## 创建 Edge type serve
nebula> CREATE EDGE serve(start_year int, end_year int);
```

更多信息，请参见快速开始。

步骤 2：处理 ORC 文件

确认以下信息：

1. 处理 ORC 文件以满足 Schema 的要求。
2. 获取 ORC 文件存储路径。

步骤 3：修改配置文件

编译 Exchange 后，复制 `target/classes/application.conf` 文件设置 ORC 数据源相关的配置。在本示例中，复制的文件名为 `orc_application.conf`。各个配置项的详细说明请参见配置说明。

```
{
  # Spark 相关配置
  spark: {
    app: {
      name: Nebula Exchange 3.0.0
    }
    driver: {
      cores: 1
      maxResultSize: 1G
    }
    executor: {
        memory:1G
    }

    cores: {
      max: 16
    }
  }

  # Nebula Graph 相关配置
  nebula: {
    address:{
      # 指定 Graph 服务和所有 Meta 服务的 IP 地址和端口。
      # 如果有多台服务器，地址之间用英文逗号（,）分隔。
```


2022 vesoft Inc.

```
    # 格式："ip1:port","ip2:port","ip3:port"
    graph:["127.0.0.1:9669"]
    meta:["127.0.0.1:9559"]
  }

  # 指定拥有 Nebula Graph 写权限的用户名和密码。
  user: root
  pswd: nebula

  # 指定图空间名称。
  space: basketballplayer
  connection: {
    timeout: 3000
    retry: 3
  }
  execution: {
    retry: 3
  }
  error: {
    max: 32
    output: /tmp/errors
  }
  rate: {
    limit: 1024
    timeout: 1000
  }
}

# 处理点
tags: [
  # 设置 Tag player 相关信息。
  {
    # 指定 Nebula Graph 中定义的 Tag 名称。
    name: player
    type: {
      # 指定数据源，使用 ORC。
      source: orc

      # 指定如何将点数据导入 Nebula Graph：Client 或 SST。
      sink: client
    }

    # 指定 ORC 文件的路径。
    # 如果文件存储在 HDFS 上，用双引号括起路径，以 hdfs://开头，例如"hdfs://ip:port/xx/xx"。
    # 如果文件存储在本地，用双引号括起路径，以 file://开头，例如"file:///tmp/xx.orc"。
    path: "hdfs://192.168.*.*:9000/data/vertex_player.orc"

    # 在 fields 里指定 ORC 文件中 key 名称，其对应的 value 会作为 Nebula Graph 中指定属性的数据源。
    # 如果需要指定多个值，用英文逗号（,）隔开。
    fields: [age,name]

    # 指定 Nebula Graph 中定义的属性名称。
    # fields 与 nebula.fields 的顺序必须一一对应。
    nebula.fields: [age, name]

    # 指定一个列作为 VID 的源。
    # vertex 的值必须与 ORC 文件中的字段保持一致。
    # 目前，Nebula Graph 3.1.0仅支持字符串或整数类型的 VID。
    vertex: {
      field:id
    }

    # 指定单批次写入 Nebula Graph 的最大点数量。
    batch: 256

    # 指定 Spark 分片数量。
    partition: 32
  }

  # 设置 Tag team 相关信息。
  {
    # 指定 Nebula Graph 中定义的 Tag 名称。
    name: team
    type: {
      # 指定数据源，使用 ORC。
      source: orc

      # 指定如何将点数据导入 Nebula Graph：Client 或 SST。
      sink: client
    }

    # 指定 ORC 文件的路径。
    # 如果文件存储在 HDFS 上，用双引号括起路径，以 hdfs://开头，例如"hdfs://ip:port/xx/xx"。
    # 如果文件存储在本地，用双引号括起路径，以 file://开头，例如"file:///tmp/xx.orc"。
    path: "hdfs://192.168.*.*:9000/data/vertex_team.orc"

    # 在 fields 里指定 ORC 文件中 key 名称，其对应的 value 会作为 Nebula Graph 中指定属性的数据源。
    # 如果需要指定多个值，用英文逗号（,）隔开。
    fields: [name]

    # 指定 Nebula Graph 中定义的属性名称。
    # fields 与 nebula.fields 的顺序必须一一对应。
    nebula.fields: [name]
```


2022 vesoft Inc.

```
    # 指定一个列作为 VID 的源。
    # vertex 的值必须与 ORC 文件中的字段保持一致。
    # 目前，Nebula Graph 3.1.0仅支持字符串或整数类型的 VID。
    vertex: {
      field:id
    }

    # 指定单批次写入 Nebula Graph 的最大点数量。
    batch: 256

    # 指定 Spark 分片数量。
    partition: 32
  }

  # 如果需要添加更多点，请参考前面的配置进行添加。
]
# 处理边
edges: [
  # 设置 Edge type follow 相关信息。
  {
    # 指定 Nebula Graph 中定义的 Edge type 名称。
    name: follow
    type: {
      # 指定数据源，使用 ORC。
      source: orc

      # 指定如何将点数据导入 Nebula Graph：Client 或 SST。
      sink: client
    }

    # 指定 ORC 文件的路径。
    # 如果文件存储在 HDFS 上，用双引号括起路径，以 hdfs://开头，例如"hdfs://ip:port/xx/xx"。
    # 如果文件存储在本地，用双引号括起路径，以 file://开头，例如"file:///tmp/xx.orc"。
    path: "hdfs://192.168.*.*:9000/data/edge_follow.orc"

    # 在 fields 里指定 ORC 文件中 key 名称，其对应的 value 会作为 Nebula Graph 中指定属性的数据源。
    # 如果需要指定多个值，用英文逗号（,）隔开。
    fields: [degree]

    # 指定 Nebula Graph 中定义的属性名称。
    # fields 与 nebula.fields 的顺序必须一一对应。
    nebula.fields: [degree]

    # 指定一个列作为起始点和目的点的源。
    # vertex 的值必须与 ORC 文件中的字段保持一致。
    # 目前，Nebula Graph 3.1.0仅支持字符串或整数类型的 VID。
    source: {
      field: src
    }
    target: {
      field: dst
    }

    # 指定一个列作为 rank 的源（可选）。
    #ranking: rank

    # 指定单批次写入 Nebula Graph 的最大边数量。
    batch: 256

    # 指定 Spark 分片数量。
    partition: 32
  }

  # 设置 Edge type serve 相关信息。
  {
    # 指定 Nebula Graph 中定义的 Edge type 名称。
    name: serve
    type: {
      # 指定数据源，使用 ORC。
      source: orc

      # 指定如何将点数据导入 Nebula Graph：Client 或 SST。
      sink: client
    }

    # 指定 ORC 文件的路径。
    # 如果文件存储在 HDFS 上，用双引号括起路径，以 hdfs://开头，例如"hdfs://ip:port/xx/xx"。
    # 如果文件存储在本地，用双引号括起路径，以 file://开头，例如"file:///tmp/xx.orc"。
    path: "hdfs://192.168.*.*:9000/data/edge_serve.orc"

    # 在 fields 里指定 ORC 文件中 key 名称，其对应的 value 会作为 Nebula Graph 中指定属性的数据源。
    # 如果需要指定多个值，用英文逗号（,）隔开。
    fields: [start_year,end_year]

    # 指定 Nebula Graph 中定义的属性名称。
    # fields 与 nebula.fields 的顺序必须一一对应。
    nebula.fields: [start_year, end_year]

    # 指定一个列作为起始点和目的点的源。
    # vertex 的值必须与 ORC 文件中的字段保持一致。
    # 目前，Nebula Graph 3.1.0仅支持字符串或整数类型的 VID。
    source: {
      field: src
    }
```


2022 vesoft Inc.

```
      target: {
        field: dst
      }

      # 指定一个列作为 rank 的源（可选）。
      #ranking: _c5

      # 指定单批次写入 Nebula Graph 的最大边数量。
      batch: 256

      # 指定 Spark 分片数量。
      partition: 32
    }

  ]
  # 如果需要添加更多边，请参考前面的配置进行添加。
}
```

步骤 **4**：向 **NEBULA GRAPH** 导入数据

运行如下命令将 ORC 文件数据导入到 Nebula Graph 中。关于参数的说明，请参见导入命令参数。

```
${SPARK_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange <nebula-exchange-3.0.0.jar_path> -c <orc_application.conf_path>
```

> **Note**
>
> JAR 包有两种获取方式：自行编译或者从 maven 仓库下载。

示例：

```
${SPARK_HOME}/bin/spark-submit  --master "local" --class com.vesoft.nebula.exchange.Exchange  /root/nebula-exchange/nebula-exchange/target/nebula-exchange-3.0.0.jar  -c /root/nebula-exchange/nebula-exchange/target/classes/orc_application.conf
```

用户可以在返回信息中搜索 `batchSuccess.<tag_name/edge_name>`，确认成功的数量。例如 `batchSuccess.follow: 300` 。

步骤 **5**：（可选）验证数据

用户可以在 Nebula Graph 客户端（例如 Nebula Studio）中执行查询语句，确认数据是否已导入。例如：

```
GO FROM "player100" OVER follow;
```

用户也可以使用命令 `SHOW STATS` 查看统计数据。

步骤 **6**：（如有）在 **NEBULA GRAPH** 中重建索引

导入数据后，用户可以在 Nebula Graph 中重新创建并重建索引。详情请参见索引介绍。

最后更新: June 20, 2023

2022 vesoft Inc.

19.4.4 导入 Parquet 文件数据

本文以一个示例说明如何使用 Exchange 将存储在 HDFS 或本地的 Parquet 文件数据导入 Nebula Graph。

如果要向 Nebula Graph 导入本地 Parquet 文件，请参见 Nebula Importer。

数据集

本文以 basketballplayer 数据集为例。

环境配置

本文示例在 MacOS 下完成，以下是相关的环境配置信息：

- 硬件规格：
- CPU：1.7 GHz Quad-Core Intel Core i7
- 内存：16 GB
- Spark：2.4.7 单机版
- Hadoop：2.9.2 伪分布式部署
- Nebula Graph：3.1.0。使用 Docker Compose 部署。

前提条件

开始导入数据之前，用户需要确认以下信息：

- 已经安装部署 Nebula Graph 并获取如下信息：
- Graph 服务和 Meta 服务的的 IP 地址和端口。
- 拥有 Nebula Graph 写权限的用户名和密码。
- 已经编译 Exchange。详情请参见编译 Exchange。本示例中使用 Exchange 3.0.0。
- 已经安装 Spark。
- 了解 Nebula Graph 中创建 Schema 的信息，包括 Tag 和 Edge type 的名称、属性等。
- 如果文件存储在 HDFS 上，需要确认 Hadoop 服务运行正常。
- 如果文件存储在本地且 Nebula Graph 是集群架构，需要在集群每台机器本地相同目录下放置文件。

2022 vesoft Inc.

操作步骤

步骤 1：在 NEBULA GRAPH 中创建 SCHEMA

分析 Parquet 文件中的数据，按以下步骤在 Nebula Graph 中创建 Schema：

1. 确认 Schema 要素。Nebula Graph 中的 Schema 要素如下表所示。

要素	名称	属性
Tag	`player`	`name string, age int`
Tag	`team`	`name string`
Edge Type	`follow`	`degree int`
Edge Type	`serve`	`start_year int, end_year int`

2. 使用 Nebula Console 创建一个图空间 **basketballplayer**，并创建一个 Schema，如下所示。

```
## 创建图空间
nebula> CREATE SPACE basketballplayer \
        (partition_num = 10, \
        replica_factor = 1, \
        vid_type = FIXED_STRING(30));

## 选择图空间 basketballplayer
nebula> USE basketballplayer;

## 创建 Tag player
nebula> CREATE TAG player(name string, age int);

## 创建 Tag team
nebula> CREATE TAG team(name string);

## 创建 Edge type follow
nebula> CREATE EDGE follow(degree int);

## 创建 Edge type serve
nebula> CREATE EDGE serve(start_year int, end_year int);
```

更多信息，请参见快速开始。

步骤 2：处理 PARQUET 文件

确认以下信息：

1. 处理 Parquet 文件以满足 Schema 的要求。
2. 获取 Parquet 文件存储路径。

步骤 3：修改配置文件

编译 Exchange 后，复制 `target/classes/application.conf` 文件设置 Parquet 数据源相关的配置。在本示例中，复制的文件名为 `parquet_application.conf`。各个配置项的详细说明请参见配置说明。

```
{
  # Spark 相关配置
  spark: {
    app: {
      name: Nebula Exchange 3.0.0
    }
    driver: {
      cores: 1
      maxResultSize: 1G
    }
    executor: {
        memory:1G
    }

    cores: {
      max: 16
    }
  }

  # Nebula Graph 相关配置
  nebula: {
    address:{
      # 指定 Graph 服务和所有 Meta 服务的 IP 地址和端口。
      # 如果有多台服务器，地址之间用英文逗号（,）分隔。
```


2022 vesoft Inc.

```
    # 格式："ip1:port","ip2:port","ip3:port"
    graph:["127.0.0.1:9669"]
    meta:["127.0.0.1:9559"]
  }

  # 指定拥有 Nebula Graph 写权限的用户名和密码。
  user: root
  pswd: nebula

  # 指定图空间名称。
  space: basketballplayer
  connection: {
    timeout: 3000
    retry: 3
  }
  execution: {
    retry: 3
  }
  error: {
    max: 32
    output: /tmp/errors
  }
  rate: {
    limit: 1024
    timeout: 1000
  }
}

# 处理点
tags: [
  # 设置 Tag player 相关信息。
  {
    # 指定 Nebula Graph 中定义的 Tag 名称。
    name: player
    type: {
      # 指定数据源，使用 Parquet。
      source: parquet

      # 指定如何将点数据导入 Nebula Graph：Client 或 SST。
      sink: client
    }

    # 指定 Parquet 文件的路径。
    # 如果文件存储在 HDFS 上，用双引号括起路径，以 hdfs://开头，例如"hdfs://ip:port/xx/xx"。
    # 如果文件存储在本地，用双引号括起路径，以 file://开头，例如"file:///tmp/xx.csv"。
    path: "hdfs://192.168.11.13900O/data/vertex_player.parquet"

    # 在 fields 里指定 Parquet 文件中 key 名称，其对应的 value 会作为 Nebula Graph 中指定属性的数据源。
    # 如果需要指定多个值，用英文逗号（,）隔开。
    fields: [age,name]

    # 指定 Nebula Graph 中定义的属性名称。
    # fields 与 nebula.fields 的顺序必须一一对应。
    nebula.fields: [age, name]

    # 指定一个列作为 VID 的源。
    # vertex 的值必须与 Parquet 文件中的字段保持一致。
    # 目前，Nebula Graph 3.1.0仅支持字符串或整数类型的 VID。
    vertex: {
      field:id
    }

    # 指定单批次写入 Nebula Graph 的最大点数量。
    batch: 256

    # 指定 Spark 分片数量。
    partition: 32
  }

  # 设置 Tag team 相关信息。
  {
    # 指定 Nebula Graph 中定义的 Tag 名称。
    name: team
    type: {
      # 指定数据源，使用 Parquet。
      source: parquet

      # 指定如何将点数据导入 Nebula Graph：Client 或 SST。
      sink: client
    }

    # 指定 Parquet 文件的路径。
    # 如果文件存储在 HDFS 上，用双引号括起路径，以 hdfs://开头，例如"hdfs://ip:port/xx/xx"。
    # 如果文件存储在本地，用双引号括起路径，以 file://开头，例如"file:///tmp/xx.csv"。
    path: "hdfs://192.168.11.13:9000/data/vertex_team.parquet"

    # 在 fields 里指定 Parquet 文件中 key 名称，其对应的 value 会作为 Nebula Graph 中指定属性的数据源。
    # 如果需要指定多个值，用英文逗号（,）隔开。
    fields: [name]

    # 指定 Nebula Graph 中定义的属性名称。
    # fields 与 nebula.fields 的顺序必须一一对应。
    nebula.fields: [name]
```


2022 vesoft Inc.

```
    # 指定一个列作为 VID 的源。
    # vertex 的值必须与 Parquet 文件中的字段保持一致。
    # 目前，Nebula Graph 3.1.0仅支持字符串或整数类型的 VID。
    vertex: {
      field:id
    }

    # 指定单批次写入 Nebula Graph 的最大点数量。
    batch: 256

    # 指定 Spark 分片数量。
    partition: 32
  }

  # 如果需要添加更多点，请参考前面的配置进行添加。
]
# 处理边
edges: [
  # 设置 Edge type follow 相关信息。
  {
    # 指定 Nebula Graph 中定义的 Edge type 名称。
    name: follow
    type: {
      # 指定数据源，使用 Parquet。
      source: parquet

      # 指定如何将点数据导入 Nebula Graph：Client 或 SST。
      sink: client
    }

    # 指定 Parquet 文件的路径。
    # 如果文件存储在 HDFS 上，用双引号括起路径，以 hdfs://开头，例如"hdfs://ip:port/xx/xx"。
    # 如果文件存储在本地，用双引号括起路径，以 file://开头，例如"file:///tmp/xx.csv"。
    path: "hdfs://192.168.11.13:9000/data/edge_follow.parquet"

    # 在 fields 里指定 Parquet 文件中 key 名称，其对应的 value 会作为 Nebula Graph 中指定属性的数据源。
    # 如果需要指定多个值，用英文逗号（,）隔开。
    fields: [degree]

    # 指定 Nebula Graph 中定义的属性名称。
    # fields 与 nebula.fields 的顺序必须一一对应。
    nebula.fields: [degree]

    # 指定一个列作为起始点和目的点的源。
    # vertex 的值必须与 Parquet 文件中的字段保持一致。
    # 目前，Nebula Graph 3.1.0仅支持字符串或整数类型的 VID。
    source: {
      field: src
    }
    target: {
      field: dst
    }

    # 指定一个列作为 rank 的源（可选）。
    #ranking: rank

    # 指定单批次写入 Nebula Graph 的最大边数量。
    batch: 256

    # 指定 Spark 分片数量。
    partition: 32
  }

  # 设置 Edge type serve 相关信息。
  {
    # 指定 Nebula Graph 中定义的 Edge type 名称。
    name: serve
    type: {
      # 指定数据源，使用 Parquet。
      source: parquet

      # 指定如何将点数据导入 Nebula Graph：Client 或 SST。
      sink: client
    }

    # 指定 Parquet 文件的路径。
    # 如果文件存储在 HDFS 上，用双引号括起路径，以 hdfs://开头，例如"hdfs://ip:port/xx/xx"。
    # 如果文件存储在本地，用双引号括起路径，以 file://开头，例如"file:///tmp/xx.csv"。
    path: "hdfs://192.168.11.13:9000/data/edge_serve.parquet"

    # 在 fields 里指定 Parquet 文件中 key 名称，其对应的 value 会作为 Nebula Graph 中指定属性的数据源。
    # 如果需要指定多个值，用英文逗号（,）隔开。
    fields: [start_year,end_year]

    # 指定 Nebula Graph 中定义的属性名称。
    # fields 与 nebula.fields 的顺序必须一一对应。
    nebula.fields: [start_year, end_year]

    # 指定一个列作为起始点和目的点的源。
    # vertex 的值必须与 Parquet 文件中的字段保持一致。
    # 目前，Nebula Graph 3.1.0仅支持字符串或整数类型的 VID。
    source: {
      field: src
    }
```


2022 vesoft Inc.

```
      target: {
        field: dst
      }

      # 指定一个列作为 rank 的源（可选）。
      #ranking: _c5

      # 指定单批次写入 Nebula Graph 的最大边数量。
      batch: 256

      # 指定 Spark 分片数量。
      partition: 32
    }

  ]
  # 如果需要添加更多边，请参考前面的配置进行添加。
}
```

步骤 **4**：向 **NEBULA GRAPH** 导入数据

运行如下命令将 Parquet 文件数据导入到 Nebula Graph 中。关于参数的说明，请参见导入命令参数。

```
${SPARK_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange <nebula-exchange-3.0.0.jar_path> -c <parquet_application.conf_path>
```

> **Note**
>
> JAR 包有两种获取方式：自行编译或者从 maven 仓库下载。

示例：

```
${SPARK_HOME}/bin/spark-submit  --master "local" --class com.vesoft.nebula.exchange.Exchange  /root/nebula-exchange/nebula-exchange/target/nebula-exchange-3.0.0.jar  -c /root/nebula-exchange/nebula-exchange/target/classes/parquet_application.conf
```

用户可以在返回信息中搜索 `batchSuccess.<tag_name/edge_name>`，确认成功的数量。例如 `batchSuccess.follow: 300` 。

步骤 **5**：（可选）验证数据

用户可以在 Nebula Graph 客户端（例如 Nebula Studio）中执行查询语句，确认数据是否已导入。例如：

```
GO FROM "player100" OVER follow;
```

用户也可以使用命令 `SHOW STATS` 查看统计数据。

步骤 **6**：（如有）在 **NEBULA GRAPH** 中重建索引

导入数据后，用户可以在 Nebula Graph 中重新创建并重建索引。详情请参见索引介绍。

最后更新: June 20, 2023

2022 vesoft Inc.

19.4.5 导入 HBase 数据

本文以一个示例说明如何使用 Exchange 将存储在 HBase 上的数据导入 Nebula Graph。

数据集

本文以 basketballplayer 数据集为例。

在本示例中，该数据集已经存入 HBase 中，以 `player`、`team`、`follow` 和 `serve` 四个表存储了所有点和边的信息。以下为各个表的部分数据。

```
hbase(main):002:0> scan "player"
ROW                              COLUMN+CELL
 player100                       column=cf:age, timestamp=1618881347530, value=42
 player100                       column=cf:name, timestamp=1618881354604, value=Tim Duncan
 player101                       column=cf:age, timestamp=1618881369124, value=36
 player101                       column=cf:name, timestamp=1618881379102, value=Tony Parker
 player102                       column=cf:age, timestamp=1618881386987, value=33
 player102                       column=cf:name, timestamp=1618881393370, value=LaMarcus Aldridge
 player103                       column=cf:age, timestamp=1618881402002, value=32
 player103                       column=cf:name, timestamp=1618881407882, value=Rudy Gay
 ...

hbase(main):003:0> scan "team"
ROW                              COLUMN+CELL
 team200                         column=cf:name, timestamp=1618881445563, value=Warriors
 team201                         column=cf:name, timestamp=1618881453636, value=Nuggets
 ...

hbase(main):004:0> scan "follow"
ROW                              COLUMN+CELL
 player100                       column=cf:degree, timestamp=1618881804853, value=95
 player100                       column=cf:dst_player, timestamp=1618881791522, value=player101
 player101                       column=cf:degree, timestamp=1618881824685, value=90
 player101                       column=cf:dst_player, timestamp=1618881816042, value=player102
 ...

hbase(main):005:0> scan "serve"
ROW                              COLUMN+CELL
 player100                       column=cf:end_year, timestamp=1618881899333, value=2016
 player100                       column=cf:start_year, timestamp=1618881890117, value=1997
 player100                       column=cf:teamid, timestamp=1618881875739, value=team204
 ...
```

环境配置

本文示例在 MacOS 下完成，以下是相关的环境配置信息：

- 硬件规格：
- CPU：1.7 GHz Quad-Core Intel Core i7
- 内存：16 GB
- Spark：2.4.7，单机版
- Hadoop：2.9.2，伪分布式部署
- HBase：2.2.7
- Nebula Graph：3.1.0。使用 Docker Compose 部署。

2022 vesoft Inc.

前提条件

开始导入数据之前，用户需要确认以下信息：

- 已经安装部署 Nebula Graph 并获取如下信息：
- Graph 服务和 Meta 服务的的 IP 地址和端口。
- 拥有 Nebula Graph 写权限的用户名和密码。
- 已经编译 Exchange。详情请参见编译 Exchange。本示例中使用 Exchange 3.0.0。
- 已经安装 Spark。
- 了解 Nebula Graph 中创建 Schema 的信息，包括 Tag 和 Edge type 的名称、属性等。
- 已经安装并开启 Hadoop 服务。

操作步骤

步骤 1：在 NEBULA GRAPH 中创建 SCHEMA

分析数据，按以下步骤在 Nebula Graph 中创建 Schema：

1. 确认 Schema 要素。Nebula Graph 中的 Schema 要素如下表所示。

要素	名称	属性
Tag	`player`	`name string, age int`
Tag	`team`	`name string`
Edge Type	`follow`	`degree int`
Edge Type	`serve`	`start_year int, end_year int`

2. 在 Nebula Graph 中创建一个图空间 **basketballplayer**，并创建一个 Schema，如下所示。

```
## 创建图空间
nebula> CREATE SPACE basketballplayer \
        (partition_num = 10, \
        replica_factor = 1, \
        vid_type = FIXED_STRING(30));

## 选择图空间 basketballplayer
nebula> USE basketballplayer;

## 创建 Tag player
nebula> CREATE TAG player(name string, age int);

## 创建 Tag team
nebula> CREATE TAG team(name string);

## 创建 Edge type follow
nebula> CREATE EDGE follow(degree int);

## 创建 Edge type serve
nebula> CREATE EDGE serve(start_year int, end_year int);
```

更多信息，请参见快速开始。

步骤 2：修改配置文件

编译 Exchange 后，复制 `target/classes/application.conf` 文件设置 HBase 数据源相关的配置。在本示例中，复制的文件名为 `hbase_application.conf` 。各个配置项的详细说明请参见配置说明。

```
{
  # Spark 相关配置
  spark: {
    app: {
      name: Nebula Exchange 3.0.0
    }
    driver: {
      cores: 1
      maxResultSize: 1G
```


2022 vesoft Inc.

```
    }
    cores: {
      max: 16
    }
  }

  # Nebula Graph 相关配置
  nebula: {
    address:{
      # 以下为 Nebula Graph 的 Graph 服务和 Meta 服务所在机器的 IP 地址及端口。
      # 如果有多个地址，格式为 "ip1:port","ip2:port","ip3:port"。
      # 不同地址之间以英文逗号 (,) 隔开。
      graph:["127.0.0.1:9669"]
      meta:["127.0.0.1:9559"]
    }
    # 填写的账号必须拥有 Nebula Graph 相应图空间的写数据权限。
    user: root
    pswd: nebula
    # 填写 Nebula Graph 中需要写入数据的图空间名称。
    space: basketballplayer
    connection: {
      timeout: 3000
      retry: 3
    }
    execution: {
      retry: 3
    }
    error: {
      max: 32
      output: /tmp/errors
    }
    rate: {
      limit: 1024
      timeout: 1000
    }
  }
  # 处理点
  tags: [
    # 设置 Tag player 相关信息。
    # 如果需要将 rowkey 设置为数据源，请填写“rowkey”，列族内的列请填写实际列名。
    {
      # Nebula Graph 中对应的 Tag 名称。
      name: player
      type: {
        # 指定数据源文件格式，设置为 HBase。
        source: hbase
        # 指定如何将点数据导入 Nebula Graph：Client 或 SST。
        sink: client
      }
      host:192.168.*.*
      port:2181
      table:"player"
      columnFamily:"cf"

      # 在 fields 里指定 player 表中的列名称，其对应的 value 会作为 Nebula Graph 中指定属性。
      # fields 和 nebula.fields 里的配置必须一一对应。
      # 如果需要指定多个列名称，用英文逗号 (,) 隔开。
      fields: [age,name]
      nebula.fields: [age,name]

      # 指定表中某一列数据为 Nebula Graph 中点 VID 的来源。
      # 例如 rowkey 作为 VID 的来源，请填写“rowkey”。
      vertex:{
          field:rowkey
      }

      # 单批次写入 Nebula Graph 的数据条数。
      batch: 256

      # Spark 分区数量
      partition: 32
    }
    # 设置 Tag team 相关信息。
    {
      name: team
      type: {
        source: hbase
        sink: client
      }
      host:192.168.*.*
      port:2181
      table:"team"
      columnFamily:"cf"
      fields: [name]
      nebula.fields: [name]
      vertex:{
          field:rowkey
      }
      batch: 256
      partition: 32
    }

  ]
```


2022 vesoft Inc.

```
  # 处理边数据
  edges: [
    # 设置 Edge type follow 相关信息
    {
      # Nebula Graph 中对应的 Edge type 名称。
      name: follow

      type: {
        # 指定数据源文件格式，设置为 HBase。
        source: hbase

        # 指定边数据导入 Nebula Graph 的方式，
        # 指定如何将点数据导入 Nebula Graph：Client 或 SST。
        sink: client
      }

      host:192.168.*.*
      port:2181
      table:"follow"
      columnFamily:"cf"

      # 在 fields 里指定 follow 表中的列名称，其对应的 value 会作为 Nebula Graph 中指定属性。
      # fields 和 nebula.fields 里的配置必须一一对应。
      # 如果需要指定多个列名称，用英文逗号（,）隔开。
      fields: [degree]
      nebula.fields: [degree]

      # 在 source 里，将 follow 表中某一列作为边的起始点数据源。示例使用 rowkey。
      # 在 target 里，将 follow 表中某一列作为边的目的点数据源。示例使用列 dst_player。
      source:{
          field:rowkey
      }

      target:{
          field:dst_player
      }

      # 指定一个列作为 rank 的源（可选）。
      #ranking: rank

      # 单批次写入 Nebula Graph 的数据条数。
      batch: 256

      # Spark 分区数量
      partition: 32
    }

    # 设置 Edge type serve 相关信息
    {
      name: serve
      type: {
        source: hbase
        sink: client
      }
      host:192.168.*.*
      port:2181
      table:"serve"
      columnFamily:"cf"

      fields: [start_year,end_year]
      nebula.fields: [start_year,end_year]
      source:{
          field:rowkey
      }

      target:{
          field:teamid
      }

      # 指定一个列作为 rank 的源（可选）。
      #ranking: rank

      batch: 256
      partition: 32
    }
  ]
}
```

步骤 3：向 NEBULA GRAPH 导入数据

运行如下命令将 HBase 数据导入到 Nebula Graph 中。关于参数的说明，请参见导入命令参数。

```
${SPARK_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange <nebula-exchange-3.0.0.jar_path> -c <hbase_application.conf_path>
```

> **Note**

JAR 包有两种获取方式：自行编译或者从 maven 仓库下载。

2022 vesoft Inc.

示例：

```
${SPARK_HOME}/bin/spark-submit  --master "local" --class com.vesoft.nebula.exchange.Exchange  /root/nebula-exchange/nebula-exchange/target/nebula-exchange-3.0.0.jar  -c /root/nebula-exchange/nebula-exchange/target/classes/hbase_application.conf
```

用户可以在返回信息中搜索 `batchSuccess.<tag_name/edge_name>`，确认成功的数量。例如 `batchSuccess.follow: 300`。

步骤 **4**：（可选）验证数据

用户可以在 Nebula Graph 客户端（例如 Nebula Studio）中执行查询语句，确认数据是否已导入。例如：

```
GO FROM "player100" OVER follow;
```

用户也可以使用命令 `SHOW STATS` 查看统计数据。

步骤 **5**：（如有）在 **NEBULA GRAPH** 中重建索引

导入数据后，用户可以在 Nebula Graph 中重新创建并重建索引。详情请参见索引介绍。

最后更新: June 20, 2023

2022 vesoft Inc.

19.4.6 导入 MySQL/PostgreSQL 数据

本文以一个示例说明如何使用 Exchange 将存储在 MySQL 上的数据导入 Nebula Graph，也适用于从 PostgreSQL 导出数据到 Nebula Graph。

数据集

本文以 basketballplayer 数据集为例。

在本示例中，该数据集已经存入 MySQL 中名为 `basketball` 的数据库中，以 `player` 、`team` 、`follow` 和 `serve` 四个表存储了所有点和边的信息。以下为各个表的结构。

```
mysql> desc player;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| playerid | varchar(30) | YES  |     | NULL    |       |
| age      | int         | YES  |     | NULL    |       |
| name     | varchar(30) | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+

mysql> desc team;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| teamid | varchar(30) | YES  |     | NULL    |       |
| name   | varchar(30) | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+

mysql> desc follow;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| src_player | varchar(30) | YES  |     | NULL    |       |
| dst_player | varchar(30) | YES  |     | NULL    |       |
| degree     | int         | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+

mysql> desc serve;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| playerid   | varchar(30) | YES  |     | NULL    |       |
| teamid     | varchar(30) | YES  |     | NULL    |       |
| start_year | int         | YES  |     | NULL    |       |
| end_year   | int         | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
```

环境配置

本文示例在 MacOS 下完成，以下是相关的环境配置信息：

- 硬件规格：
- CPU：1.7 GHz Quad-Core Intel Core i7
- 内存：16 GB
- Spark：2.4.7，单机版
- Hadoop：2.9.2，伪分布式部署
- MySQL： 8.0.23
- Nebula Graph：3.1.0。使用 Docker Compose 部署。

2022 vesoft Inc.

前提条件

开始导入数据之前，用户需要确认以下信息：

- 已经安装部署 Nebula Graph 并获取如下信息：
- Graph 服务和 Meta 服务的的 IP 地址和端口。
- 拥有 Nebula Graph 写权限的用户名和密码。
- 已经编译 Exchange。详情请参见编译 Exchange。本示例中使用 Exchange 3.0.0。
- 已经安装 Spark。
- 了解 Nebula Graph 中创建 Schema 的信息，包括 Tag 和 Edge type 的名称、属性等。
- 已经安装并开启 Hadoop 服务。

操作步骤

步骤 1：在 NEBULA GRAPH 中创建 SCHEMA

分析数据，按以下步骤在 Nebula Graph 中创建 Schema：

1. 确认 Schema 要素。Nebula Graph 中的 Schema 要素如下表所示。

要素	名称	属性
Tag	`player`	`name string, age int`
Tag	`team`	`name string`
Edge Type	`follow`	`degree int`
Edge Type	`serve`	`start_year int, end_year int`

2. 在 Nebula Graph 中创建一个图空间 **basketballplayer**，并创建一个 Schema，如下所示。

```
## 创建图空间
nebula> CREATE SPACE basketballplayer \
        (partition_num = 10, \
        replica_factor = 1, \
        vid_type = FIXED_STRING(30));

## 选择图空间 basketballplayer
nebula> USE basketballplayer;

## 创建 Tag player
nebula> CREATE TAG player(name string, age int);

## 创建 Tag team
nebula> CREATE TAG team(name string);

## 创建 Edge type follow
nebula> CREATE EDGE follow(degree int);

## 创建 Edge type serve
nebula> CREATE EDGE serve(start_year int, end_year int);
```

更多信息，请参见快速开始。

步骤 2：修改配置文件

编译 Exchange 后，复制 `target/classes/application.conf` 文件设置 MySQL 数据源相关的配置。在本示例中，复制的文件名为 `mysql_application.conf`。各个配置项的详细说明请参见配置说明。

```
{
  # Spark 相关配置
  spark: {
    app: {
      name: Nebula Exchange 3.0.0
    }
    driver: {
      cores: 1
      maxResultSize: 1G
```


2022 vesoft Inc.

```
    }
    cores: {
      max: 16
    }
  }

# Nebula Graph 相关配置
  nebula: {
    address:{
      # 以下为 Nebula Graph 的 Graph 服务和 Meta 服务所在机器的 IP 地址及端口。
      # 如果有多个地址，格式为 "ip1:port","ip2:port","ip3:port"。
      # 不同地址之间以英文逗号 (,) 隔开。
      graph:["127.0.0.1:9669"]
      meta:["127.0.0.1:9559"]
    }
    # 填写的账号必须拥有 Nebula Graph 相应图空间的写数据权限。
    user: root
    pswd: nebula
    # 填写 Nebula Graph 中需要写入数据的图空间名称。
    space: basketballplayer
    connection: {
      timeout: 3000
      retry: 3
    }
    execution: {
      retry: 3
    }
    error: {
      max: 32
      output: /tmp/errors
    }
    rate: {
      limit: 1024
      timeout: 1000
    }
  }
  # 处理点
  tags: [
    # 设置 Tag player 相关信息。
    {
      # Nebula Graph 中对应的 Tag 名称。
      name: player
      type: {
        # 指定数据源文件格式，设置为 MySQL。
        source: mysql
        # 指定如何将点数据导入 Nebula Graph：Client 或 SST。
        sink: client
      }

      host:192.168.*.*
      port:3306
      database:"basketball"
      table:"player"
      user:"test"
      password:"123456"
      sentence:"select playerid, age, name from player order by playerid"

      # 在 fields 里指定 player 表中的列名称，其对应的 value 会作为 Nebula Graph 中指定属性。
      # fields 和 nebula.fields 里的配置必须一一对应。
      # 如果需要指定多个列名称，用英文逗号 (,) 隔开。
      fields: [age,name]
      nebula.fields: [age,name]

      # 指定表中某一列数据为 Nebula Graph 中点 VID 的来源。
      vertex: {
        field:playerid
      }

      # 单批次写入 Nebula Graph 的数据条数。
      batch: 256

      # Spark 分区数量
      partition: 32
    }
    # 设置 Tag team 相关信息。
    {
      name: team
      type: {
        source: mysql
        sink: client
      }

      host:192.168.*.*
      port:3306
      database:"basketball"
      table:"team"
      user:"test"
      password:"123456"
      sentence:"select teamid, name from team order by teamid"

      fields: [name]
      nebula.fields: [name]
      vertex: {
        field: teamid
```


2022 vesoft Inc.

```
      }
      batch: 256
      partition: 32
    }

  ]

  # 处理边数据
  edges: [
    # 设置 Edge type follow 相关信息
    {
      # Nebula Graph 中对应的 Edge type 名称。
      name: follow

      type: {
        # 指定数据源文件格式，设置为 MySQL。
        source: mysql

        # 指定边数据导入 Nebula Graph 的方式，
        # 指定如何将点数据导入 Nebula Graph：Client 或 SST。
        sink: client
      }

      host:192.168.*.*
      port:3306
      database:"basketball"
      table:"follow"
      user:"test"
      password:"123456"
      sentence:"select src_player,dst_player,degree from follow order by src_player"

      # 在 fields 里指定 follow 表中的列名称，其对应的 value 会作为 Nebula Graph 中指定属性。
      # fields 和 nebula.fields 里的配置必须一一对应。
      # 如果需要指定多个列名称，用英文逗号（,）隔开。
      fields: [degree]
      nebula.fields: [degree]

      # 在 source 里，将 follow 表中某一列作为边的起始点数据源。
      # 在 target 里，将 follow 表中某一列作为边的目的点数据源。
      source: {
        field: src_player
      }

      target: {
        field: dst_player
      }

      # 指定一个列作为 rank 的源（可选）。
      #ranking: rank

      # 单批次写入 Nebula Graph 的数据条数。
      batch: 256

      # Spark 分区数量
      partition: 32
    }

    # 设置 Edge type serve 相关信息
    {
      name: serve
      type: {
        source: mysql
        sink: client
      }

      host:192.168.*.*
      port:3306
      database:"basketball"
      table:"serve"
      user:"test"
      password:"123456"
      sentence:"select playerid,teamid,start_year,end_year from serve order by playerid"
      fields: [start_year,end_year]
      nebula.fields: [start_year,end_year]
      source: {
        field: playerid
      }
      target: {
        field: teamid
      }

      # 指定一个列作为 rank 的源（可选）。
      #ranking: rank

      batch: 256
      partition: 32
    }
  ]
}
```


2022 vesoft Inc.

步骤 3：向 NEBULA GRAPH 导入数据

运行如下命令将 MySQL 数据导入到 Nebula Graph 中。关于参数的说明，请参见导入命令参数。

```
${SPARK_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange <nebula-exchange-3.0.0.jar_path> -c <mysql_application.conf_path>
```

> **Note**
>
> JAR 包有两种获取方式：自行编译或者从 maven 仓库下载。

示例：

```
${SPARK_HOME}/bin/spark-submit  --master "local" --class com.vesoft.nebula.exchange.Exchange  /root/nebula-exchange/nebula-exchange/target/nebula-exchange-3.0.0.jar  -c /root/nebula-exchange/nebula-exchange/target/classes/mysql_application.conf
```

用户可以在返回信息中搜索 `batchSuccess.<tag_name/edge_name>`，确认成功的数量。例如 `batchSuccess.follow: 300` 。

步骤 4：（可选）验证数据

用户可以在 Nebula Graph 客户端（例如 Nebula Studio）中执行查询语句，确认数据是否已导入。例如：

```
GO FROM "player100" OVER follow;
```

用户也可以使用命令 `SHOW STATS` 查看统计数据。

步骤 5：（如有）在 NEBULA GRAPH 中重建索引

导入数据后，用户可以在 Nebula Graph 中重新创建并重建索引。详情请参见索引介绍。

最后更新: June 20, 2023

2022 vesoft Inc.

19.4.7 导入 ClickHouse 数据

本文以一个示例说明如何使用 Exchange 将存储在 ClickHouse 上的数据导入 Nebula Graph。

数据集

本文以 basketballplayer 数据集为例。

环境配置

本文示例在 MacOS 下完成，以下是相关的环境配置信息：

- 硬件规格：
- CPU：1.7 GHz Quad-Core Intel Core i7
- 内存：16 GB
- Spark：2.4.7，单机版
- Hadoop：2.9.2，伪分布式部署
- ClickHouse：docker 部署 yandex/clickhouse-server tag: latest(2021.07.01)
- Nebula Graph：3.1.0。使用 Docker Compose 部署。

前提条件

开始导入数据之前，用户需要确认以下信息：

- 已经安装部署 Nebula Graph 并获取如下信息：
- Graph 服务和 Meta 服务的的 IP 地址和端口。
- 拥有 Nebula Graph 写权限的用户名和密码。
- 已经编译 Exchange。详情请参见编译 Exchange。本示例中使用 Exchange 3.0.0。
- 已经安装 Spark。
- 了解 Nebula Graph 中创建 Schema 的信息，包括 Tag 和 Edge type 的名称、属性等。
- 已经安装并开启 Hadoop 服务。

2022 vesoft Inc.

操作步骤

步骤 1：在 NEBULA GRAPH 中创建 SCHEMA

分析数据，按以下步骤在 Nebula Graph 中创建 Schema：

1. 确认 Schema 要素。Nebula Graph 中的 Schema 要素如下表所示。

要素	名称	属性
Tag	`player`	`name string, age int`
Tag	`team`	`name string`
Edge Type	`follow`	`degree int`
Edge Type	`serve`	`start_year int, end_year int`

2. 在 Nebula Graph 中创建一个图空间 **basketballplayer**，并创建一个 Schema，如下所示。

```
## 创建图空间
nebula> CREATE SPACE basketballplayer \
        (partition_num = 10, \
        replica_factor = 1, \
        vid_type = FIXED_STRING(30));

## 选择图空间 basketballplayer
nebula> USE basketballplayer;

## 创建 Tag player
nebula> CREATE TAG player(name string, age int);

## 创建 Tag team
nebula> CREATE TAG team(name string);

## 创建 Edge type follow
nebula> CREATE EDGE follow(degree int);

## 创建 Edge type serve
nebula> CREATE EDGE serve(start_year int, end_year int);
```

更多信息，请参见快速开始。

步骤 2：修改配置文件

编译 Exchange 后，复制 `target/classes/application.conf` 文件设置 ClickHouse 数据源相关的配置。在本示例中，复制的文件名为 `clickhouse_application.conf`。各个配置项的详细说明请参见配置说明。

```
{
  # Spark 相关配置
  spark: {
    app: {
      name: Nebula Exchange 3.0.0
    }
    driver: {
      cores: 1
      maxResultSize: 1G
    }
    cores: {
      max: 16
    }
  }

# Nebula Graph 相关配置
  nebula: {
    address:{
      # 以下为 Nebula Graph 的 Graph 服务和 Meta 服务所在机器的 IP 地址及端口。
      # 如果有多个地址，格式为 "ip1:port","ip2:port","ip3:port"。
      # 不同地址之间以英文逗号 (,) 隔开。
      graph:["127.0.0.1:9669"]
      meta:["127.0.0.1:9559"]
    }
    # 填写的账号必须拥有 Nebula Graph 相应图空间的写数据权限。
    user: root
    pswd: nebula
    # 填写 Nebula Graph 中需要写入数据的图空间名称。
    space: basketballplayer
    connection: {
      timeout: 3000
      retry: 3
    }
    execution: {
      retry: 3
```


2022 vesoft Inc.

```
    }
    error: {
      max: 32
      output: /tmp/errors
    }
    rate: {
      limit: 1024
      timeout: 1000
    }
  }
  # 处理点
  tags: [
    # 设置 Tag player 相关信息。
    {
      name: player
      type: {
        # 指定数据源文件格式，设置为 ClickHouse。
        source: clickhouse
        # 指定如何将点数据导入 Nebula Graph：Client 或 SST。
        sink: client
      }

      # ClickHouse 的 JDBC URL
      url:"jdbc:clickhouse://192.168.*.*:8123/basketballplayer"

      user:"user"
      password:"123456"

      # ClickHouse 分区数
      numPartition:"5"

      table:"player"
      sentence:"select * from player"

      # 在 fields 里指定 player 表中的列名称，其对应的 value 会作为 Nebula Graph 中指定属性。
      # fields 和 nebula.fields 里的配置必须一一对应。
      # 如果需要指定多个列名称，用英文逗号（,）隔开。
      fields: [name,age]
      nebula.fields: [name,age]

      # 指定表中某一列数据为 Nebula Graph 中点 VID 的来源。
      vertex: {
        field:playerid
        # policy:hash
      }

      # 单批次写入 Nebula Graph 的数据条数。
      batch: 256

      # Spark 分区数量
      partition: 32
    }

    # 设置 Tag team 相关信息。
    {
      name: team
      type: {
        source: clickhouse
        sink: client
      }
      url:"jdbc:clickhouse://192.168.*.*:8123/basketballplayer"
      user:"user"
      password:"123456"
      numPartition:"5"
      table:"team"
      sentence:"select * from team"
      fields: [name]
      nebula.fields: [name]
      vertex: {
        field:teamid
      }
      batch: 256
      partition: 32
    }
  ]

  # 处理边数据
  edges: [
    # 设置 Edge type follow 相关信息
    {
      # Nebula Graph 中对应的 Edge type 名称。
      name: follow

      type: {
        # 指定数据源文件格式，设置为 ClickHouse。
        source: clickhouse

        # 指定边数据导入 Nebula Graph 的方式，
        # 指定如何将点数据导入 Nebula Graph：Client 或 SST。
        sink: client
      }

      # ClickHouse 的 JDBC URL
      url:"jdbc:clickhouse://192.168.*.*:8123/basketballplayer"
```


2022 vesoft Inc.

```
    user:"user"
    password:"123456"

    # ClickHouse 分区数
    numPartition:"5"

    table:"follow"
    sentence:"select * from follow"

    # 在 fields 里指定 follow 表中的列名称，其对应的 value 会作为 Nebula Graph 中指定属性。
    # fields 和 nebula.fields 里的配置必须一一对应。
    # 如果需要指定多个列名称，用英文逗号（,）隔开。
    fields: [degree]
    nebula.fields: [degree]

    # 在 source 里，将 follow 表中某一列作为边的起始点数据源。
    source: {
      field:src_player
    }

    # 在 target 里，将 follow 表中某一列作为边的目的点数据源。
    target: {
      field:dst_player
    }

    # 指定一个列作为 rank 的源（可选）。
    #ranking: rank

    # 单批次写入 Nebula Graph 的数据条数。
    batch: 256

    # Spark 分区数量
    partition: 32
  }

  # 设置 Edge type serve 相关信息
  {
    name: serve
    type: {
      source: clickhouse
      sink: client
    }
    url:"jdbc:clickhouse://192.168.*.*:8123/basketballplayer"
    user:"user"
    password:"123456"
    numPartition:"5"
    sentence:"select * from serve"
    fields: [start_year,end_year]
    nebula.fields: [start_year,end_year]
    source: {
      field:playerid
    }
    target: {
      field:teamid
    }

    # 指定一个列作为 rank 的源（可选）。
    #ranking: rank

    batch: 256
    partition: 32
  }
 ]
}
```

步骤 **3**：向 **NEBULA GRAPH** 导入数据

运行如下命令将 ClickHouse 数据导入到 Nebula Graph 中。关于参数的说明，请参见导入命令参数。

```
${SPARK_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange <nebula-exchange-3.0.0.jar_path> -c <clickhouse_application.conf_path>
```

Note

JAR 包有两种获取方式：自行编译或者从 maven 仓库下载。

示例：

```
${SPARK_HOME}/bin/spark-submit  --master "local" --class com.vesoft.nebula.exchange.Exchange  /root/nebula-exchange/nebula-exchange/target/nebula-exchange-3.0.0.jar  -c /root/nebula-exchange/nebula-exchange/target/classes/clickhouse_application.conf
```

用户可以在返回信息中搜索 `batchSuccess.<tag_name/edge_name>`，确认成功的数量。例如 `batchSuccess.follow: 300` 。

2022 vesoft Inc.

步骤 **4**: （可选）验证数据

用户可以在 Nebula Graph 客户端（例如 Nebula Studio）中执行查询语句，确认数据是否已导入。例如：

```
GO FROM "player100" OVER follow;
```

用户也可以使用命令 `SHOW STATS` 查看统计数据。

步骤 **5**: （如有）在 **NEBULA GRAPH** 中重建索引

导入数据后，用户可以在 Nebula Graph 中重新创建并重建索引。详情请参见索引介绍。

最后更新: June 20, 2023

2022 vesoft Inc.

19.4.8 导入 Neo4j 数据

本文以一个示例说明如何使用 Exchange 将存储在 Neo4j 的数据导入 Nebula Graph。

实现方法

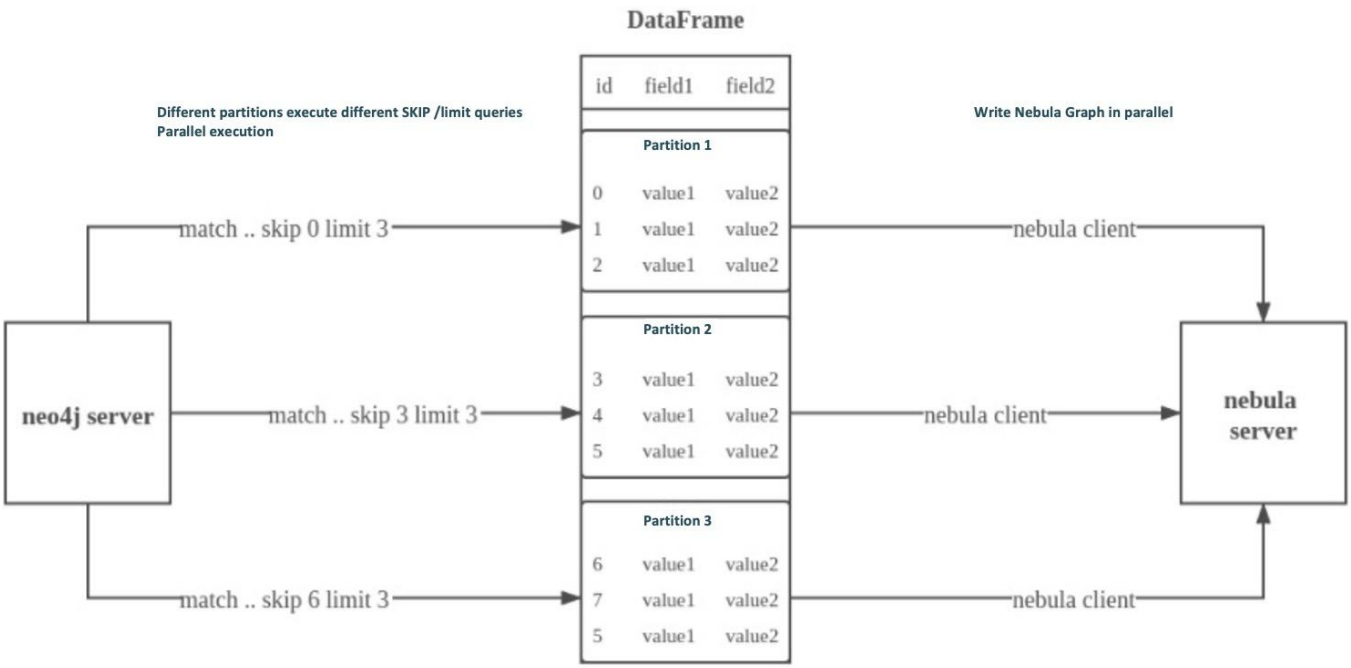
Exchange 使用 **Neo4j Driver 4.0.1** 实现对 Neo4j 数据的读取。执行批量导出之前，用户需要在配置文件中写入针对标签（label）和关系类型（Relationship Type）自动执行的 Cypher 语句，以及 Spark 分区数，提高数据导出性能。

Exchange 读取 Neo4j 数据时需要完成以下工作:

1. Exchange 中的 Reader 会将配置文件中 `exec` 部分的 Cypher `RETURN` 语句后面的语句替换为 `COUNT(*)`，并执行这个语句，从而获取数据总量，再根据 Spark 分区数量计算每个分区的起始偏移量和大小。
2. （可选）如果用户配置了 `check_point_path` 目录，Reader 会读取目录中的文件。如果处于续传状态，Reader 会计算每个 Spark 分区应该有的偏移量和大小。
3. 在每个 Spark 分区里，Exchange 中的 Reader 会在 Cypher 语句后面添加不同的 `SKIP` 和 `LIMIT` 语句，调用 Neo4j Driver 并行执行，将数据分布到不同的 Spark 分区中。
4. Reader 最后将返回的数据处理成 DataFrame。

至此，Exchange 即完成了对 Neo4j 数据的导出。之后，数据被并行写入 Nebula Graph 数据库中。

整个过程如下图所示。



数据集

本文以 basketballplayer 数据集为例。

2022 vesoft Inc.

环境配置

本文示例在 MacOS 下完成，以下是相关的环境配置信息：

- 硬件规格：
- CPU：Intel(R) Xeon(R) CPU E5-2697 v3 @ 2.60GHz
- CPU 内核数：14
- 内存：251 GB
- Spark：单机版，2.4.6 pre-build for Hadoop 2.7
- Neo4j：3.5.20 Community Edition
- Nebula Graph：3.1.0。使用 Docker Compose 部署。

前提条件

开始导入数据之前，用户需要确认以下信息：

- 已经安装部署 Nebula Graph 并获取如下信息：
- Graph 服务和 Meta 服务的 IP 地址和端口。
- 拥有 Nebula Graph 写权限的用户名和密码。
- 已经编译 Exchange。详情请参见编译 Exchange。本示例中使用 Exchange 3.0.0。
- 已经安装 Spark。
- 了解 Nebula Graph 中创建 Schema 的信息，包括 Tag 和 Edge type 的名称、属性等。

2022 vesoft Inc.

操作步骤

步骤 1：在 NEBULA GRAPH 中创建 SCHEMA

分析数据，按以下步骤在 Nebula Graph 中创建 Schema：

1. 确认 Schema 要素。Nebula Graph 中的 Schema 要素如下表所示。

要素	名称	属性
Tag	`player`	`name string, age int`
Tag	`team`	`name string`
Edge Type	`follow`	`degree int`
Edge Type	`serve`	`start_year int, end_year int`

2. 使用 Nebula Console 创建一个图空间 **basketballplayer**，并创建一个 Schema，如下所示。

```
## 创建图空间
nebula> CREATE SPACE basketballplayer \
        (partition_num = 10, \
        replica_factor = 1, \
        vid_type = FIXED_STRING(30));

## 选择图空间 basketballplayer
nebula> USE basketballplayer;

## 创建 Tag player
nebula> CREATE TAG player(name string, age int);

## 创建 Tag team
nebula> CREATE TAG team(name string);

## 创建 Edge type follow
nebula> CREATE EDGE follow(degree int);

## 创建 Edge type serve
nebula> CREATE EDGE serve(start_year int, end_year int);
```

更多信息，请参见快速开始。

步骤 2：配置源数据

为了提高 Neo4j 数据的导出速度，在 Neo4j 数据库中为相应属性创建索引。详细信息，参考 Neo4j 用户手册。

步骤 3：修改配置文件

编译 Exchange 后，复制 `target/classes/application.conf` 文件设置数据源相关的配置。在本示例中，复制的文件名为 `neo4j_application.conf`。各个配置项的详细说明请参见配置说明。

```
{
  # Spark 相关配置
  spark: {
    app: {
      name: Nebula Exchange 3.0.0
    }

    driver: {
      cores: 1
      maxResultSize: 1G
    }

    executor: {
        memory:1G
    }

    cores:{
      max: 16
    }
  }

  # Nebula Graph 相关配置
  nebula: {
    address:{
      graph:["127.0.0.1:9669"]
      meta:["127.0.0.1:9559"]
    }
    user: root
    pswd: nebula
    space: basketballplayer
```


2022 vesoft Inc.

```
  connection: {
    timeout: 3000
    retry: 3
  }

  execution: {
    retry: 3
  }

  error: {
    max: 32
    output: /tmp/errors
  }

  rate: {
    limit: 1024
    timeout: 1000
  }
}

# 处理点
tags: [

  # 设置 Tag player 相关信息。
  {
    name: player
    type: {
      source: neo4j
      sink: client
    }
    server: "bolt://192.168.*.*:7687"
    user: neo4j
    password:neo4j
    database:neo4j
    exec: "match (n:player) return n.id as id, n.age as age, n.name as name"
    fields: [age,name]
    nebula.fields: [age,name]
    vertex: {
      field:id
    }
    partition: 10
    batch: 1000
    check_point_path: /tmp/test
 }
# 设置 Tag team 相关信息。
{
    name: team
    type: {
      source: neo4j
      sink: client
    }
    server: "bolt://192.168.*.*:7687"
    user: neo4j
    password:neo4j
    database:neo4j
    exec: "match (n:team) return n.id as id,n.name as name"
    fields: [name]
    nebula.fields: [name]
    vertex: {
      field:id
    }
    partition: 10
    batch: 1000
    check_point_path: /tmp/test
 }
]

# 处理边数据
edges: [
  # 设置 Edge type follow 相关信息
  {
    name: follow
    type: {
      source: neo4j
      sink: client
    }
    server: "bolt://192.168.*.*:7687"
    user: neo4j
    password:neo4j
    database:neo4j
    exec: "match (a:player)-[r:follow]->(b:player) return a.id as src, b.id as dst, r.degree as degree  order by id(r)"
    fields: [degree]
    nebula.fields: [degree]
    source: {
      field: src
    }
    target: {
      field: dst
    }
    #ranking: rank
    partition: 10
    batch: 1000
    check_point_path: /tmp/test
```


2022 vesoft Inc.

```
    }
    # 设置 Edge type serve 相关信息
    {
       name: serve
       type: {
         source: neo4j
         sink: client
       }
       server: "bolt://192.168.*.*:7687"
       user: neo4j
       password:neo4j
       database:neo4j
       exec: "match (a:player)-[r:serve]->(b:team) return a.id as src, b.id as dst, r.start_year as start_year, r.end_year as end_year  order by id(r)"
       fields: [start_year,end_year]
       nebula.fields: [start_year,end_year]
       source: {
         field: src
       }
       target: {
         field: dst
       }
       #ranking: rank
       partition: 10
       batch: 1000
       check_point_path: /tmp/test
     }
   ]
}
```

exec 配置说明

在配置 `tags.exec` 或者 `edges.exec` 参数时，需要填写 Cypher 查询语句。为了保证每次查询结果排序一致，并且为了防止在导入时丢失数据，强烈建议在 Cypher 查询语句中加入 `ORDER BY` 子句，同时，为了提高数据导入效率，最好选取有索引的属性作为排序的属性。如果没有索引，用户也可以观察默认的排序，选择合适的属性用于排序，以提高效率。如果默认的排序找不到规律，用户可以根据点或关系的 ID 进行排序，并且将 `partition` 设置为一个尽量小的值，减轻 Neo4j 的排序压力。

说明：使用 `ORDER BY` 子句会延长数据导入的时间。

另外，Exchange 需要在不同 Spark 分区执行不同 `SKIP` 和 `LIMIT` 的 Cypher 语句，所以在 `tags.exec` 和 `edges.exec` 对应的 Cypher 语句中不能含有 `SKIP` 和 `LIMIT` 子句。

tags.vertex 或 **edges.vertex** 配置说明

Nebula Graph 在创建点和边时会将 ID 作为唯一主键，如果主键已存在则会覆盖该主键中的数据。所以，假如将某个 Neo4j 属性值作为 Nebula Graph 的 ID，而这个属性值在 Neo4j 中是有重复的，就会导致重复 ID，它们对应的数据有且只有一条会存入 Nebula Graph 中，其它的则会被覆盖掉。由于数据导入过程是并发地往 Nebula Graph 中写数据，最终保存的数据并不能保证是 Neo4j 中最新的数据。

check_point_path 配置说明

如果启用了断点续传功能，为避免数据丢失，在断点和续传之间，数据库不应该改变状态，例如不能添加数据或删除数据，同时，不能更改 `partition` 数量配置。

步骤 **4**：向 **NEBULA GRAPH** 导入数据

运行如下命令将文件数据导入到 Nebula Graph 中。关于参数的说明，请参见导入命令参数。

```
${SPARK_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange <nebula-exchange-3.0.0.jar_path> -c <neo4j_application.conf_path>
```

> **Note**
>
> JAR 包有两种获取方式：自行编译或者从 maven 仓库下载。

示例：

```
${SPARK_HOME}/bin/spark-submit  --master "local" --class com.vesoft.nebula.exchange.Exchange  /root/nebula-exchange/nebula-exchange/target/nebula-exchange-3.0.0.jar  -c /root/nebula-exchange/nebula-exchange/target/classes/neo4j_application.conf
```

用户可以在返回信息中搜索 `batchSuccess.<tag_name/edge_name>`，确认成功的数量。例如 `batchSuccess.follow: 300` 。

步骤 **5**：（可选）验证数据

用户可以在 Nebula Graph 客户端（例如 Nebula Studio）中执行查询语句，确认数据是否已导入。例如：

2022 vesoft Inc.

```
GO FROM "player100" OVER follow;
```

用户也可以使用命令 `SHOW STATS` 查看统计数据。

步骤 **6**: （如有）在 **NEBULA GRAPH** 中重建索引

导入数据后，用户可以在 Nebula Graph 中重新创建并重建索引。详情请参见索引介绍。

最后更新: June 20, 2023

2022 vesoft Inc.

19.4.9 导入 Hive 数据

本文以一个示例说明如何使用 Exchange 将存储在 Hive 上的数据导入 Nebula Graph。

数据集

本文以 basketballplayer 数据集为例。

在本示例中，该数据集已经存入 Hive 中名为 `basketball` 的数据库中，以 `player` 、`team` 、`follow` 和 `serve` 四个表存储了所有点和边的信息。以下为各个表的结构。

```
scala> spark.sql("describe basketball.player").show
+--------+---------+-------+
|col_name|data_type|comment|
+--------+---------+-------+
|playerid|   string|   null|
|     age|   bigint|   null|
|    name|   string|   null|
+--------+---------+-------+

scala> spark.sql("describe basketball.team").show
+----------+---------+-------+
|  col_name|data_type|comment|
+----------+---------+-------+
|    teamid|   string|   null|
|      name|   string|   null|
+----------+---------+-------+

scala> spark.sql("describe basketball.follow").show
+----------+---------+-------+
|  col_name|data_type|comment|
+----------+---------+-------+
|src_player|   string|   null|
|dst_player|   string|   null|
|    degree|   bigint|   null|
+----------+---------+-------+

scala> spark.sql("describe basketball.serve").show
+----------+---------+-------+
|  col_name|data_type|comment|
+----------+---------+-------+
|  playerid|   string|   null|
|    teamid|   string|   null|
|start_year|   bigint|   null|
|  end_year|   bigint|   null|
+----------+---------+-------+
```

说明：Hive 的数据类型 `bigint` 与 Nebula Graph 的 `int` 对应。

环境配置

本文示例在 MacOS 下完成，以下是相关的环境配置信息：

- 硬件规格：
- CPU：1.7 GHz Quad-Core Intel Core i7
- 内存：16 GB
- Spark：2.4.7，单机版
- Hadoop：2.9.2，伪分布式部署
- Hive：2.3.7，Hive Metastore 数据库为 MySQL 8.0.22
- Nebula Graph：3.1.0。使用 Docker Compose 部署。

2022 vesoft Inc.

前提条件

开始导入数据之前，用户需要确认以下信息：

- 已经安装部署 Nebula Graph 并获取如下信息：
- Graph 服务和 Meta 服务的的 IP 地址和端口。
- 拥有 Nebula Graph 写权限的用户名和密码。
- 已经编译 Exchange。详情请参见编译 Exchange。本示例中使用 Exchange 3.0.0。
- 已经安装 Spark。
- 了解 Nebula Graph 中创建 Schema 的信息，包括 Tag 和 Edge type 的名称、属性等。
- 已经安装并开启 Hadoop 服务，并已启动 Hive Metastore 数据库（本示例中为 MySQL）。

操作步骤

步骤 1：在 NEBULA GRAPH 中创建 SCHEMA

分析数据，按以下步骤在 Nebula Graph 中创建 Schema：

1. 确认 Schema 要素。Nebula Graph 中的 Schema 要素如下表所示。

要素	名称	属性
Tag	`player`	`name string, age int`
Tag	`team`	`name string`
Edge Type	`follow`	`degree int`
Edge Type	`serve`	`start_year int, end_year int`

2. 在 Nebula Graph 中创建一个图空间 **basketballplayer**，并创建一个 Schema，如下所示。

```
## 创建图空间
nebula> CREATE SPACE basketballplayer \
        (partition_num = 10, \
        replica_factor = 1, \
        vid_type = FIXED_STRING(30));

## 选择图空间 basketballplayer
nebula> USE basketballplayer;

## 创建 Tag player
nebula> CREATE TAG player(name string, age int);

## 创建 Tag team
nebula> CREATE TAG team(name string);

## 创建 Edge type follow
nebula> CREATE EDGE follow(degree int);

## 创建 Edge type serve
nebula> CREATE EDGE serve(start_year int, end_year int);
```

更多信息，请参见快速开始。

步骤 2：使用 SPARK SQL 确认 HIVE SQL 语句

启动 spark-shell 环境后，依次运行以下语句，确认 Spark 能读取 Hive 中的数据。

```
scala> sql("select playerid, age, name from basketball.player").show
scala> sql("select teamid, name from basketball.team").show
scala> sql("select src_player, dst_player, degree from basketball.follow").show
scala> sql("select playerid, teamid, start_year, end_year from basketball.serve").show
```

以下为表 `basketball.player` 中读出的结果。

```
+---------+---+----------------+
| playerid| age|            name|
```


2022 vesoft Inc.

```
+---------+----+----------------+
|player100|  42|       Tim Duncan|
|player101|  36|      Tony Parker|
|player102|  33|LaMarcus Aldridge|
|player103|  32|         Rudy Gay|
|player104|  32|  Marco Belinelli|
+---------+----+----------------+
...
```

步骤 **3**：修改配置文件

编译 Exchange 后，复制 `target/classes/application.conf` 文件设置 Hive 数据源相关的配置。在本示例中，复制的文件名为 `hive_application.conf`。各个配置项的详细说明请参见配置说明。

```
{
  # Spark 相关配置
  spark: {
    app: {
      name: Nebula Exchange 3.0.0
    }
    driver: {
      cores: 1
      maxResultSize: 1G
    }
    cores: {
      max: 16
    }
  }

  # 如果 Spark 和 Hive 部署在不同集群，才需要配置连接 Hive 的参数，否则请忽略这些配置。
  #hive: {
  #  waredir: "hdfs://NAMENODE_IP:9000/apps/svr/hive-xxx/warehouse/"
  #  connectionURL: "jdbc:mysql://your_ip:3306/hive_spark?characterEncoding=UTF-8"
  #  connectionDriverName: "com.mysql.jdbc.Driver"
  #  connectionUserName: "user"
  #  connectionPassword: "password"
  #}

  # Nebula Graph 相关配置
  nebula: {
    address:{
      # 以下为 Nebula Graph 的 Graph 服务和所有 Meta 服务所在机器的 IP 地址及端口。
      # 如果有多个地址，格式为 "ip1:port","ip2:port","ip3:port"。
      # 不同地址之间以英文逗号 (,) 隔开。
      graph:["127.0.0.1:9669"]
      meta:["127.0.0.1:9559"]
    }
    # 填写的账号必须拥有 Nebula Graph 相应图空间的写数据权限。
    user: root
    pswd: nebula
    # 填写 Nebula Graph 中需要写入数据的图空间名称。
    space: basketballplayer
    connection: {
      timeout: 3000
      retry: 3
    }
    execution: {
      retry: 3
    }
    error: {
      max: 32
      output: /tmp/errors
    }
    rate: {
      limit: 1024
      timeout: 1000
    }
  }
  # 处理点
  tags: [
    # 设置 Tag player 相关信息。
    {
      # Nebula Graph 中对应的 Tag 名称。
      name: player
      type: {
        # 指定数据源文件格式，设置为 hive。
        source: hive
        # 指定如何将点数据导入 Nebula Graph：Client 或 SST。
        sink: client
      }

      # 设置读取数据库 basketball 中 player 表数据的 SQL 语句
      exec: "select playerid, age, name from basketball.player"

      # 在 fields 里指定 player 表中的列名称，其对应的 value 会作为 Nebula Graph 中指定属性。
      # fields 和 nebula.fields 里的配置必须一一对应。
      # 如果需要指定多个列名称，用英文逗号 (,) 隔开。
      fields: [age,name]
      nebula.fields: [age,name]

      # 指定表中某一列数据为 Nebula Graph 中点 VID 的来源。
```


2022 vesoft Inc.

```
    vertex:{
      field:playerid
    }

    # 单批次写入 Nebula Graph 的最大数据条数。
    batch: 256

    # Spark 分区数量
    partition: 32
  }
  # 设置 Tag team 相关信息。
  {
    name: team
    type: {
      source: hive
      sink: client
    }
    exec: "select teamid, name from basketball.team"
    fields: [name]
    nebula.fields: [name]
    vertex: {
      field: teamid
    }
    batch: 256
    partition: 32
  }

]

# 处理边数据
edges: [
  # 设置 Edge type follow 相关信息
  {
    # Nebula Graph 中对应的 Edge type 名称。
    name: follow

    type: {
      # 指定数据源文件格式，设置为 hive。
      source: hive

      # 指定边数据导入 Nebula Graph 的方式，
      # 指定如何将点数据导入 Nebula Graph：Client 或 SST。
      sink: client
    }

    # 设置读取数据库 basketball 中 follow 表数据的 SQL 语句。
    exec: "select src_player, dst_player, degree from basketball.follow"

    # 在 fields 里指定 follow 表中的列名称，其对应的 value 会作为 Nebula Graph 中指定属性。
    # fields 和 nebula.fields 里的配置必须一一对应。
    # 如果需要指定多个列名称，用英文逗号（,）隔开。
    fields: [degree]
    nebula.fields: [degree]

    # 在 source 里，将 follow 表中某一列作为边的起始点数据源。
    # 在 target 里，将 follow 表中某一列作为边的目的点数据源。
    source: {
      field: src_player
    }

    target: {
      field: dst_player
    }

    # 指定一个列作为 rank 的源（可选）。
    #ranking: rank

    # 单批次写入 Nebula Graph 的最大数据条数。
    batch: 256

    # Spark 分区数量
    partition: 32
  }

  # 设置 Edge type serve 相关信息
  {
    name: serve
    type: {
      source: hive
      sink: client
    }
    exec: "select playerid, teamid, start_year, end_year from basketball.serve"
    fields: [start_year,end_year]
    nebula.fields: [start_year,end_year]
    source: {
      field: playerid
    }
    target: {
      field: teamid
    }

    # 指定一个列作为 rank 的源（可选）。
    #ranking: rank
```


2022 vesoft Inc.

```
      batch: 256
      partition: 32
    }
  ]
}
```

步骤 4：向 NEBULA GRAPH 导入数据

运行如下命令将 Hive 数据导入到 Nebula Graph 中。关于参数的说明，请参见导入命令参数。

```
${SPARK_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange <nebula-exchange-3.0.0.jar_path> -c <hive_application.conf_path> -h
```

> **Note**
>
> JAR 包有两种获取方式：自行编译或者从 maven 仓库下载。

示例：

```
${SPARK_HOME}/bin/spark-submit  --master "local" --class com.vesoft.nebula.exchange.Exchange  /root/nebula-exchange/nebula-exchange/target/nebula-exchange-3.0.0.jar  -c /root/nebula-exchange/nebula-exchange/target/classes/hive_application.conf -h
```

用户可以在返回信息中搜索 `batchSuccess.<tag_name/edge_name>`，确认成功的数量。例如 `batchSuccess.follow: 300` 。

步骤 5：（可选）验证数据

用户可以在 Nebula Graph 客户端（例如 Nebula Studio）中执行查询语句，确认数据是否已导入。例如：

```
GO FROM "player100" OVER follow;
```

用户也可以使用命令 `SHOW STATS` 查看统计数据。

步骤 6：（如有）在 NEBULA GRAPH 中重建索引

导入数据后，用户可以在 Nebula Graph 中重新创建并重建索引。详情请参见索引介绍。

最后更新: June 20, 2023

2022 vesoft Inc.

19.4.10 导入 MaxCompute 数据

本文以一个示例说明如何使用 Exchange 将存储在 MaxCompute 上的数据导入 Nebula Graph。

数据集

本文以 basketballplayer 数据集为例。

环境配置

本文示例在 MacOS 下完成，以下是相关的环境配置信息：

- 硬件规格：
- CPU：1.7 GHz Quad-Core Intel Core i7
- 内存：16 GB
- Spark：2.4.7，单机版
- Hadoop：2.9.2，伪分布式部署
- MaxCompute：阿里云官方版本
- Nebula Graph：3.1.0。使用 Docker Compose 部署。

前提条件

开始导入数据之前，用户需要确认以下信息：

- 已经安装部署 Nebula Graph 并获取如下信息：
- Graph 服务和 Meta 服务的的 IP 地址和端口。
- 拥有 Nebula Graph 写权限的用户名和密码。
- 已经编译 Exchange。详情请参见编译 Exchange。本示例中使用 Exchange 3.0.0。
- 已经安装 Spark。
- 了解 Nebula Graph 中创建 Schema 的信息，包括 Tag 和 Edge type 的名称、属性等。
- 已经安装并开启 Hadoop 服务。

2022 vesoft Inc.

操作步骤

步骤 1：在 NEBULA GRAPH 中创建 SCHEMA

分析数据，按以下步骤在 Nebula Graph 中创建 Schema：

1. 确认 Schema 要素。Nebula Graph 中的 Schema 要素如下表所示。

要素	名称	属性
Tag	`player`	`name string, age int`
Tag	`team`	`name string`
Edge Type	`follow`	`degree int`
Edge Type	`serve`	`start_year int, end_year int`

2. 在 Nebula Graph 中创建一个图空间 **basketballplayer**，并创建一个 Schema，如下所示。

```
## 创建图空间
nebula> CREATE SPACE basketballplayer \
        (partition_num = 10, \
        replica_factor = 1, \
        vid_type = FIXED_STRING(30));

## 选择图空间 basketballplayer
nebula> USE basketballplayer;

## 创建 Tag player
nebula> CREATE TAG player(name string, age int);

## 创建 Tag team
nebula> CREATE TAG team(name string);

## 创建 Edge type follow
nebula> CREATE EDGE follow(degree int);

## 创建 Edge type serve
nebula> CREATE EDGE serve(start_year int, end_year int);
```

更多信息，请参见快速开始。

步骤 2：修改配置文件

编译 Exchange 后，复制 `target/classes/application.conf` 文件设置 MaxCompute 数据源相关的配置。在本示例中，复制的文件名为 `maxcompute_application.conf`。各个配置项的详细说明请参见配置说明。

```
{
  # Spark 相关配置
  spark: {
    app: {
      name: Nebula Exchange 3.0.0
    }
    driver: {
      cores: 1
      maxResultSize: 1G
    }
    cores: {
      max: 16
    }
  }

# Nebula Graph 相关配置
  nebula: {
    address:{
      # 以下为 Nebula Graph 的 Graph 服务和 Meta 服务所在机器的 IP 地址及端口。
      # 如果有多个地址，格式为 "ip1:port","ip2:port","ip3:port"。
      # 不同地址之间以英文逗号 (,) 隔开。
      graph:["127.0.0.1:9669"]
      meta:["127.0.0.1:9559"]
    }
    # 填写的账号必须拥有 Nebula Graph 相应图空间的写数据权限。
    user: root
    pswd: nebula
    # 填写 Nebula Graph 中需要写入数据的图空间名称。
    space: basketballplayer
    connection: {
      timeout: 3000
      retry: 3
    }
    execution: {
      retry: 3
```


2022 vesoft Inc.

```
    }
    error: {
      max: 32
      output: /tmp/errors
    }
    rate: {
      limit: 1024
      timeout: 1000
    }
  }
  # 处理点
  tags: [
    # 设置 Tag player 相关信息
    {
      name: player
      type: {
        # 指定数据源文件格式，设置为 MaxCompute。
        source: maxcompute
        # 指定如何将点数据导入 Nebula Graph：Client 或 SST。
        sink: client
      }

      # MaxCompute 的表名
      table:player

      # MaxCompute 的项目名
      project:project

      # MaxCompute 服务的 odpsUrl 和 tunnelUrl，
      # 地址可在 https://help.aliyun.com/document_detail/34951.html 查看。
      odpsUrl:"http://service.cn-hangzhou.maxcompute.aliyun.com/api"
      tunnelUrl:"http://dt.cn-hangzhou.maxcompute.aliyun.com"

      # MaxCompute 服务的 accessKeyId 和 accessKeySecret。
      accessKeyId:xxx
      accessKeySecret:xxx

      # MaxCompute 表的分区描述，该配置可选。
      partitionSpec:"dt='partition1'"

      # MaxCompute 的 Spark 连接器在读取 MaxCompute 数据时使用的分区数。默认为1，该配置可选。
      numPartitions:100

      # 请确保 SQL 语句中的表名和上方 table 的值相同，该配置可选。
      sentence:"select id, name, age, playerid from player where id < 10"

      # 在 fields 里指定 player 表中的列名称，其对应的 value 会作为 Nebula Graph 中指定属性。
      # fields 和 nebula.fields 里的配置必须一一对应。
      # 如果需要指定多个列名称，用英文逗号（,）隔开。
      fields:[name, age]
      nebula.fields:[name, age]

      # 指定表中某一列数据为 Nebula Graph 中点 VID 的来源。
      vertex:{
        field: playerid
      }

      # 单批次写入 Nebula Graph 的数据条数。
      batch: 256

      # Spark 分区数量
      partition: 32
    }

    # 设置 Tag team 相关信息。
    {
      name: team
      type: {
        source: maxcompute
        sink: client
      }
      table:team
      project:project
      odpsUrl:"http://service.cn-hangzhou.maxcompute.aliyun.com/api"
      tunnelUrl:"http://dt.cn-hangzhou.maxcompute.aliyun.com"
      accessKeyId:xxx
      accessKeySecret:xxx
      partitionSpec:"dt='partition1'"
      sentence:"select id, name, teamid from team where id < 10"
      fields:[name]
      nebula.fields:[name]
      vertex:{
        field: teamid
      }
      batch: 256
      partition: 32
    }
  ]

  # 处理边数据
  edges: [
    # 设置 Edge type follow 相关信息
    {
      # Nebula Graph 中对应的 Edge type 名称。
```


2022 vesoft Inc.

```
      name: follow

      type:{
        # 指定数据源文件格式，设置为 MaxCompute。
        source:maxcompute

        # 指定边数据导入 Nebula Graph 的方式，
        # 指定如何将点数据导入 Nebula Graph：Client 或 SST。
        sink:client
      }

      # MaxCompute 的表名
      table:follow

      # MaxCompute 的项目名
      project:project

      # MaxCompute 服务的 odpsUrl 和 tunnelUrl，
      # 地址可在 https://help.aliyun.com/document_detail/34951.html 查看。
      odpsUrl:"http://service.cn-hangzhou.maxcompute.aliyun.com/api"
      tunnelUrl:"http://dt.cn-hangzhou.maxcompute.aliyun.com"

      # MaxCompute 服务的 accessKeyId 和 accessKeySecret。
      accessKeyId:xxx
      accessKeySecret:xxx

      # MaxCompute 表的分区描述，该配置可选。
      partitionSpec:"dt='partition1'"

      # 请确保 SQL 语句中的表名和上方 table 的值相同，该配置可选。
      sentence:"select * from follow"

      # 在 fields 里指定 follow 表中的列名称，其对应的 value 会作为 Nebula Graph 中指定属性。
      # fields 和 nebula.fields 里的配置必须一一对应。
      # 如果需要指定多个列名称，用英文逗号（,）隔开。
      fields:[degree]
      nebula.fields:[degree]

      # 在 source 里，将 follow 表中某一列作为边的起始点数据源。
      source:{
        field: src_player
      }

      # 在 target 里，将 follow 表中某一列作为边的目的点数据源。
      target:{
        field: dst_player
      }

      # 指定一个列作为 rank 的源（可选）。
      #ranking: rank

      # Spark 分区数量
      partition:10

      # 单批次写入 Nebula Graph 的数据条数。
      batch:10
    }

    # 设置 Edge type serve 相关信息
    {
      name: serve
      type:{
        source:maxcompute
        sink:client
      }
      table:serve
      project:project
      odpsUrl:"http://service.cn-hangzhou.maxcompute.aliyun.com/api"
      tunnelUrl:"http://dt.cn-hangzhou.maxcompute.aliyun.com"
      accessKeyId:xxx
      accessKeySecret:xxx
      partitionSpec:"dt='partition1'"
      sentence:"select * from serve"
      fields:[start_year,end_year]
      nebula.fields:[start_year,end_year]
      source:{
        field: playerid
      }
      target:{
        field: teamid
      }

      # 指定一个列作为 rank 的源（可选）。
      #ranking: rank

      partition:10
      batch:10
    }
  ]
}
```


2022 vesoft Inc.

步骤 3：向 NEBULA GRAPH 导入数据

运行如下命令将 MaxCompute 数据导入到 Nebula Graph 中。关于参数的说明，请参见导入命令参数。

```
${SPARK_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange <nebula-exchange-3.0.0.jar_path> -c <maxcompute_application.conf_path>
```

> **Note**
>
> JAR 包有两种获取方式：自行编译或者从 maven 仓库下载。

示例：

```
${SPARK_HOME}/bin/spark-submit  --master "local" --class com.vesoft.nebula.exchange.Exchange  /root/nebula-exchange/nebula-exchange/target/nebula-exchange-3.0.0.jar  -c /root/nebula-exchange/nebula-exchange/target/classes/maxcompute_application.conf
```

用户可以在返回信息中搜索 `batchSuccess.<tag_name/edge_name>`，确认成功的数量。例如 `batchSuccess.follow: 300` 。

步骤 4：（可选）验证数据

用户可以在 Nebula Graph 客户端（例如 Nebula Studio）中执行查询语句，确认数据是否已导入。例如：

```
GO FROM "player100" OVER follow;
```

用户也可以使用命令 `SHOW STATS` 查看统计数据。

步骤 5：（如有）在 NEBULA GRAPH 中重建索引

导入数据后，用户可以在 Nebula Graph 中重新创建并重建索引。详情请参见索引介绍。

最后更新: June 20, 2023

2022 vesoft Inc.

19.4.11 导入 Pulsar 数据

本文简单说明如何使用 Exchange 将存储在 Pulsar 上的数据导入 Nebula Graph。

环境配置

本文示例在 MacOS 下完成，以下是相关的环境配置信息：

- 硬件规格：
- CPU：1.7 GHz Quad-Core Intel Core i7
- 内存：16 GB
- Spark：2.4.7，单机版
- Nebula Graph：3.1.0。使用 Docker Compose 部署。

前提条件

开始导入数据之前，用户需要确认以下信息：

- 已经安装部署 Nebula Graph 并获取如下信息：
- Graph 服务和 Meta 服务的的 IP 地址和端口。
- 拥有 Nebula Graph 写权限的用户名和密码。
- 已经编译 Exchange。详情请参见编译 Exchange。本示例中使用 Exchange 3.0.0。
- 已经安装 Spark。
- 了解 Nebula Graph 中创建 Schema 的信息，包括 Tag 和 Edge type 的名称、属性等。
- 已经安装并开启 Pulsar 服务。

2022 vesoft Inc.

操作步骤

步骤 1：在 NEBULA GRAPH 中创建 SCHEMA

分析数据，按以下步骤在 Nebula Graph 中创建 Schema：

1. 确认 Schema 要素。Nebula Graph 中的 Schema 要素如下表所示。

要素	名称	属性
Tag	`player`	`name string, age int`
Tag	`team`	`name string`
Edge Type	`follow`	`degree int`
Edge Type	`serve`	`start_year int, end_year int`

2. 在 Nebula Graph 中创建一个图空间 **basketballplayer**，并创建一个 Schema，如下所示。

```
## 创建图空间
nebula> CREATE SPACE basketballplayer \
        (partition_num = 10, \
        replica_factor = 1, \
        vid_type = FIXED_STRING(30));

## 选择图空间 basketballplayer
nebula> USE basketballplayer;

## 创建 Tag player
nebula> CREATE TAG player(name string, age int);

## 创建 Tag team
nebula> CREATE TAG team(name string);

## 创建 Edge type follow
nebula> CREATE EDGE follow(degree int);

## 创建 Edge type serve
nebula> CREATE EDGE serve(start_year int, end_year int);
```

更多信息，请参见快速开始。

步骤 2：修改配置文件

编译 Exchange 后，复制 `target/classes/application.conf` 文件设置 Pulsar 数据源相关的配置。在本示例中，复制的文件名为 `pulsar_application.conf`。各个配置项的详细说明请参见配置说明。

```
{
  # Spark 相关配置
  spark: {
    app: {
      name: Nebula Exchange 3.0.0
    }
    driver: {
      cores: 1
      maxResultSize: 1G
    }
    cores: {
      max: 16
    }
  }

  # Nebula Graph 相关配置
  nebula: {
    address:{
      # 以下为 Nebula Graph 的 Graph 服务和 Meta 服务所在机器的 IP 地址及端口。
      # 如果有多个地址，格式为 "ip1:port","ip2:port","ip3:port"。
      # 不同地址之间以英文逗号 (,) 隔开。
      graph:["127.0.0.1:9669"]
      meta:["127.0.0.1:9559"]
    }
    # 填写的账号必须拥有 Nebula Graph 相应图空间的写数据权限。
    user: root
    pswd: nebula
    # 填写 Nebula Graph 中需要写入数据的图空间名称。
    space: basketballplayer
    connection: {
      timeout: 3000
      retry: 3
    }
    execution: {
      retry: 3
```


2022 vesoft Inc.

```
    }
    error: {
      max: 32
      output: /tmp/errors
    }
    rate: {
      limit: 1024
      timeout: 1000
    }
  }
  # 处理点
  tags: [
    # 设置 Tag player 相关信息。
    {
      # Nebula Graph 中对应的 Tag 名称。
      name: player
      type: {
        # 指定数据源文件格式，设置为 Pulsar。
        source: pulsar
        # 指定如何将点数据导入 Nebula Graph：Client 或 SST。
        sink: client
      }
      # Pulsar 服务器地址。
      service: "pulsar://127.0.0.1:6650"
      # 连接 pulsar 的 admin.url。
      admin: "http://127.0.0.1:8081"
      # Pulsar 的选项，可以从 topic、topics 和 topicsPattern 选择一个进行配置。
      options: {
        topics: "topic1,topic2"
      }

      # 在 fields 里指定 player 表中的列名称，其对应的 value 会作为 Nebula Graph 中指定属性。
      # fields 和 nebula.fields 里的配置必须一一对应。
      # 如果需要指定多个列名称，用英文逗号（,）隔开。
      fields: [age,name]
      nebula.fields: [age,name]

      # 指定表中某一列数据为 Nebula Graph 中点 VID 的来源。
      vertex:{
          field:playerid
      }

      # 单批次写入 Nebula Graph 的数据条数。
      batch: 10

      # Spark 分区数量
      partition: 10
      # 读取消息的间隔。单位：秒。
      interval.seconds: 10
    }
    # 设置 Tag team 相关信息。
    {
      name: team
      type: {
        source: pulsar
        sink: client
      }
      service: "pulsar://127.0.0.1:6650"
      admin: "http://127.0.0.1:8081"
      options: {
        topics: "topic1,topic2"
      }
      fields: [name]
      nebula.fields: [name]
      vertex:{
          field:teamid
      }
      batch: 10
      partition: 10
      interval.seconds: 10
    }

  ]

  # 处理边数据
  edges: [
    # 设置 Edge type follow 相关信息
    {
      # Nebula Graph 中对应的 Edge type 名称。
      name: follow

      type: {
        # 指定数据源文件格式，设置为 Pulsar。
        source: pulsar

        # 指定边数据导入 Nebula Graph 的方式，
        # 指定如何将点数据导入 Nebula Graph：Client 或 SST。
        sink: client
      }

      # Pulsar 服务器地址。
      service: "pulsar://127.0.0.1:6650"
      # 连接 pulsar 的 admin.url。
      admin: "http://127.0.0.1:8081"
```


2022 vesoft Inc.

```
      # Pulsar 的选项，可以从 topic、topics 和 topicsPattern 选择一个进行配置。
      options: {
        topics: "topic1,topic2"
      }

      # 在 fields 里指定 follow 表中的列名称，其对应的 value 会作为 Nebula Graph 中指定属性。
      # fields 和 nebula.fields 里的配置必须一一对应。
      # 如果需要指定多个列名称，用英文逗号（,）隔开。
      fields: [degree]
      nebula.fields: [degree]

      # 在 source 里，将 follow 表中某一列作为边的起始点数据源。
      # 在 target 里，将 follow 表中某一列作为边的目的点数据源。
      source:{
          field:src_player
      }

      target:{
          field:dst_player
      }

      # 指定一个列作为 rank 的源（可选）。
      #ranking: rank

      # 单批次写入 Nebula Graph 的数据条数。
      batch: 10

      # Spark 分区数量
      partition: 10

      # 读取消息的间隔。单位：秒。
      interval.seconds: 10
    }

    # 设置 Edge type serve 相关信息
    {
      name: serve
      type: {
        source: Pulsar
        sink: client
      }
      service: "pulsar://127.0.0.1:6650"
      admin: "http://127.0.0.1:8081"
      options: {
        topics: "topic1,topic2"
      }

      fields: [start_year,end_year]
      nebula.fields: [start_year,end_year]
      source:{
          field:playerid
      }

      target:{
          field:teamid
      }

      # 指定一个列作为 rank 的源（可选）。
      #ranking: rank

      batch: 10
      partition: 10
      interval.seconds: 10
    }
  ]
}
```

步骤 3：向 NEBULA GRAPH 导入数据

运行如下命令将 Pulsar 数据导入到 Nebula Graph 中。关于参数的说明，请参见导入命令参数。

```
${SPARK_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange <nebula-exchange-3.0.0.jar_path> -c <pulsar_application.conf_path>
```

> **Note**
>
> JAR 包有两种获取方式：自行编译或者从 maven 仓库下载。

示例：

```
${SPARK_HOME}/bin/spark-submit  --master "local" --class com.vesoft.nebula.exchange.Exchange  /root/nebula-exchange/nebula-exchange/target/nebula-exchange-3.0.0.jar  -c /root/nebula-exchange/nebula-exchange/target/classes/pulsar_application.conf
```

用户可以在返回信息中搜索 `batchSuccess.<tag_name/edge_name>`，确认成功的数量。例如 `batchSuccess.follow: 300` 。

2022 vesoft Inc.

步骤 **4**: （可选）验证数据

用户可以在 Nebula Graph 客户端（例如 Nebula Studio）中执行查询语句，确认数据是否已导入。例如：

```
GO FROM "player100" OVER follow;
```

用户也可以使用命令 `SHOW STATS` 查看统计数据。

步骤 **5**: （如有）在 **NEBULA GRAPH** 中重建索引

导入数据后，用户可以在 Nebula Graph 中重新创建并重建索引。详情请参见索引介绍。

最后更新: June 20, 2023

2022 vesoft Inc.

19.4.12 导入 Kafka 数据

本文简单说明如何使用 Exchange 将存储在 Kafka 上的数据导入 Nebula Graph。

环境配置

本文示例在 MacOS 下完成，以下是相关的环境配置信息：

- 硬件规格：
- CPU：1.7 GHz Quad-Core Intel Core i7
- 内存：16 GB
- Spark：2.4.7，单机版
- Nebula Graph：3.1.0。使用 Docker Compose 部署。

前提条件

开始导入数据之前，用户需要确认以下信息：

- 已经安装部署 Nebula Graph 并获取如下信息：
- Graph 服务和 Meta 服务的的 IP 地址和端口。
- 拥有 Nebula Graph 写权限的用户名和密码。
- 已经编译 Exchange。详情请参见编译 Exchange。本示例中使用 Exchange 3.0.0。
- 已经安装 Spark。
- 了解 Nebula Graph 中创建 Schema 的信息，包括 Tag 和 Edge type 的名称、属性等。
- 已经安装并开启 Kafka 服务。

2022 vesoft Inc.

操作步骤

步骤 1：在 NEBULA GRAPH 中创建 SCHEMA

分析数据，按以下步骤在 Nebula Graph 中创建 Schema：

1. 确认 Schema 要素。Nebula Graph 中的 Schema 要素如下表所示。

要素	名称	属性
Tag	`player`	`name string, age int`
Tag	`team`	`name string`
Edge Type	`follow`	`degree int`
Edge Type	`serve`	`start_year int, end_year int`

2. 在 Nebula Graph 中创建一个图空间 **basketballplayer**，并创建一个 Schema，如下所示。

```
## 创建图空间
nebula> CREATE SPACE basketballplayer \
        (partition_num = 10, \
        replica_factor = 1, \
        vid_type = FIXED_STRING(30));

## 选择图空间 basketballplayer
nebula> USE basketballplayer;

## 创建 Tag player
nebula> CREATE TAG player(name string, age int);

## 创建 Tag team
nebula> CREATE TAG team(name string);

## 创建 Edge type follow
nebula> CREATE EDGE follow(degree int);

## 创建 Edge type serve
nebula> CREATE EDGE serve(start_year int, end_year int);
```

更多信息，请参见快速开始。

步骤 2：修改配置文件

编译 Exchange 后，复制 `target/classes/application.conf` 文件设置 Kafka 数据源相关的配置。在本示例中，复制的文件名为 `kafka_application.conf`。各个配置项的详细说明请参见配置说明。

```
{
  # Spark 相关配置
  spark: {
    app: {
      name: Nebula Exchange 3.0.0
    }
    driver: {
      cores: 1
      maxResultSize: 1G
    }
    cores: {
      max: 16
    }
  }

  # Nebula Graph 相关配置
  nebula: {
    address:{
      # 以下为 Nebula Graph 的 Graph 服务和 Meta 服务所在机器的 IP 地址及端口。
      # 如果有多个地址，格式为 "ip1:port","ip2:port","ip3:port"。
      # 不同地址之间以英文逗号 (,) 隔开。
      graph:["127.0.0.1:9669"]
      meta:["127.0.0.1:9559"]
    }
    # 填写的账号必须拥有 Nebula Graph 相应图空间的写数据权限。
    user: root
    pswd: nebula
    # 填写 Nebula Graph 中需要写入数据的图空间名称。
    space: basketballplayer
    connection: {
      timeout: 3000
      retry: 3
    }
    execution: {
      retry: 3
```


2022 vesoft Inc.

```
    }
    error: {
      max: 32
      output: /tmp/errors
    }
    rate: {
      limit: 1024
      timeout: 1000
    }
  }
  # 处理点
  tags: [
    # 设置 Tag player 相关信息。
    {
      # Nebula Graph 中对应的 Tag 名称。
      name: player
      type: {
        # 指定数据源文件格式，设置为 Kafka。
        source: kafka
        # 指定如何将点数据导入 Nebula Graph：Client 或 SST。
        sink: client
      }
      # Kafka 服务器地址。
      service: "127.0.0.1:9092"
      # 消息类别。
      topic: "topic_name1"

      # 在 fields 里指定 Kafka value 中的字段名称，多个字段用英文逗号（,）隔开。Spark Structured Streaming 读取 Kafka 数据后会将其以 JSON 格式存储于 value 字段中，而这里的 fields 要配置
JSON 的 key 名。示例如下：
      fields: [personName, personAge]
      # 设置与 fields 中的 key 对应的 Nebula Graph 属性名，key 的 value 将保存为相应的属性值。下方设置会将 personName 的 value 保存到 Nebula Graph 中的 name 属性，personAge 的 value 则保存
到 age 属性。
      nebula.fields: [name, age]

      # 指定表中某一列数据为 Nebula Graph 中点 VID 的来源。
      # 这里的值 key 和上面的 key 重复，表示 key 既作为 VID，也作为属性 name。
      vertex:{
          field:personId
      }

      # 单批次写入 Nebula Graph 的数据条数。
      batch: 10

      # Spark 分区数量
      partition: 10
      # 读取消息的间隔。单位：秒。
      interval.seconds: 10
    }
    # 设置 Tag team 相关信息。
    {
      name: team
      type: {
        source: kafka
        sink: client
      }
      service: "127.0.0.1:9092"
      topic: "topic_name2"
      fields: [key]
      nebula.fields: [name]
      vertex:{
          field:teamId
      }
      batch: 10
      partition: 10
      interval.seconds: 10
    }

  ]

  # 处理边数据
  edges: [
    # 设置 Edge type follow 相关信息
    {
      # Nebula Graph 中对应的 Edge type 名称。
      name: follow

      type: {
        # 指定数据源文件格式，设置为 Kafka。
        source: kafka

        # 指定边数据导入 Nebula Graph 的方式，
        # 指定如何将点数据导入 Nebula Graph：Client 或 SST。
        sink: client
      }

      # Kafka 服务器地址。
      service: "127.0.0.1:9092"
      # 消息类别。
      topic: "topic_name3"

      # 在 fields 里指定 Kafka value 中的字段名称，多个字段用英文逗号（,）隔开。Spark Structured Streaming 读取 Kafka 数据后会将其以 JSON 格式存储于 value 字段中，而这里的 fields 要配置
JSON 的 key 名。示例如下：
      fields: [degree]
      # 设置与 fields 中的 key 对应的 Nebula Graph 属性名，key 的 value 将保存为相应的属性值。下方设置会将 degree 的 value 保存到 Nebula Graph 中的 degree 属性。
```


2022 vesoft Inc.

```
      nebula.fields: [degree]

      # 在 source 里，将 topic 中某一列作为边的起始点数据源。
      # 在 target 里，将 topic 中某一列作为边的目的点数据源。
      source:{
          field:srcPersonId
      }

      target:{
          field:dstPersonId
      }

      # 指定一个列作为 rank 的源（可选）。
      #ranking: rank

      # 单批次写入 Nebula Graph 的数据条数。
      batch: 10

      # Spark 分区数量
      partition: 10

      # 读取消息的间隔。单位：秒。
      interval.seconds: 10
    }

    # 设置 Edge type serve 相关信息
    {
      name: serve
      type: {
        source: kafka
        sink: client
      }
      service: "127.0.0.1:9092"
      topic: "topic_name4"

      fields: [startYear,endYear]
      nebula.fields: [start_year,end_year]
      source:{
          field:personId
      }

      target:{
          field:teamId
      }

      # 指定一个列作为 rank 的源（可选）。
      #ranking: rank

      batch: 10
      partition: 10
      interval.seconds: 10
    }
  ]
}
```

步骤 **3**：向 **NEBULA GRAPH** 导入数据

运行如下命令将 Kafka 数据导入到 Nebula Graph 中。关于参数的说明，请参见导入命令参数。

```
${SPARK_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange <nebula-exchange-3.0.0.jar_path> -c <kafka_application.conf_path>
```

> **Note**
>
> JAR 包有两种获取方式：自行编译或者从 maven 仓库下载。

示例：

```
${SPARK_HOME}/bin/spark-submit  --master "local" --class com.vesoft.nebula.exchange.Exchange  /root/nebula-exchange/nebula-exchange/target/nebula-exchange-3.0.0.jar  -c /root/nebula-exchange/nebula-exchange/target/classes/kafka_application.conf
```

用户可以在返回信息中搜索 `batchSuccess.<tag_name/edge_name>`，确认成功的数量。例如 `batchSuccess.follow: 300` 。

步骤 **4**：（可选）验证数据

用户可以在 Nebula Graph 客户端（例如 Nebula Studio）中执行查询语句，确认数据是否已导入。例如：

```
GO FROM "player100" OVER follow;
```

用户也可以使用命令 `SHOW STATS` 查看统计数据。

2022 vesoft Inc.

步骤 **5**: （如有）在 **NEBULA GRAPH** 中重建索引

导入数据后，用户可以在 Nebula Graph 中重新创建并重建索引。详情请参见索引介绍。

最后更新: June 20, 2023

2022 vesoft Inc.

19.4.13 导入 SST 文件数据

本文以一个示例说明如何将数据源的数据生成 SST（Sorted String Table）文件并保存在 HDFS 上，然后导入 Nebula Graph，示例数据源是 CSV 文件。

注意事项

- 仅 Linux 系统支持导入 SST 文件。
- 不支持属性的 Default 值。
- 企业版 Exchange 3.0.0不支持基于 GEOGRAPHY 类型的数据生成 SST 文件。

背景信息

Exchange 支持两种数据导入模式:

- 直接将数据源的数据通过 **nGQL** 语句的形式导入 Nebula Graph。
- 将数据源的数据生成 SST 文件，然后借助 Console 将 SST 文件导入 Nebula Graph。

下文将介绍生成 SST 文件并用其导入数据的适用场景、实现方法、前提条件、操作步骤等内容。

适用场景

- 适合在线业务，因为生成时几乎不会影响业务（只是读取 Schema），导入速度快。

  Caution

  导入期间（大约 10 秒）会阻塞对应空间的写操作，并且之后数小时内可能有历史数据整理，建议在业务低峰期进行导入。

- 适合数据源数据量较大的场景，导入速度快。

实现方法

Nebula Graph 底层使用 RocksDB 作为键值型存储引擎。RocksDB 是基于硬盘的存储引擎，提供了一系列 API 用于创建及导入 SST 格式的文件，有助于快速导入海量数据。

SST 文件是一个内部包含了任意长度的有序键值对集合的文件，用于高效地存储大量键值型数据。生成 SST 文件的整个过程主要由 Exchange 的 Reader、sstProcessor 和 sstWriter 完成。整个数据处理过程如下:

1. Reader 从数据源中读取数据。
2. sstProcessor 根据 Nebula Graph 的 Schema 信息生成 SST 文件，然后上传至 HDFS。SST 文件的格式请参见数据存储格式。
3. sstWriter 打开一个文件并插入数据。生成 SST 文件时，Key 必须按照顺序写入。
4. 生成 SST 文件之后，RocksDB 通过 `IngestExternalFile()` 方法将 SST 文件导入到 Nebula Graph 中。例如:

```
IngestExternalFileOptions ifo;
# 导入两个 SST 文件
Status s = db_->IngestExternalFile({"/home/usr/file1.sst", "/home/usr/file2.sst"}, ifo);
if (!s.ok()) {
  printf("Error while adding file %s and %s, Error %s\n",
         file_path1.c_str(), file_path2.c_str(), s.ToString().c_str());
  return 1;
}
```

调用 `IngestExternalFile()` 方法时，RocksDB 默认会将文件拷贝到数据目录，并且阻塞 RocksDB 写入操作。如果 SST 文件中的键范围覆盖了 Memtable 键的范围，则将 Memtable 落盘（flush）到硬盘。将 SST 文件放置在 LSM 树最优位置后，为文件分配一个全局序列号，并打开写操作。

2022 vesoft Inc.

数据集

本文以 basketballplayer 数据集为例。

环境配置

本文示例在 MacOS 下完成，以下是相关的环境配置信息：

- 硬件规格：
- CPU：1.7 GHz Quad-Core Intel Core i7
- 内存：16 GB
- Spark：2.4.7 单机版
- Hadoop：2.9.2 伪分布式部署
- Nebula Graph：3.1.0。

前提条件

开始导入数据之前，用户需要确认以下信息：

- 已经安装部署 Nebula Graph 3.1.0 并获取如下信息：
- Graph 服务和 Meta 服务的的 IP 地址和端口。
- 拥有 Nebula Graph 写权限的用户名和密码。
- Meta 服务配置文件中的 `--ws_storage_http_port` 和 Storage 服务配置文件中的 `--ws_http_port` 一致。例如都为 `19779`。
- Graph 服务配置文件中的 `--ws_meta_http_port` 和 Meta 服务配置文件中的 `--ws_http_port` 一致。例如都为 `19559`。
- Schema 的信息，包括 Tag 和 Edge type 的名称、属性等。
- 已经编译 Exchange，或者直接下载编译完成的。jar 文件。本示例中使用 Exchange 3.0.0。
- 已经安装 Spark。
- 已经安装 JDK 1.8 或以上版本，并配置环境变量 JAVA_HOME。
- 确认 Hadoop 服务在所有部署 Storage 服务的机器上运行正常。

> **Note**
>
> - 如果需要生成其他数据源的 SST 文件，请参见相应数据源的文档，查看前提条件部分。
> - 如果只需要生成 SST 文件，不需要在部署 Storage 服务的机器上安装 Hadoop 服务。
> - 如需在 INGEST（数据导入）结束后自动移除 SST 文件，在 Storage 服务配置文件中增加 `--move_files=true`，该配置会让 Nebula Graph 在 INGEST 后将 SST 文件移动（`mv`）到 `data` 目录下。`--move_files` 的默认值为 `false`，此时 Nebula Graph 会复制（`cp`）SST 文件而不是移动。

2022 vesoft Inc.

操作步骤

步骤 1：在 NEBULA GRAPH 中创建 SCHEMA

分析 CSV 文件中的数据，按以下步骤在 Nebula Graph 中创建 Schema：

1. 确认 Schema 要素。Nebula Graph 中的 Schema 要素如下表所示。

要素	名称	属性
Tag	`player`	`name string, age int`
Tag	`team`	`name string`
Edge Type	`follow`	`degree int`
Edge Type	`serve`	`start_year int, end_year int`

2. 使用 Nebula Console 创建一个图空间 **basketballplayer**，并创建一个 Schema，如下所示。

```
## 创建图空间
nebula> CREATE SPACE basketballplayer \
        (partition_num = 10, \
        replica_factor = 1, \
        vid_type = FIXED_STRING(30));

## 选择图空间 basketballplayer
nebula> USE basketballplayer;

## 创建 Tag player
nebula> CREATE TAG player(name string, age int);

## 创建 Tag team
nebula> CREATE TAG team(name string);

## 创建 Edge type follow
nebula> CREATE EDGE follow(degree int);

## 创建 Edge type serve
nebula> CREATE EDGE serve(start_year int, end_year int);
```

更多信息，请参见快速开始。

步骤 2：处理 CSV 文件

确认以下信息：

1. 处理 CSV 文件以满足 Schema 的要求。

> **Note**
>
> 可以使用有表头或者无表头的 CSV 文件。

2. 获取 CSV 文件存储路径。

步骤 3：修改配置文件

编译 Exchange 后，复制 `target/classes/application.conf` 文件设置相关配置。在本示例中，复制的文件名为 `sst_application.conf` 。各个配置项的详细说明请参见配置说明。

```
{
  # Spark 相关配置
  spark: {
    app: {
      name: Nebula Exchange 3.0.0
    }

    master:local

    driver: {
      cores: 1
      maxResultSize: 1G
    }

    executor: {
```


2022 vesoft Inc.

```
      memory:1G
  }

  cores:{
    max: 16
  }
}

# Nebula Graph 相关配置
nebula: {
  address:{
    graph:["127.0.0.1:9669"]
    meta:["127.0.0.1:9559"]
  }
  user: root
  pswd: nebula
  space: basketballplayer

  # SST 文件相关配置
  path:{
      # 本地临时存放生成的 SST 文件的目录
      local:"/tmp"

      # SST 文件在 HDFS 的存储路径
      remote:"/sst"

      # HDFS 的 NameNode 地址
      hdfs.namenode: "hdfs://*.*.*.*:9000"
  }

  # 客户端连接参数
  connection: {
    # socket 连接、执行的超时时间，单位：毫秒。
    timeout: 30000
  }

  error: {
    # 最大失败数，超过后会退出应用程序。
    max: 32
    # 失败的导入作业将记录在输出路径中。
    output: /tmp/errors
  }

  # 使用Google Guava RateLimiter 来限制发送到 Nebula Graph 的请求。
  rate: {
    # RateLimiter 的稳定吞吐量。
    limit: 1024

    # 从 RateLimiter 获取允许的超时时间，单位：毫秒
    timeout: 1000
  }
}

# 处理点
tags: [
  # 设置 Tag player 相关信息。
  {
    # 指定 Nebula Graph 中定义的 Tag 名称。
    name: player
    type: {
      # 指定数据源，使用 CSV。
      source: csv

      # 指定如何将点数据导入 Nebula Graph：Client 或 SST。
      sink: sst
    }

    # 指定 CSV 文件的路径。
    # 文件存储在 HDFS 上，用双引号括起路径，以 hdfs://开头，例如"hdfs://ip:port/xx/xx.csv"。
    path: "hdfs://*.*.*.*:9000/dataset/vertex_player.csv"

    # 如果 CSV 文件没有表头，使用 [_c0, _c1, _c2, ..., _cn] 表示其表头，并将列指示为属性值的源。
    # 如果 CSV 文件有表头，则使用实际的列名。
    fields: [_c1, _c2]

    # 指定 Nebula Graph 中定义的属性名称。
    # fields 与 nebula.fields 的顺序必须一一对应。
    nebula.fields: [age, name]

    # 指定一个列作为 VID 的源。
    # vertex 的值必须与上述 fields 或者 csv.fields 中的列名保持一致。
    # 目前，Nebula Graph 3.1.0仅支持字符串或整数类型的 VID。
    vertex: {
      field:_c0
    }

    # 指定的分隔符。默认值为英文逗号（,）。
    separator: ","

    # 如果 CSV 文件有表头，请将 header 设置为 true。
    # 如果 CSV 文件没有表头，请将 header 设置为 false。默认值为 false。
    header: false

    # 指定单批次写入 Nebula Graph 的最大点数量。
```


2022 vesoft Inc.

```
    batch: 256

    # 指定 Spark 分片数量。
    partition: 32

    # 生成 SST 文件时是否要基于 Nebula Graph 中图空间的 partition 进行数据重分区。
    repartitionWithNebula: false
  }

  # 设置 Tag team 相关信息。
  {
    # 指定 Nebula Graph 中定义的 Tag 名称。
    name: team
    type: {
      # 指定数据源，使用 CSV。
      source: csv

      # 指定如何将点数据导入 Nebula Graph：Client 或 SST。
      sink: sst
    }

    # 指定 CSV 文件的路径。
    # 文件存储在 HDFS 上，用双引号括起路径，以 hdfs://开头，例如"hdfs://ip:port/xx/xx.csv"。
    path: "hdfs://*.*.*.*:9000/dataset/vertex_team.csv"

    # 如果 CSV 文件没有表头，使用 [_c0, _c1, _c2, ..., _cn] 表示其表头，并将列指示为属性值的源。
    # 如果 CSV 文件有表头，则使用实际的列名。
    fields: [_c1]

    # 指定 Nebula Graph 中定义的属性名称。
    # fields 与 nebula.fields 的顺序必须一一对应。
    nebula.fields: [name]

    # 指定一个列作为 VID 的源。
    # vertex 的值必须与上述 fields 或者 csv.fields 中的列名保持一致。
    # 目前，Nebula Graph 3.1.0仅支持字符串或整数类型的 VID。
    vertex: {
      field:_c0
    }

    # 指定的分隔符。默认值为英文逗号（,）。
    separator: ","

    # 如果 CSV 文件有表头，请将 header 设置为 true。
    # 如果 CSV 文件没有表头，请将 header 设置为 false。默认值为 false。
    header: false

    # 指定单批次写入 Nebula Graph 的最大点数量。
    batch: 256

    # 指定 Spark 分片数量。
    partition: 32

    # 生成 SST 文件时是否要基于 Nebula Graph 中图空间的 partition 进行数据重分区。
    repartitionWithNebula: false
  }

  # 如果需要添加更多点，请参考前面的配置进行添加。
]
# 处理边
edges: [
  # 设置 Edge type follow 相关信息。
  {
    # 指定 Nebula Graph 中定义的 Edge type 名称。
    name: follow
    type: {
      # 指定数据源，使用 CSV。
      source: csv

      # 指定如何将点数据导入 Nebula Graph：Client 或 SST。
      sink: sst
    }

    # 指定 CSV 文件的路径。
    # 文件存储在 HDFS 上，用双引号括起路径，以 hdfs://开头，例如"hdfs://ip:port/xx/xx.csv"。
    path: "hdfs://*.*.*.*:9000/dataset/edge_follow.csv"

    # 如果 CSV 文件没有表头，使用 [_c0, _c1, _c2, ..., _cn] 表示其表头，并将列指示为属性值的源。
    # 如果 CSV 文件有表头，则使用实际的列名。
    fields: [_c2]

    # 指定 Nebula Graph 中定义的属性名称。
    # fields 与 nebula.fields 的顺序必须一一对应。
    nebula.fields: [degree]

    # 指定一个列作为起始点和目的点的源。
    # vertex 的值必须与上述 fields 或者 csv.fields 中的列名保持一致。
    # 目前，Nebula Graph 3.1.0仅支持字符串或整数类型的 VID。
    source: {
      field: _c0
    }
    target: {
      field: _c1
    }
```


2022 vesoft Inc.

```
    # 指定的分隔符。默认值为英文逗号（,）。
    separator: ","

    # 指定一个列作为 rank 的源（可选）。

    #ranking: rank

    # 如果 CSV 文件有表头，请将 header 设置为 true。
    # 如果 CSV 文件没有表头，请将 header 设置为 false。默认值为 false。
    header: false

    # 指定单批次写入 Nebula Graph 的最大边数量。
    batch: 256

    # 指定 Spark 分片数量。
    partition: 32

    # 生成 SST 文件时是否要基于 Nebula Graph 中图空间的 partition 进行数据重分区。
    repartitionWithNebula: false
  }

  # 设置 Edge type serve 相关信息。
  {
    # 指定 Nebula Graph 中定义的 Edge type 名称。
    name: serve
    type: {
      # 指定数据源，使用 CSV。
      source: csv

      # 指定如何将点数据导入 Nebula Graph：Client 或 SST。
      sink: sst
    }

    # 指定 CSV 文件的路径。
    # 文件存储在 HDFS 上，用双引号括起路径，以 hdfs://开头，例如"hdfs://ip:port/xx/xx.csv"。
    path: "hdfs://*.*.*.*:9000/dataset/edge_serve.csv"

    # 如果 CSV 文件没有表头，使用 [_c0, _c1, _c2, ..., _cn] 表示其表头，并将列指示为属性值的源。
    # 如果 CSV 文件有表头，则使用实际的列名。
    fields: [_c2,_c3]

    # 指定 Nebula Graph 中定义的属性名称。
    # fields 与 nebula.fields 的顺序必须一一对应。
    nebula.fields: [start_year, end_year]

    # 指定一个列作为起始点和目的点的源。
    # vertex 的值必须与上述 fields 或者 csv.fields 中的列名保持一致。
    # 目前，Nebula Graph 3.1.0仅支持字符串或整数类型的 VID。
    source: {
      field: _c0
    }
    target: {
      field: _c1
    }

    # 指定的分隔符。默认值为英文逗号（,）。
    separator: ","

    # 指定一个列作为 rank 的源（可选）。
    #ranking: _c5

    # 如果 CSV 文件有表头，请将 header 设置为 true。
    # 如果 CSV 文件没有表头，请将 header 设置为 false。默认值为 false。
    header: false

    # 指定单批次写入 Nebula Graph 的最大边数量。
    batch: 256

    # 指定 Spark 分片数量。
    partition: 32

    # 生成 SST 文件时是否要基于 Nebula Graph 中图空间的 partition 进行数据重分区。
    repartitionWithNebula: false
  }

 ]
 # 如果需要添加更多边，请参考前面的配置进行添加。
}
```

步骤 **4**：生成 **SST** 文件

运行如下命令将 CSV 源文件生成为 SST 文件。关于参数的说明，请参见命令参数。

```
${SPARK_HOME}/bin/spark-submit --master "local" --conf spark.sql.shuffle.partitions=<shuffle_concurrency> --class com.vesoft.nebula.exchange.Exchange <nebula-exchange-3.0.0.jar_path> -c
<sst_application.conf_path>
```


2022 vesoft Inc.

Note

生成 SST 文件时，会涉及到 Spark 的 shuffle 操作，请注意在提交命令中增加 `spark.sql.shuffle.partitions` 的配置。

Note

JAR 包有两种获取方式：自行编译或者从 maven 仓库下载。

示例：

```
${SPARK_HOME}/bin/spark-submit  --master "local" --conf spark.sql.shuffle.partitions=200 --class com.vesoft.nebula.exchange.Exchange  /root/nebula-exchange/nebula-exchange/target/nebula-exchange-3.0.0.jar  -c /root/nebula-exchange/nebula-exchange/target/classes/sst_application.conf
```

任务执行完成后，可以在 HDFS 上的 `/sst` 目录（`nebula.path.remote` 参数指定）内查看到生成的 SST 文件。

Note

如果对 Schema 有修改操作，例如重建图空间、修改 Tag、修改 Edge type 等，需要重新生成 SST 文件，因为 SST 文件会验证 Space ID、Tag ID、Edge ID 等信息。

步骤 **5**：导入 **SST** 文件

Note

导入前请确认以下信息：

- 确认所有部署 Storage 服务的机器上都已部署 Hadoop 服务，并配置 HADOOP_HOME 和 JAVA_HOME。
- Meta 服务配置文件中的 `--ws_storage_http_port`（如果没有，请手动添加）和 Storage 服务配置文件中的 `--ws_http_port` 一致。例如都为 `19779`。
- Graph 服务配置文件中的 `--ws_meta_http_port`（如果没有，请手动添加）和 Meta 服务配置文件中的 `--ws_http_port` 一致。例如都为 `19559`。

使用客户端工具连接 Nebula Graph 数据库，按如下操作导入 SST 文件：

1. 执行命令选择之前创建的图空间。

```
nebula> USE basketballplayer;
```

2. 执行命令下载 SST 文件：

```
nebula> SUBMIT JOB DOWNLOAD HDFS "hdfs://<hadoop_address>:<hadoop_port>/<sst_file_path>";
```

示例：

```
nebula> SUBMIT JOB DOWNLOAD HDFS "hdfs://*.*.*.*:9000/sst";
```

3. 执行命令导入 SST 文件：

```
nebula> SUBMIT JOB INGEST;
```

Note

- 如果需要重新下载，请在 Nebula Graph 安装路径内的 `data/storage/nebula` 目录内，将对应 Space ID 目录内的 `download` 文件夹删除，然后重新下载 SST 文件。如果图空间是多副本，保存副本的所有机器都需要删除 `download` 文件夹。
- 如果导入时出现问题需要重新导入，重新执行 `SUBMIT JOB INGEST;` 即可。

2022 vesoft Inc.

步骤 6: （可选）验证数据

用户可以在 Nebula Graph 客户端（例如 Nebula Studio）中执行查询语句，确认数据是否已导入。例如：

```
GO FROM "player100" OVER follow;
```

用户也可以使用命令 `SHOW STATS` 查看统计数据。

步骤 7: （如有）在 NEBULA GRAPH 中重建索引

导入数据后，用户可以在 Nebula Graph 中重新创建并重建索引。详情请参见索引介绍。

最后更新: June 20, 2023

2022 vesoft Inc.

19.4.14 导出 Nebula Graph 数据

本文以一个示例说明如何使用 Exchange 将 Nebula Graph 中的数据导出到 CSV 文件中。

Enterpriseonly

仅企业版 Exchange 支持导出 Nebula Graph 数据，且仅能导出到 CSV 文件。

环境准备

本示例在 Linux 系统的虚拟机环境下完成，导出数据前准备的软硬件信息如下。

硬件

类型	信息
CPU	4 Intel(R) Xeon(R) Platinum 8260 CPU @ 2.30GHz
内存	16G
硬盘	50G

系统

CentOS 7.9.2009

软件

名称	版本
JDK	1.8.0
Hadoop	2.10.1
Scala	2.12.11
Spark	2.4.7
Nebula Graph	3.1.0

数据集

在本示例中，作为数据源的 Nebula Graph 存储着 basketballplayer 数据集，其中的 Schema 要素如下表所示。

要素	名称	属性
Tag	`player`	`name string, age int`
Tag	`team`	`name string`
Edge type	`follow`	`degree int`
Edge type	`serve`	`start_year int, end_year int`

操作步骤

1. 从 Nebula Graph 企业版套餐 中获取企业版 Exchange 的 JAR 文件。
2. 修改配置文件。

   企业版 Exchange 提供了导出 Nebula Graph 数据专用的配置文件模板 `export_application.conf`，其中各配置项的说明参见 Exchange 配置。本示例使用的配置文件核心内容如下：

2022 vesoft Inc.

```
...

  # Processing tags
  # There are tag config examples for different dataSources.
  tags: [
    # export NebulaGraph tag data to csv, only support export to CSV for now.
    {
      name: player
      type: {
        source: Nebula
        sink: CSV
      }
      # the path to save the NebulaGrpah data, make sure the path doesn't exist.
      path:"hdfs://192.168.8.177:9000/vertex/player"
      # if no need to export any properties when export NebulaGraph tag data
      # if noField is configured true, just export vertexId
      noField:false
      # define properties to export from NebulaGraph tag data
      # if return.fields is configured as empty list, then export all properties
      return.fields:[]
      # nebula space partition number
      partition:10
    }

...

  ]

  # Processing edges
  # There are edge config examples for different dataSources.
  edges: [
    # export NebulaGraph tag data to csv, only support export to CSV for now.
    {
      name: follow
      type: {
        source: Nebula
        sink: CSV
      }
      # the path to save the NebulaGrpah data, make sure the path doesn't exist.
      path:"hdfs://192.168.8.177:9000/edge/follow"
      # if no need to export any properties when export NebulaGraph edge data
      # if noField is configured true, just export src,dst,rank
      noField:false
      # define properties to export from NebulaGraph edge data
      # if return.fields is configured as empty list, then export all properties
      return.fields:[]
      # nebula space partition number
      partition:10
    }

...

  ]
}
```

3. 使用如下命令导出 Nebula Graph 中的数据。

```
<spark_install_path>/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange nebula-exchange-x.y.z.jar_path> -c <export_application.conf_path>
```

本示例使用的导出命令如下。

```
$ ./spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange \
  ~/exchange-ent/nebula-exchange-ent-3.0.0.jar -c ~/exchange-ent/export_application.conf
```

4. 检查导出的数据。

a. 查看目标路径下是否成功生成了 CSV 文件。

```
$ hadoop fs -ls /vertex/player
Found 11 items
-rw-r--r--   3 nebula supergroup          0 2021-11-05 07:36 /vertex/player/_SUCCESS
-rw-r--r--   3 nebula supergroup        160 2021-11-05 07:36 /vertex/player/    part-00000-17293020-ba2e-4243-b834-34495c0536b3-c000.csv
-rw-r--r--   3 nebula supergroup        163 2021-11-05 07:36 /vertex/player/    part-00001-17293020-ba2e-4243-b834-34495c0536b3-c000.csv
-rw-r--r--   3 nebula supergroup        172 2021-11-05 07:36 /vertex/player/    part-00002-17293020-ba2e-4243-b834-34495c0536b3-c000.csv
-rw-r--r--   3 nebula supergroup        172 2021-11-05 07:36 /vertex/player/    part-00003-17293020-ba2e-4243-b834-34495c0536b3-c000.csv
-rw-r--r--   3 nebula supergroup        144 2021-11-05 07:36 /vertex/player/    part-00004-17293020-ba2e-4243-b834-34495c0536b3-c000.csv
-rw-r--r--   3 nebula supergroup        173 2021-11-05 07:36 /vertex/player/    part-00005-17293020-ba2e-4243-b834-34495c0536b3-c000.csv
-rw-r--r--   3 nebula supergroup        160 2021-11-05 07:36 /vertex/player/    part-00006-17293020-ba2e-4243-b834-34495c0536b3-c000.csv
-rw-r--r--   3 nebula supergroup        148 2021-11-05 07:36 /vertex/player/    part-00007-17293020-ba2e-4243-b834-34495c0536b3-c000.csv
-rw-r--r--   3 nebula supergroup        125 2021-11-05 07:36 /vertex/player/    part-00008-17293020-ba2e-4243-b834-34495c0536b3-c000.csv
-rw-r--r--   3 nebula supergroup        119 2021-11-05 07:36 /vertex/player/    part-00009-17293020-ba2e-4243-b834-34495c0536b3-c000.csv
```

b. 检查 CSV 文件内容，确定数据导出成功。

2022 vesoft Inc.

最后更新: June 20, 2023

2022 vesoft Inc.

19.5 Exchange 常见问题

19.5.1 编译问题

Q：部分非 **central** 仓库的包下载失败，报错 `Could not resolve dependencies for project xxx`

请检查 Maven 安装目录下 `libexec/conf/settings.xml` 文件的 `mirror` 部分：

```
<mirror>
    <id>alimaven</id>
    <mirrorOf>central</mirrorOf>
    <name>aliyun maven</name>
    <url>http://maven.aliyun.com/nexus/content/repositories/central/</url>
</mirror>
```

检查 `mirrorOf` 的值是否配置为 `*`，如果为 `*`，请修改为 `central` 或 `*,!SparkPackagesRepo,!bintray-streamnative-maven`。

原因：Exchange 的 `pom.xml` 中有两个依赖包不在 Maven 的 central 仓库中，`pom.xml` 配置了这两个依赖所在的仓库地址。如果 maven 中配置的镜像地址对应的 `mirrorOf` 值为 `*`，那么所有依赖都会在 central 仓库下载，导致下载失败。

Q：编译 **Exchange** 时无法下载 **SNAPSHOT** 包

现象：编译时提示 `Could not find artifact com.vesoft:client:jar:xxx-SNAPSHOT`。

原因：本地 maven 没有配置用于下载 SNAPSHOT 的仓库。maven 中默认的 central 仓库用于存放正式发布版本，而不是开发版本（SNAPSHOT）。

解决方案：在 maven 的 setting.xml文件的 `profiles` 作用域内中增加以下配置：

```
  <profile>
     <activation>
        <activeByDefault>true</activeByDefault>
     </activation>
     <repositories>
        <repository>
            <id>snapshots</id>
            <url>https://oss.sonatype.org/content/repositories/snapshots/</url>
            <snapshots>
               <enabled>true</enabled>
            </snapshots>
      </repository>
     </repositories>
  </profile>
```

19.5.2 执行问题

Q：**Yarn-Cluster** 模式下如何提交?

在 Yarn-Cluster 模式下提交任务，请参考如下命令：

```
$SPARK_HOME/bin/spark-submit --class com.vesoft.nebula.exchange.Exchange \
--master yarn-cluster \
--files application.conf \
--conf spark.driver.extraClassPath=./ \
--conf spark.executor.extraClassPath=./ \
nebula-exchange-3.0.0.jar \
-c application.conf
```

Q：报错 `method name xxx not found`

一般是端口配置错误，需检查 Meta 服务、Graph 服务、Storage 服务的端口配置。

Q：报 **NoSuchMethod**、**MethodNotFound** 错误（`Exception in thread "main" java.lang.NoSuchMethodError` 等）

绝大多数是因为 JAR 包冲突和版本冲突导致的报错，请检查报错服务的版本，与 Exchange 中使用的版本进行对比，检查是否一致，尤其是 Spark 版本、Scala 版本、Hive 版本。

2022 vesoft Inc.

Q：**Exchange** 导入 **Hive** 数据时报错 `Exception in thread "main" org.apache.spark.sql.AnalysisException: Table or view not found`

检查提交 exchange 任务的命令中是否遗漏参数 `-h`，检查 table 和 database 是否正确，在 spark-sql 中执行用户配置的 exec 语句，验证 exec 语句的正确性。

Q：运行时报错 `com.facebook.thrift.protocol.TProtocolException: Expected protocol id xxx`

请检查 Nebula Graph 服务端口配置是否正确。

- 如果是源码、RPM 或 DEB 安装，请配置各个服务的配置文件中 `--port` 对应的端口号。
- 如果是 docker 安装，请配置 docker 映射出来的端口号，查看方式如下：

  在 `nebula-docker-compose` 目录下执行 `docker-compose ps`，例如：

```
$ docker-compose ps
              Name                            Command                State                                                          Ports
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
nebula-docker-compose_graphd_1      /usr/local/nebula/bin/nebu ...   Up (healthy)   0.0.0.0:33205->19669/tcp, 0.0.0.0:33204->19670/tcp, 0.0.0.0:9669->9669/tcp
nebula-docker-compose_metad0_1      ./bin/nebula-metad --flagf ...   Up (healthy)   0.0.0.0:33165->19559/tcp, 0.0.0.0:33162->19560/tcp, 0.0.0.0:33167->9559/tcp, 9560/tcp
nebula-docker-compose_metad1_1      ./bin/nebula-metad --flagf ...   Up (healthy)   0.0.0.0:33166->19559/tcp, 0.0.0.0:33163->19560/tcp, 0.0.0.0:33168->9559/tcp, 9560/tcp
nebula-docker-compose_metad2_1      ./bin/nebula-metad --flagf ...   Up (healthy)   0.0.0.0:33161->19559/tcp, 0.0.0.0:33160->19560/tcp, 0.0.0.0:33164->9559/tcp, 9560/tcp
nebula-docker-compose_storaged0_1   ./bin/nebula-storaged --fl ...   Up (healthy)   0.0.0.0:33180->19779/tcp, 0.0.0.0:33178->19780/tcp, 9777/tcp, 9778/tcp, 0.0.0.0:33183->9779/tcp, 9780/
tcp
nebula-docker-compose_storaged1_1   ./bin/nebula-storaged --fl ...   Up (healthy)   0.0.0.0:33175->19779/tcp, 0.0.0.0:33172->19780/tcp, 9777/tcp, 9778/tcp, 0.0.0.0:33177->9779/tcp, 9780/
tcp
nebula-docker-compose_storaged2_1   ./bin/nebula-storaged --fl ...   Up (healthy)   0.0.0.0:33184->19779/tcp, 0.0.0.0:33181->19780/tcp, 9777/tcp, 9778/tcp, 0.0.0.0:33185->9779/tcp, 9780/
tcp
```

  查看 `Ports` 列，查找 docker 映射的端口号，例如：

  - Graph 服务可用的端口号是 9669。

  - Meta 服务可用的端口号有 33167、33168、33164。

  - Storage 服务可用的端口号有 33183、33177、33185。

Q：运行时报错 `Exception in thread "main" com.facebook.thrift.protocol.TProtocolException: The field 'code' has been assigned the invalid value -4`

检查 Exchange 版本与 Nebula Graph 版本是否匹配，详细信息可参考使用限制。

Q：将 **Hive** 中的数据导入 **Nebula Graph** 时出现乱码如何解决?

如果 Hive 中数据的属性值包含中文字符，可能出现该情况。解决方案是在导入命令中的 JAR 包路径前加上以下选项：

```
--conf spark.driver.extraJavaOptions=-Dfile.encoding=utf-8
--conf spark.executor.extraJavaOptions=-Dfile.encoding=utf-8
```

即：

```
<spark_install_path>/bin/spark-submit --master "local" \
--conf spark.driver.extraJavaOptions=-Dfile.encoding=utf-8 \
--conf spark.executor.extraJavaOptions=-Dfile.encoding=utf-8 \
--class com.vesoft.nebula.exchange.Exchange \
<nebula-exchange-3.x.y.jar_path> -c <application.conf_path>
```

如果是在 YARN 中，则用以下命令：

```
<spark_install_path>/bin/spark-submit \
--class com.vesoft.nebula.exchange.Exchange \
--master yarn-cluster \
--files <application.conf_path> \
--conf spark.driver.extraClassPath=./ \
--conf spark.executor.extraClassPath=./ \
--conf spark.driver.extraJavaOptions=-Dfile.encoding=utf-8 \
--conf spark.executor.extraJavaOptions=-Dfile.encoding=utf-8 \
<nebula-exchange-3.x.y.jar_path> \
-c application.conf
```


2022 vesoft Inc.

Q：**Hive** 数据导入时提示 **schema** 版本不一致

Spark 日志提示 `Hive Schema version 1.2.0 does not match metastore's schema version 2.1.0 Metastore is not upgraded or corrupt` 的原因是 Hive 环境中配置的 metastore schema 版本和 Spark 使用的 metastore 版本不一致。

解决方法：

1. 将 Hive 环境中存储 Hive metastore 信息的 MySQL version 信息更新为 Spark 中使用的 metastore 版本。

   假设 Hive 在 MySQL 中存储 metastore 的数据库是 `hive`，需要按如下方式修改 `hive.VERSION` 表中的 `version` 字段：

   ```
   update hive.VERSION set SCHEMA_VERSION="2.1.0" where VER_ID=1
   ```

2. 在 Hive 环境的 `hive-site.xml` 文件中增加如下配置：

   ```
   <property>
   <name>hive.metastore.schema.verification</name>
   <value>false</value>
   </property>
   ```

3. 重启 Hive。

Q: 生成 **SST** 时提示 **org.rocksdb.RocksDBException: While open a file for appending: /path/sst/1-xxx.sst: No such file or directory**

排查方法：

1. 检查 `/path` 是否存在，如没有或者路径设置错误，创建或修正路径。
2. 检查 Spark 在每台机器上的当前用户对 `/path` 是否有操作权限，如没有，添加权限。

19.5.3 配置问题

Q：哪些配置项影响导入性能?

- batch：每次发送给 Nebula Graph 服务的 nGQL 语句中包含的数据条数。
- partition：Spark 数据的分区数，表示数据导入的并发数。
- nebula.rate：向 Nebula Graph 发送请求前先去令牌桶获取令牌。
  - limit：表示令牌桶的大小。
  - timeout：表示获取令牌的超时时间。

根据机器性能可适当调整这四项参数的值。如果在导入过程中，Storage 服务的 leader 变更，可以适当调小这四项参数的值，降低导入速度。

19.5.4 其他问题

Q：**Exchange** 支持哪些版本的 **Nebula Graph**?

请参见 Exchange 的使用限制。

Q：**Exchange** 与 **Spark Writer** 有什么关系?

Exchange 是在 Spark Writer 基础上开发的 Spark 应用程序，二者均适用于在分布式环境中将集群的数据批量迁移到 Nebula Graph 中，但是后期的维护工作将集中在 Exchange 上。与 Spark Writer 相比，Exchange 有以下改进：

- 支持更丰富的数据源，如 MySQL、Neo4j、Hive、HBase、Kafka、Pulsar 等。
- 修复了 Spark Writer 的部分问题。例如 Spark 读取 HDFS 里的数据时，默认读取到的源数据均为 String 类型，可能与 Nebula Graph 定义的 Schema 不同，所以 Exchange 增加了数据类型的自动匹配和类型转换，当 Nebula Graph 定义的 Schema 中数据类型为非 String 类型（如 double）时，Exchange 会将 String 类型的源数据转换为对应的类型（如 double）。

2022 vesoft Inc.

Q：**Exchange** 传输数据的性能如何?

Exchange 的性能测试数据和测试方法参见 Nebula Exchange test result。

最后更新: June 20, 2023

2022 vesoft Inc.

20. Nebula Operator

20.1 什么是 Nebula Operator

20.1.1 基本概念

Nebula Operator 是用于在 Kubernetes 系统上自动化部署和运维 Nebula Graph 集群的工具。依托于 Kubernetes 扩展机制，Nebula Graph 将其运维领域的知识全面注入至 Kubernetes 系统中，让 Nebula Graph 成为真正的云原生图数据库。

20.1.2 工作原理

对于 Kubernetes 系统内不存在的资源类型，用户可以通过添加自定义 API 对象的方式注册，常见的方法是使用 CustomResourceDefinition（CRD） 。

Nebula Operator 将 Nebula Graph 集群的部署管理抽象为 CRD。通过结合多个内置的 API 对象，包括 StatefulSet、Service 和 ConfigMap，Nebula Graph 集群的日常管理和维护被编码为一个控制循环。在 Kubernetes 系统内，每一种内置资源对象，都运行着一个特定的控制循环，将它的实际状态通过事先规定好的编排动作，逐步调整为最终的期望状态。当一个 CR 实例被提交时，Nebula Operator 会根据控制流程驱动数据库集群进入最终状态。

20.1.3 功能介绍

Nebula Operator 已具备的功能如下：

- 集群创建和卸载：Nebula Operator 简化了用户部署和卸载集群的过程。用户只需提供对应的 CR 文件，Nebula Operator 即可快速创建或者删除一个对应的 Nebula Graph 集群。更多信息参考使用 Kubectl 部署 Nebula Graph 集群或者使用 Helm 部署 Nebula Graph 集群。
- 集群扩容和缩容：通过在控制循环中调用 Nebula Graph 原生提供的扩缩容接口，Nebula Graph 封装 Nebula Operator 实现了扩缩容的逻辑，用户可以通过 YAML 配置进行简单的扩缩容，且保证数据的稳定性。更多信息参考使用 Kubeclt 扩缩容集群 或使用 Helm 扩缩容集群。
- 集群升级：支持升级 3.0.0 版的 Nebula Graph 集群至 3.1.x 版。
- 故障自愈：Nebula Operator 调用 Nebula Graph 集群提供的接口，动态地感知服务状态。一旦发现异常，Nebula Operator 自动进行容错处理。更多信息参考故障自愈。
- 均衡调度：基于调度器扩展接口，Nebula Operator 提供的调度器可以将应用 Pods 均匀地分布在 Nebula Graph 集群中。

20.1.4 使用限制

版本限制

Nebula Operator 不支持 v1.x 版本的 Nebula Graph，其与 Nebula Graph 版本的对应关系如下：

Nebula Operator 版本	**Nebula Graph** 版本
1.1.0	3.0.0 ~ 3.1.x
1.0.0	3.0.0 ~ 3.1.x
0.9.0	2.5.x ~ 2.6.x
0.8.0	2.5.x

版本兼容性

- 1.x 版本的 Nebula Operator 不兼容 3.x 以下版本的 Nebula Graph.
- 由于 0.9.0 版本的 Nebula Operator 的日志盘和数据盘分开存储，因此用 0.9.0 版的 Nebula Operator 管理通过 0.8.0 版本创建的 2.5.x 版本的 Nebula Graph 集群会导致兼容性问题。用户可以备份 2.5.x 版本的 Nebula Graph 集群，然后使用 0.9.0 版本的 Operator 创建 2.6.x 版本集群。

2022 vesoft Inc.

功能限制

Nebula Operator 的扩缩容功能仅对企业版的 Nebula Graph 集群开放，不支持使用 Nebula Operator 扩缩容社区版的 Nebula Graph 集群。

20.1.5 更新说明

Release

最后更新: June 20, 2023

2022 vesoft Inc.

20.2 使用流程

使用 Nebula Operator 访问 Nebula Graph 集群服务的流程如下：

1. 安装 Nebula Operator。
2. 创建 Nebula Graph 集群。

   具体步骤参考使用 Kubectl 部署 Nebula Graph 集群或者使用 Helm 部署 Nebula Graph 集群。
3. 连接 Nebula Graph 服务。

最后更新: June 20, 2023

2022 vesoft Inc.

20.3 部署 Nebula Operator

用户可使用 Helm 工具部署 Nebula Operator。

20.3.1 背景信息

Nebula Operator 为用户管理 Nebula Graph 集群，使用户无需在生产环境中手动安装、扩展、升级和卸载 Nebula Graph，减轻用户管理不同应用版本的负担。

20.3.2 前提条件

安装软件

安装 Nebula Operator 前，用户需要安装以下软件并确保安装版本的正确性：

软件	版本要求
Kubernetes	>= 1.16
Helm	>= 3.2.0
CoreDNS	>= 1.6.0
CertManager	>= 1.2.0
OpenKruise	>= 0.8.0

如果使用基于角色的访问控制的策略，用户需开启 RBAC（可选）。

2022 vesoft Inc.

软件说明

> **Note**
>
> 以下内容为 Nebula Operator 使用的第三方项目，Nebula Operator 不负责处理安装这些软件过程中出现的问题。

- CoreDNS

  CoreDNS 是一个灵活的、可扩展的 DNS 服务器，被安装在集群内作为集群内 Pods 的 DNS 服务器。

  Nebula Graph 集群中的每个组件通过 DNS 解析类似 `x.default.svc.cluster.local` 这样的域名相互通信。

- cert-manager

  > **Note**
  >
  > 如果用户已将 Nebula Operator 配置项 `admissionWebhook.create` 的值设为 `false`，无需安装 cert-manager。有关配置项的详情，请参考下文安装 **Nebula Operator** 中的自定义配置 **Chart** 部分。

  cert-manager 是一个自动化管理证书的工具，利用 Kubernetes API 扩展功能，使用 Webhook 服务器提供对 cert-manager 资源的动态准入控制。用户可参考 cert-manager installation documentation 安装 cert-manager。

  cert-manager 用于验证 Nebula Graph 的每个组件副本。如果用户在生产环境中运行它并关心 Nebula Graph 的高可用性，建议将 `admissionWebhook.create` 的值设为 `true`，然后再安装 cert-manager。

- OpenKruise

  OpenKruise 是 Kubernetes 的一整套标准扩展，能与原始的 Kubernetes 一起工作，为应用 Pod、Sidecar 容器，甚至是节点中的镜像的管理供更强大和高效的功能。Nebula Operator 启动时需要用到 OpenKruise 开启针对 StatefulSet 的高级功能。用户可参考 openkruise installation documentation 安装 OpenKruise。

20.3.3 操作步骤

安装 **Nebula Operator**

1. 添加 Nebula Operator chart 仓库至 Helm。

```
helm repo add nebula-operator https://vesoft-inc.github.io/nebula-operator/charts
```

2. 拉取最新的 Helm 仓库。

```
helm repo update
```

参考 Helm 仓库获取更多 `helm repo` 相关信息。

3. 安装 Nebula Operator。

```
helm install nebula-operator nebula-operator/nebula-operator --namespace=<namespace_name> --version=${chart_version}
```

例如，安装1.1.0版的 Operator 命令如下。

```
helm install nebula-operator nebula-operator/nebula-operator --namespace=nebula-operator-system --version=1.1.0
```

- 上述命令中的 `nebula-operator-system` 为用户创建的命名空间。如果用户未创建该命名空间，可以执行 `kubectl create namespace nebula-operator-system` 进行创建。用户也可创建其他命名空间。
- `1.1.0` 为 Nebula Operator chart 的版本。当 Chart 中只有一个默认版本时，可不指定。执行 `helm search repo -l nebula-operator` 查看 Chart 版本。

用户可在执行安装 Nebula Operator chart 命令时自定义其配置。更多信息，查看下文自定义配置 **Chart**。

2022 vesoft Inc.

自定义配置 **Chart**

执行 `helm show values [CHART] [flags]` 查看可配置的选项。

示例如下：

```
[abby@master ~]$ helm show values nebula-operator/nebula-operator
image:
  nebulaOperator:
    image: vesoft/nebula-operator:v1.1.0
    imagePullPolicy: Always
  kubeRBACProxy:
    image: gcr.io/kubebuilder/kube-rbac-proxy:v0.8.0
    imagePullPolicy: Always
  kubeScheduler:
    image: k8s.gcr.io/kube-scheduler:v1.18.8
    imagePullPolicy: Always

imagePullSecrets: []
kubernetesClusterDomain: ""

controllerManager:
  create: true
  replicas: 2
  env: []
  resources:
    limits:
      cpu: 200m
      memory: 200Mi
    requests:
      cpu: 100m
      memory: 100Mi

admissionWebhook:
  create: true

scheduler:
  create: true
  schedulerName: nebula-scheduler
  replicas: 2
  env: []
  resources:
    limits:
      cpu: 200m
      memory: 200Mi
    requests:
      cpu: 100m
      memory: 100Mi
...
```

部分参数描述如下：

参数	默认值	描述
`image.nebulaOperator.image`	`vesoft/nebula-operator:v1.1.0`	Nebula Operator 的镜像，版本为1.1.0。
`image.nebulaOperator.imagePullPolicy`	`IfNotPresent`	镜像拉取策略。
`imagePullSecrets`	-	镜像拉取密钥。
`kubernetesClusterDomain`	`cluster.local`	集群域名。
`controllerManager.create`	`true`	是否启用 controller-manager。
`controllerManager.replicas`	`2`	controller-manager 副本数。
`admissionWebhook.create`	`true`	是否启用 Admission Webhook。
`shceduler.create`	`true`	是否启用 Scheduler。
`shceduler.schedulerName`	`nebula-scheduler`	调度器名称。
`shceduler.replicas`	`2`	nebula-scheduler 副本数。

执行 `helm install [NAME] [CHART] [flags]` 命令安装 Chart 时，可指定 Chart 配置。更多信息，参考安装前自定义 Chart。

以下示例为在安装 Nebula Operator 时，指定 Nebula Operator 的 AdmissionWebhook 机制为关闭状态（默认开启 AdmissionWebhook）：

2022 vesoft Inc.

```
helm install nebula-operator nebula-operator/nebula-operator --namespace=<nebula-operator-system> --set admissionWebhook.create=false
```

参考 Helm 安装获取更多 `helm install` 相关信息。

更新 **Nebula Operator**

1. 拉取最新的 Helm 仓库。

```
helm repo update
```

2. 通过 `--set` 或 `--values` 传递配置参数，更新 Nebula Operator。

- `--set`：通过命令行的方式对指定项进行覆盖。
- `--values` (或 -f)：使用 YAML 文件覆盖配置。

有关可以更新的配置项，查看上文自定义配置 **Chart**。

例如，更新 Nebula Operator 的 AdmissionWebhook 机制为关闭状态（默认开启 AdmissionWebhook）。

```
helm upgrade nebula-operator nebula-operator/nebula-operator --namespace=nebula-operator-system --version=1.1.0 --set admissionWebhook.create=false
```

更多信息，参考 Helm 升级。

2022 vesoft Inc.

升级 **Nebula Operator**

版本兼容性

- 不支持升级 0.9.0 及以下版本的 NebulaGraph Operator 至 1.x 版本。
- 1.x 版本的 Nebula Operator 不兼容 3.x 以下版本的 Nebula Graph。

1. 拉取最新的 Helm 仓库。

```
helm repo update
```

2. 升级 Nebula Operator 至 1.1.0 版本。

```
helm upgrade nebula-operator nebula-operator/nebula-operator --namespace=<namespace_name> --version=1.1.0
```

示例：

```
helm upgrade nebula-operator nebula-operator/nebula-operator --namespace=nebula-operator-system --version=1.1.0
```

输出：

```
Release "nebula-operator" has been upgraded. Happy Helming!
NAME: nebula-operator
LAST DEPLOYED: Tue Nov 16 02:21:08 2021
NAMESPACE: nebula-operator-system
STATUS: deployed
REVISION: 3
TEST SUITE: None
NOTES:
Nebula Operator installed!
```

3. 拉取最新的 CRD 配置文件。

Note

升级 Operator 后，需要同时升级相应的 CRD 配置，否则 Nebula Graph 集群创建会失败。有关 CRD 的配置，参见 apps.nebula-graph.io_nebulaclusters.yaml。

a. 下载 Nebula Operator chart 至本地。

```
helm pull nebula-operator/nebula-operator --version=1.1.0
```

- `--version`：升级版本号。如不指定，则默认为最新版本。

b. 执行 `tar -zxvf` 解压安装包。

例如：解压 1.1.0 chart 包至 `/tmp` 路径下。

```
tar -zxvf nebula-operator-1.1.0.tgz -C /tmp
```

- `-C /tmp`：如不指定，则默认解压至当前路径。

4. 在 `nebula-operator` 目录下升级 CRD 配置文件。

```
kubectl apply -f crds/nebulacluster.yaml
```

输出：

```
customresourcedefinition.apiextensions.k8s.io/nebulaclusters.apps.nebula-graph.io configured
```

卸载 **Nebula Operator**

1. 卸载 Nebula Operator chart。

2022 vesoft Inc.

```
helm uninstall nebula-operator --namespace=<nebula-operator-system>
```

2. 删除 CRD。

```
kubectl delete crd nebulaclusters.apps.nebula-graph.io
```

20.3.4 后续操作

使用 Nebula Operator 自动化部署 Nebula Graph 集群。更多信息，请参考使用 Kubectl 部署 Nebula Graph 集群或者使用 Helm 部署 Nebula Graph 集群。

最后更新: June 20, 2023

2022 vesoft Inc.

20.4 部署 Nebula Graph集群

20.4.1 使用 Kubectl 部署 Nebula Graph 集群

> 版本兼容性

1.x 版本的 Nebula Operator 不兼容 3.x 以下版本的 Nebula Graph。

前提条件

- 安装 Nebula Operator
- 准备相应的 License 文件

> **Enterpriseonly**

只有当创建的 Nebula Graph 集群是企业版本时，才需要提供 License 文件。

2022 vesoft Inc.

创建集群

本文以创建名为 `nebula` 的集群为例，说明如何部署 Nebula Graph 集群。

2022 vesoft Inc.

1. 创建名为 `apps_v1alpha1_nebulacluster.yaml` 的文件。

2022 vesoft Inc.

社区版集群示例

```
apiVersion: apps.nebula-graph.io/v1alpha1
kind: NebulaCluster
metadata:
  name: nebula
spec:
  graphd:
    resources:
      requests:
        cpu: "500m"
        memory: "500Mi"
      limits:
        cpu: "1"
        memory: "1Gi"
    replicas: 1
    image: vesoft/nebula-graphd
    version: v3.1.0
    service:
      type: NodePort
      externalTrafficPolicy: Local
    logVolumeClaim:
      resources:
        requests:
          storage: 2Gi
      storageClassName: gp2
  metad:
    resources:
      requests:
        cpu: "500m"
        memory: "500Mi"
      limits:
        cpu: "1"
        memory: "1Gi"
    replicas: 1
    image: vesoft/nebula-metad
    version: v3.1.0
    dataVolumeClaim:
      resources:
        requests:
          storage: 2Gi
      storageClassName: gp2
    logVolumeClaim:
      resources:
        requests:
          storage: 2Gi
      storageClassName: gp2
  storaged:
    resources:
      requests:
        cpu: "500m"
        memory: "500Mi"
      limits:
        cpu: "1"
        memory: "1Gi"
    replicas: 3
    image: vesoft/nebula-storaged
    version: v3.1.0
    dataVolumeClaim:
      resources:
        requests:
          storage: 2Gi
      storageClassName: gp2
    logVolumeClaim:
      resources:
        requests:
          storage: 2Gi
      storageClassName: gp2
  reference:
    name: statefulsets.apps
    version: v1
  schedulerName: default-scheduler
  imagePullPolicy: Always
```

企业版集群示例

```
# 联系销售人员获取完整配置。
apiVersion: apps.nebula-graph.io/v1alpha1
kind: NebulaCluster
metadata:
  annotations:
    nebula-graph.io/owner: test
  name: nebula
spec:
  enablePVReclaim: true
  graphd:
    readinessProbe:
      failureThreshold: 3
      httpGet:
        path: /status
        port: 19669
        scheme: HTTP
      initialDelaySeconds: 40
      periodSeconds: 10
      successThreshold: 1
      timeoutSeconds: 10
    image: reg.vesoft-inc.com/vesoft-ent/nebula-graphd
    logVolumeClaim:
      resources:
        requests:
          storage: 2Gi
      storageClassName: fast-disks
```


2022 vesoft Inc.

参数描述如下：

参数	默认值	描述
`metadata.name`	-	创建的 Nebula Graph 集群名称。
`spec.graphd.replicas`	`1`	Graphd 服务的副本数。
`spec.graphd.images`	`vesoft/nebula-graphd`	Graphd 服务的容器镜像。
`spec.graphd.version`	`v3.1.0`	Graphd 服务的版本号。
`spec.graphd.service`	-	Graphd 服务 Service 配置。
`spec.graphd.logVolumeClaim.storageClassName`	-	Graphd 服务的日志盘存储配置。
`spec.metad.replicas`	`1`	Metad 服务的副本数。
`spec.metad.images`	`vesoft/nebula-metad`	Metad 服务的容器镜像。
`spec.metad.version`	`v3.1.0`	Metad 服务的版本号。
`spec.metad.dataVolumeClaim.storageClassName`	-	Metad 服务的数据盘存储配置。
`spec.metad.logVolumeClaim.storageClassName`	-	Metad 服务的日志盘存储配置。
`spec.storaged.replicas`	`3`	Storaged 服务的副本数。
`spec.storaged.images`	`vesoft/nebula-storaged`	Storaged 服务的容器镜像。
`spec.storaged.version`	`v3.1.0`	Storaged 服务的版本号。
`spec.storaged.dataVolumeClaim.storageClassName`	-	Storaged 服务的数据盘存储配置。
`spec.storaged.logVolumeClaim.storageClassName`	-	Storaged 服务的日志盘存储配置。
`spec.reference.name`	-	依赖的控制器名称。
`spec.schedulerName`	-	调度器名称。
`spec.imagePullPolicy`	Nebula Graph 镜像的拉取策略。关于拉取策略详情，请参考 Image pull policy。	镜像拉取策略。
`spec.metad.license`	-	创建企业版 Nebula Graph 集群所需的 License 配置。

Enterpriseonly

拉取企业版 Nebula Graph 集群镜像前，请确保已联系销售人员（inqury@vesoft.com）获取企业版 Nebula Graph 集群的镜像信息。

2. 创建 Nebula Graph 集群。

```
kubectl create -f apps_v1alpha1_nebulacluster.yaml
```

返回：

```
nebulacluster.apps.nebula-graph.io/nebula created
```

3. 配置集群 License。

Enterpriseonly

- 此步骤只适用于创建企业版 Nebula Graph集群。
- 创建社区版 Nebula Graph 集群时，忽略此步骤。

```
kubectl create secret generic nebula-license --from-file=nebula.license
```


2022 vesoft Inc.

查看集群 License 信息：

```
kubectl get secrets nebula-license -o yaml
```

4. 查看 Nebula Graph 集群状态。

```
kubectl get nebulaclusters.apps.nebula-graph.io nebula
```

返回：

```
NAME     GRAPHD-DESIRED   GRAPHD-READY   METAD-DESIRED   METAD-READY   STORAGED-DESIRED   STORAGED-READY   AGE
nebula   1                1              1               1             3                  3                86s
```

扩缩容集群

Enterpriseonly

- 仅支持通过 v1.1.0 及以上版本的 Nebula Operator 扩缩容企业版的 Nebula Graph 集群。
- 不支持扩缩容社区版的 Nebula Graph 集群。

用户可以通过编辑 `apps_v1alpha1_nebulacluster.yaml` 文件中的 `replicas` 的值进行 Nebula Graph 集群的扩缩容。

2022 vesoft Inc.

扩容集群

本文举例扩容 Nebula Graph 集群中 Storage 服务至 5 个。步骤如下：

1. 将 `apps_v1alpha1_nebulacluster.yaml` 文件中 `storaged.replicas` 的参数值从 `3` 改为 `5`。

```
  storaged:
    resources:
      requests:
        cpu: "500m"
        memory: "500Mi"
      limits:
        cpu: "1"
        memory: "1Gi"
    replicas: 5
    image: vesoft/nebula-storaged
    version: v3.1.0
    dataVolumeClaim:
      resources:
        requests:
          storage: 2Gi
      storageClassName: gp2
    logVolumeClaim:
      resources:
        requests:
          storage: 2Gi
      storageClassName: gp2
  reference:
    name: statefulsets.apps
    version: v1
  schedulerName: default-scheduler
```

2. 执行以下命令使上述更新同步至 Nebula Graph 集群 CR 中。

```
kubectl apply -f apps_v1alpha1_nebulacluster.yaml
```

3. 查看 Storage 服务的副本数。

```
kubectl  get pods -l app.kubernetes.io/cluster=nebula
```

返回：

```
NAME                READY   STATUS    RESTARTS   AGE
nebula-graphd-0     1/1     Running   0          2m
nebula-metad-0      1/1     Running   0          2m
nebula-storaged-0   1/1     Running   0          2m
nebula-storaged-1   1/1     Running   0          2m
nebula-storaged-2   1/1     Running   0          2m
nebula-storaged-3   1/1     Running   0          5m
nebula-storaged-4   1/1     Running   0          5m
```

由上可看出 Storage 服务的副本数被扩容至 5 个。

缩容集群

缩容集群的原理和扩容一样，用户只需将 `apps_v1alpha1_nebulacluster.yaml` 文件中的 `replicas` 的值缩小。具体操作，请参考上文的扩容集群部分。

Caution

目前仅支持对 Nebula Graph 集群中的 Graph 服务和 Storage 服务进行扩缩容，不支持扩缩容 Meta 服务。

删除集群

使用 Kubectl 删除 Nebula Graph 集群的命令如下：

```
kubectl delete -f apps_v1alpha1_nebulacluster.yaml
```

后续操作

连接 Nebula Graph 数据库

2022 vesoft Inc.

最后更新: June 20, 2023

2022 vesoft Inc.

20.4.2 使用 Helm 部署 Nebula Graph 集群

> 版本兼容性

1.x 版本的 Nebula Operator 不兼容 3.x 以下版本的 Nebula Graph。

前提条件

安装 Nebula Operator

创建 **Nebula Graph** 集群

1. 添加 Nebula Operator chart 仓库至 Helm（如已创添加，略过前面 1 至 2 步，从第 3 步开始执行）。

```
helm repo add nebula-operator https://vesoft-inc.github.io/nebula-operator/charts
```

2. 更新 Helm 仓库，拉取最新仓库资源。

```
helm repo update
```

3. 配置 Helm 的环境变量。

```
export NEBULA_CLUSTER_NAME=nebula         # Nebula Graph 集群的名字。
export NEBULA_CLUSTER_NAMESPACE=nebula    # Nebula Graph 集群所处的命名空间的名字。
export STORAGE_CLASS_NAME=gp2             # Nebula Graph 集群的 StorageClass。
```

4. 为 Nebula Graph 集群创建命名空间（如已创建，略过此步）。

```
kubectl create namespace "${NEBULA_CLUSTER_NAMESPACE}"
```

5. 创建 Nebula Graph 集群。

```
helm install "${NEBULA_CLUSTER_NAME}" nebula-operator/nebula-cluster \
    --namespace="${NEBULA_CLUSTER_NAMESPACE}" \
    --set nameOverride=${NEBULA_CLUSTER_NAME} \
    --set nebula.storageClassName="${STORAGE_CLASS_NAME}"
```

6. 查看 Nebula Graph 集群创建状态。

```
kubectl -n "${NEBULA_CLUSTER_NAMESPACE}" get pod -l "app.kubernetes.io/cluster=${NEBULA_CLUSTER_NAME}"
```

返回示例：

```
NAME                READY   STATUS    RESTARTS   AGE
nebula-graphd-0     1/1     Running   0          5m34s
nebula-graphd-1     1/1     Running   0          5m34s
nebula-metad-0      1/1     Running   0          5m34s
nebula-metad-1      1/1     Running   0          5m34s
nebula-metad-2      1/1     Running   0          5m34s
nebula-storaged-0   1/1     Running   0          5m34s
nebula-storaged-1   1/1     Running   0          5m34s
nebula-storaged-2   1/1     Running   0          5m34s
```

扩缩容集群

> **terpriseonly**

- 仅支持通过 v1.1.0 及以上版本的 Nebula Operator 扩缩容企业版的 Nebula Graph 集群。
- 不支持扩缩容社区版的 Nebula Graph 集群。

用户可通过定义 Nebula Graph 中不同服务对应的 `replicas` 的值扩缩容 Nebuala Graph 集群。

2022 vesoft Inc.

例如，扩容 Nebula Graph 集群中 Storage 的副本数为 5（原始值为 2），命令如下：

```
helm upgrade "${NEBULA_CLUSTER_NAME}" nebula-operator/nebula-cluster \
    --namespace="${NEBULA_CLUSTER_NAMESPACE}" \
    --set nameOverride=${NEBULA_CLUSTER_NAME} \
    --set nebula.storageClassName="${STORAGE_CLASS_NAME}" \
    --set nebula.storaged.replicas=5
```

同理，将 Nebula Graph 集群中服务对应的 `replicas` 的值设置成小于原始值，即可实现集群服务的缩容。

Caution

目前仅支持对 Nebula Graph 集群中的 Graph 服务和 Storage 服务进行扩缩容，不支持扩缩容 Meta 服务。

用户可点击 nebula-cluster/values.yaml 查看 Nebula Cluster 集群 Chart 的更多配置。有关文件中配置项的解释，参考下文 **Nebula Graph 集群 Chart 配置参数说明**。

删除集群

使用 Helm 删除集群的命令如下：

```
helm uninstall "${NEBULA_CLUSTER_NAME}" --namespace="${NEBULA_CLUSTER_NAMESPACE}"
```

或者使用真实值删除集群，例如：

```
helm uninstall nebula --namespace=nebula
```

后续操作

连接 Nebula Graph 数据库

2022 vesoft Inc.

Nebula Graph 集群 **Chart** 配置参数说明

2022 vesoft Inc.

参数	默认值	描述
`nameOverride`	`nil`	覆盖集群 Chart 的名称。
`nebula.version`	`v3.1.0`	Nebula Graph 的版本。
`nebula.imagePullPolicy`	`IfNotPresent`	Nebula Graph 镜像的拉取策略。关于拉取策略详情，请参考 Image pull policy。
`nebula.storageClassName`	`nil`	持久存储卷的类型，默认使用 StorageClass 的名字。
`nebula.schedulerName`	`default-scheduler`	Nebula Graph 集群的调度器。
`nebula.reference`	`{"name": "statefulsets.apps", "version": "v1"}`	为 Nebula Graph 引用的工作负载。
`nebula.graphd.image`	`vesoft/nebula-graphd`	Graphd 容器镜像名称。使用 `nebula.version` 中的值作为版本。
`nebula.graphd.replicas`	`2`	Graphd 服务的副本数。
`nebula.graphd.env`	`[]`	Graphd 服务的环境变量。
`nebula.graphd.resources`	`{"resources":{"requests":{"cpu":"500m","memory":"500Mi"},"limits":{"cpu":"1","memory":"1Gi"}}}`	Graphd 资源配置。
`nebula.graphd.logStorage`	`500Mi`	Graphd 服务的日志盘大小。
`nebula.graphd.podLabels`	`{}`	Graphd 服务 Pod 的标签。
`nebula.graphd.podAnnotations`	`{}`	Graphd 服务 Pod 的注解。
`nebula.graphd.nodeSelector`	`{}`	为 Graphd pod 设置节点标签以分配至指定的节点中。
`nebula.graphd.tolerations`	`{}`	为 Graphd pod 设置容忍度。
`nebula.graphd.affinity`	`{}`	为 Graphd pod 设置亲和性。
`nebula.graphd.readinessProbe`	`{}`	为 Graphd pod 设置就绪探针以检测容器的状态。
`nebula.graphd.sidecarContainers`	`{}`	为 Graphd pod 设置 Sidecar Containers。
`nebula.graphd.sidecarVolumes`	`{}`	为 Graphd pod 设置 Sidecar Volumes。
`nebula.metad.image`	`vesoft/nebula-metad`	Metad 容器镜像名称。使用 `nebula.version` 中的值作为版本。
`nebula.metad.replicas`	`3`	Metad 服务的副本数。
`nebula.metad.env`	`[]`	Metad 服务的环境变量。
`nebula.metad.resources`	`{"resources":{"requests":{"cpu":"500m","memory":"500Mi"},"limits":{"cpu":"1","memory":"1Gi"}}}`	Metad 服务的资源配置。
`nebula.metad.logStorage`	`500Mi`	Metad 服务的日志盘大小。
`nebula.metad.dataStorage`	`1Gi`	Metad 服务的数据盘大小。
`nebula.metad.podLabels`	`{}`	Metad 服务 Pod 的标签。
`nebula.metad.podAnnotations`	`{}`	Metad 服务 Pod 的注解。

2022 vesoft Inc.

参数	默认值	描述
nebula.metad.nodeSelector	{}	为 Metad pod 设置节点标签以分配至指定的节点中。
nebula.metad.tolerations	{}	为 Metad pod 设置容忍度。
nebula.metad.affinity	{}	为 Metad pod 设置亲和性。
nebula.metad.readinessProbe	{}	为 Metad pod 设置就绪探针以检测容器的状态。
nebula.metad.sidecarContainers	{}	为 Metad pod 设置 Sidecar Containers。
nebula.metad.sidecarVolumes	{}	为 Metad pod 设置 Sidecar Volumes。
nebula.storaged.image	vesoft/nebula-storaged	Storaged 容器镜像名称。使用 nebula.version 中的值作为版本。
nebula.storaged.replicas	3	Storaged 服务的副本数。
nebula.storaged.env	[]	Storaged 服务的环境变量。
nebula.storaged.resources	{"resources":{"requests":{"cpu":"500m","memory":"500Mi"},"limits":{"cpu":"1","memory":"1Gi"}}}	Storaged 服务的资源配置。
nebula.storaged.logStorage	500Mi	Storaged 服务的日志盘大小。
nebula.storaged.dataStorage	1Gi	Storaged 服务的数据盘大小。
nebula.storaged.podLabels	{}	Storaged 服务 Pod 的标签。
nebula.storaged.podAnnotations	{}	Storaged 服务 Pod 的注解。
nebula.storaged.nodeSelector	{}	为 Storaged pod 设置节点标签以分配至指定的节点中。
nebula.storaged.tolerations	{}	为 Storaged pod 设置容忍度。
nebula.storaged.affinity	{}	为 Storaged pod 设置亲和性。
nebula.storaged.readinessProbe	{}	为 Storaged pod 设置就绪探针以检测容器的状态。
nebula.storaged.sidecarContainers	{}	为 Storaged pod 设置 Sidecar Containers。
nebula.storaged.sidecarVolumes	{}	为 Storaged Pod 设置 Sidecar Volumes。
imagePullSecrets	[]	拉取镜像的 Secret。

最后更新: June 20, 2023

2022 vesoft Inc.

20.5 配置 Nebula Graph 集群

20.5.1 自定义 Nebula Graph 集群的配置参数

Nebula Graph 集群中 Meta、Storage、Graph 服务都有各自的配置，其在用户创建的 CR 实例（Nebula Graph 集群）的 YAML 文件中被定义为 `config`。`config` 中的设置会被映射并加载到对应服务的 ConfigMap 中。

> **Note**
>
> 暂不支持通过 Helm 自定义 Nebula Graph 集群的配置参数。

`config` 结构如下：

```
Config map[string]string `json:"config,omitempty"`
```

前提条件

已使用 K8s 创建一个集群。具体步骤，参见使用 Kubectl 创建 Nebula Graph 集群。

操作步骤

以下示例使用名为 `nebula` 的集群、名为 `nebula_cluster.yaml` 的 YAML 配置文件，说明如何为集群的 Graph 服务配置 `config`：

1. 执行以下命令进入 `nebula` 集群的编辑页面。

```
kubectl edit nebulaclusters.apps.nebula-graph.io nebula
```

2. 在 YAML 文件的 `spec.graphd.config` 配置项中，添加 `enable_authorize` 和 `auth_type`。

```
apiVersion: apps.nebula-graph.io/v1alpha1
kind: NebulaCluster
metadata:
  name: nebula
  namespace: default
spec:
  graphd:
    resources:
      requests:
        cpu: "500m"
        memory: "500Mi"
      limits:
        cpu: "1"
        memory: "1Gi"
    replicas: 1
    image: vesoft/nebula-graphd
    version: v3.1.0
    storageClaim:
      resources:
        requests:
          storage: 2Gi
      storageClassName: gp2
    config: //为 Graph 服务自定义参数。
      "enable_authorize": "true"
      "auth_type": "password"
...
```

3. 执行 `kubectl apply -f nebula_cluster.yaml` 使上述更新生效。

在自定义参数 `enable_authorize` 和 `auth_type` 后，Graph 服务对应的 ConfigMap（`nebula-graphd`）中的配置将被覆盖。

更多信息

有关 Meta、Storage、Graph 服务的配置参数的详细介绍，参见服务配置文件。

2022 vesoft Inc.

最后更新: June 20, 2023

2022 vesoft Inc.

20.5.2 回收 PV

Nebula Operator 使用持久化卷 PV（Persistent Volume）和持久化卷声明 PVC（Persistent Volume Claim）来存储持久化数据。如果用户不小心删除了一个 Nebula Graph 集群，PV 和 PVC 对象及其数据仍可保留，以确保数据安全。

用户可以在集群的 CR 实例的配置文件中通过参数 `enablePVReclaim` 定义是否回收 PV。

如果用户需要删除图空间并想保留相关数据，可以更新 Nebula Graph 集群，即设置 `enablePVReclaim` 为 `true`。

前提条件

已使用 K8s 创建一个集群。具体步骤，参见使用 Kubectl 创建 Nebula Graph 集群。

2022 vesoft Inc.

操作步骤

以下示例使用名为 `nebula` 的集群、名为 `nebula_cluster.yaml` 的 YAML 配置文件，说明如何设置 `enablePVReclaim`：

1. 执行以下命令进入 `nebula` 集群的编辑页面。

```
kubectl edit nebulaclusters.apps.nebula-graph.io nebula
```

2. 在 YAML 文件的 `spec` 配置项中，添加 `enablePVReclaim` 并设置其值为 `true`。

```
apiVersion: apps.nebula-graph.io/v1alpha1
kind: NebulaCluster
metadata:
  name: nebula
spec:
  enablePVReclaim: true  //设置其值为 true。
  graphd:
    image: vesoft/nebula-graphd
    logVolumeClaim:
      resources:
        requests:
          storage: 2Gi
      storageClassName: fast-disks
    replicas: 1
    resources:
      limits:
        cpu: "1"
        memory: 1Gi
      requests:
        cpu: 500m
        memory: 500Mi
    version: v3.1.0
  imagePullPolicy: IfNotPresent
  metad:
    dataVolumeClaim:
      resources:
        requests:
          storage: 2Gi
      storageClassName: fast-disks
    image: vesoft/nebula-metad
    logVolumeClaim:
      resources:
        requests:
          storage: 2Gi
      storageClassName: fast-disks
    replicas: 1
    resources:
      limits:
        cpu: "1"
        memory: 1Gi
      requests:
        cpu: 500m
        memory: 500Mi
    version: v3.1.0
  nodeSelector:
    nebula: cloud
  reference:
    name: statefulsets.apps
    version: v1
  schedulerName: default-scheduler
  storaged:
    dataVolumeClaim:
      resources:
        requests:
          storage: 2Gi
      storageClassName: fast-disks
    image: vesoft/nebula-storaged
    logVolumeClaim:
      resources:
        requests:
          storage: 2Gi
      storageClassName: fast-disks
    replicas: 3
    resources:
      limits:
        cpu: "1"
        memory: 1Gi
      requests:
        cpu: 500m
        memory: 500Mi
    version: v3.1.0
...
```

a. 执行 `kubectl apply -f nebula_cluster.yaml` 使上述更新生效。

2022 vesoft Inc.

最后更新: June 20, 2023

2022 vesoft Inc.

20.5.3 均衡扩容后的 Storage 数据

Enterpriseonly

仅企业版 Nebula Graph 集群支持均衡扩容后的 Storage 数据。

用户在扩容 Storage 服务后，可以自主决定是否均衡 Storage 服务中的数据。本文介绍如何设置均衡 Storage 数据。

Nebula Graph 的 Storage 服务的扩容分为两个阶段，第一个阶段等待所有扩容的 Pods 为 `Ready` 的状态；第二个阶段执行 `BALANCE DATA` 和 `BALANCE LEADER` 命令均衡数据。这两个阶段将控制器副本的扩容过程与均衡数据过程解耦，使用户可以选择在低流量时进行均衡数据操作，有效降低数据迁移对在线服务的影响。

用户可以在集群的 CR 实例的配置文件中通过参数 `enableAutoBalance` 来控制是否自动均衡数据。

前提条件

已使用 K8s 创建一个集群。具体步骤，参见使用 Kubectl 创建 Nebula Graph 集群。

2022 vesoft Inc.

操作步骤

以下示例使用名为 `nebula` 的集群、名为 `nebula_cluster.yaml` 的 YAML 配置文件，说明如何设置 `enableAutoBalance`：

2022 vesoft Inc.

1. 执行以下命令进入 `nebula` 集群的编辑页面。

```
kubectl edit nebulaclusters.apps.nebula-graph.io nebula
```

2. 在 YAML 文件的 `spec.storaged` 配置项中，添加 `enableAutoBalance` 并设置其值为 `true`。

```
apiVersion: apps.nebula-graph.io/v1alpha1
kind: NebulaCluster
metadata:
  name: nebula
spec:
  graphd:
    image: vesoft/nebula-graphd
    logVolumeClaim:
      resources:
        requests:
          storage: 2Gi
      storageClassName: fast-disks
    replicas: 1
    resources:
      limits:
        cpu: "1"
        memory: 1Gi
      requests:
        cpu: 500m
        memory: 500Mi
    version: v3.1.0
  imagePullPolicy: IfNotPresent
  metad:
    dataVolumeClaim:
      resources:
        requests:
          storage: 2Gi
      storageClassName: fast-disks
    image: vesoft/nebula-metad
    logVolumeClaim:
      resources:
        requests:
          storage: 2Gi
      storageClassName: fast-disks
    replicas: 1
    resources:
      limits:
        cpu: "1"
        memory: 1Gi
      requests:
        cpu: 500m
        memory: 500Mi
    version: v3.1.0
  nodeSelector:
    nebula: cloud
  reference:
    name: statefulsets.apps
    version: v1
  schedulerName: default-scheduler
  storaged:
    enableAutoBalance: true   //将其值设置为 true 时表示扩容后自动均衡 Storage 数据。
    dataVolumeClaim:
      resources:
        requests:
          storage: 2Gi
      storageClassName: fast-disks
    image: vesoft/nebula-storaged
    logVolumeClaim:
      resources:
        requests:
          storage: 2Gi
      storageClassName: fast-disks
    replicas: 3
    resources:
      limits:
        cpu: "1"
        memory: 1Gi
      requests:
        cpu: 500m
        memory: 500Mi
    version: v3.1.0
...
```

- 当设置 `enableAutoBalance` 的值为 `true` 时，表示扩容后自动均衡 Storage 数据。
- 当设置 `enableAutoBalance` 的值为 `false` 时，表示扩容后不会均衡 Storage 数据。
- 当不设置 `enableAutoBalance` 参数时，默认扩容 Storage 后系统不会自动均衡 Storage 数据。

3. 执行 `kubectl apply -f nebula_cluster.yaml` 使上述更新生效。

2022 vesoft Inc.

最后更新: June 20, 2023

2022 vesoft Inc.

20.6 升级 Nebula Graph 集群

本文介绍如何升级通过 Nebula Operator 创建的 Nebula Graph 集群。

> 版本兼容性
>
> 1.x 版本的 Nebula Operator 不兼容 3.x 以下版本的 Nebula Graph。

20.6.1 使用限制

- 只支持使用 Nebula Operator 创建的 Nebula Graph 集群。
- 只支持升级 Nebula Graph 3.0.0 至 3.1.x 版本。

20.6.2 使用 Kubectl 升级 Nebula Graph 集群

前提条件

已创建 Nebula Graph 集群。具体步骤，参见使用 Kubectl 创建 Nebula Graph 集群。

本文示例中待升级的 Nebula Graph 版本为 3.0.0，其 YAML 文件名为 `apps_v1alpha1_nebulacluster.yaml`。

操作步骤

1. 查看集群中服务的镜像版本。

```
kubectl get pods -l app.kubernetes.io/cluster=nebula  -o jsonpath="{.items[*].spec.containers[*].image}" |tr -s '[[:space:]]' '\n' |sort |uniq -c
```

返回：

```
      1 vesoft/nebula-graphd:v3.0.0
      1 vesoft/nebula-metad:v3.0.0
      3 vesoft/nebula-storaged:v3.0.0
```

2. 编辑 `apps_v1alpha1_nebulacluster.yaml` 并将所有 `version` 的值从 3.0.0 修改至 v3.1.0。

修改后的 YAML 文件内容如下：

```
apiVersion: apps.nebula-graph.io/v1alpha1
kind: NebulaCluster
metadata:
  name: nebula
spec:
  graphd:
    resources:
      requests:
        cpu: "500m"
        memory: "500Mi"
      limits:
        cpu: "1"
        memory: "1Gi"
    replicas: 1
    image: vesoft/nebula-graphd
    version: v3.1.0 //将值从 3.0.0 修改至 v3.1.0。
    service:
      type: NodePort
      externalTrafficPolicy: Local
    logVolumeClaim:
      resources:
        requests:
          storage: 2Gi
      storageClassName: gp2
  metad:
    resources:
      requests:
        cpu: "500m"
        memory: "500Mi"
      limits:
        cpu: "1"
        memory: "1Gi"
    replicas: 1
```


2022 vesoft Inc.

```
    image: vesoft/nebula-metad
    version: v3.1.0 //将值从 3.0.0 修改至 v3.1.0。
    dataVolumeClaim:
      resources:
        requests:
          storage: 2Gi
      storageClassName: gp2
    logVolumeClaim:
      resources:
        requests:
          storage: 2Gi
      storageClassName: gp2
  storaged:
    resources:
      requests:
        cpu: "500m"
        memory: "500Mi"
      limits:
        cpu: "1"
        memory: "1Gi"
    replicas: 3
    image: vesoft/nebula-storaged
    version: v3.1.0 //将值从 3.0.0 修改至 v3.1.0。
    dataVolumeClaim:
      resources:
        requests:
          storage: 2Gi
      storageClassName: gp2
    logVolumeClaim:
      resources:
        requests:
          storage: 2Gi
      storageClassName: gp2
  reference:
    name: statefulsets.apps
    version: v1
  schedulerName: default-scheduler
  imagePullPolicy: Always
```

3. 执行以下命令将版本升级的更新应用于集群 CR 中。

```
kubectl apply -f apps_v1alpha1_nebulacluster.yaml
```

4. 等待约 2 分钟后，执行以下命令可查看到服务的镜像版本变更为 v3.1.0。

```
kubectl get pods -l app.kubernetes.io/cluster=nebula  -o jsonpath="{.items[*].spec.containers[*].image}" |tr -s '[[:space:]]' '\n' |sort |uniq -c
```

返回：

```
      1 vesoft/nebula-graphd:v3.1.0
      1 vesoft/nebula-metad:v3.1.0
      3 vesoft/nebula-storaged:v3.1.0
```

20.6.3 使用 Helm 升级 Nebula Graph 集群

前提条件

已创建 Nebula Graph 集群。具体步骤，参见使用 Helm 创建 Nebula Graph 集群。

操作步骤

1. 更新 Helm 仓库，拉取最新的仓库资源。

```
helm repo update
```

2. 配置 Helm 的环境变量。

```
export NEBULA_CLUSTER_NAME=nebula         # Nebula Graph 集群的名字。
export NEBULA_CLUSTER_NAMESPACE=nebula    # Nebula Graph 集群所处的命名空间的名字。
```

3. 升级 Nebula Graph 集群。

例如升级至 v3.1.0 Nebula Graph 集群的命令如下。

```
helm upgrade "${NEBULA_CLUSTER_NAME}" nebula-operator/nebula-cluster \
    --namespace="${NEBULA_CLUSTER_NAMESPACE}" \
    --set nameOverride=${NEBULA_CLUSTER_NAME} \
    --set nebula.version=v3.1.0
```


2022 vesoft Inc.

`--set nebula.version` 的值指需要升级集群的目标版本号。

4. 执行以下命令查看集群状态及集群版本。

   查看集群状态:

```
$ kubectl -n "${NEBULA_CLUSTER_NAMESPACE}" get pod -l "app.kubernetes.io/cluster=${NEBULA_CLUSTER_NAME}"
NAME                READY   STATUS    RESTARTS   AGE
nebula-graphd-0     1/1     Running   0          2m
nebula-graphd-1     1/1     Running   0          2m
nebula-metad-0      1/1     Running   0          2m
nebula-metad-1      1/1     Running   0          2m
nebula-metad-2      1/1     Running   0          2m
nebula-storaged-0   1/1     Running   0          2m
nebula-storaged-1   1/1     Running   0          2m
nebula-storaged-2   1/1     Running   0          2m
```

   查看集群版本:

```
$ kubectl get pods -l app.kubernetes.io/cluster=nebula  -o jsonpath="{.items[*].spec.containers[*].image}" |tr -s '[[:space:]]' '\n' |sort |uniq -c
      1 vesoft/nebula-graphd:v3.1.0
      1 vesoft/nebula-metad:v3.1.0
      3 vesoft/nebula-storaged:v3.1.0
```

最后更新: June 20, 2023

2022 vesoft Inc.

20.7 通过 Nebular Operator 连接 Nebula Graph 数据库

使用 Nebula Operator 创建 Nebula Graph 集群后，用户可在 Nebula Graph 集群内部访问 Nebula Graph 数据库，也可在集群外访问 Nebula Graph 数据库。

20.7.1 前提条件

使用 Nebula Operator 创建 Nebula Graph 集群。具体步骤参考使用 Kubectl 部署 Nebula Graph 集群 或者使用 Helm 部署 Nebula Graph 集群。

20.7.2 在 Nebula Graph 集群内连接 Nebula Graph 数据库

当使用 Nebula Operator 创建 Nebula Graph 集群后，Nebula Operator 会自动在同一命名空间下，创建名为 `<cluster-name>-graphd-svc` 、类型为 `ClusterIP` 的 Service。通过该 Service 的 IP 和数据库的端口号，用户可连接 Nebula Graph 数据库。

1. 查看 Service，命令如下：

```
$ kubectl get service -l app.kubernetes.io/cluster=<nebula>  #<nebula>为变量值，请用实际集群名称替换。
NAME                         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)                                 AGE
nebula-graphd-svc            ClusterIP   10.98.213.34   <none>        9669/TCP,19669/TCP,19670/TCP            23h
nebula-metad-headless        ClusterIP   None           <none>        9559/TCP,19559/TCP,19560/TCP            23h
nebula-storaged-headless     ClusterIP   None           <none>        9779/TCP,19779/TCP,19780/TCP,9778/TCP   23h
```

`ClusterIP` 类型的 Service 只允许在集群内部访问容器应用。更多信息，请参考 ClusterIP。

2. 使用上述 `<cluster-name>-graphd-svc` Service 的 IP 连接 Nebula Graph 数据库：

```
kubectl run -ti --image vesoft/nebula-console:v3.0.0 --restart=Never -- <nebula_console_name> -addr <cluster_ip>  -port <service_port> -u <username> -p <password>
```

示例：

```
kubectl run -ti --image vesoft/nebula-console:v3.0.0 --restart=Never -- nebula-console -addr 10.98.213.34  -port 9669 -u root -p vesoft
```

- `--image`：为连接 Nebula Graph 的工具 Nebula Console 的镜像。
- `<nebula-console>`：自定义的 Pod 名称。
- `-addr`：连接 Graphd 服务的 IP 地址，即 `ClusterIP` 类型的 Service IP 地址。
- `-port`：连接 Graphd 服务的端口。默认端口为 9669。
- `-u`：Nebula Graph 账号的用户名。未启用身份认证时，可以使用任意已存在的用户名（默认为 root）。
- `-p`：用户名对应的密码。未启用身份认证时，密码可以填写任意字符。

如果返回以下内容，说明成功连接数据库：

```
If you don't see a command prompt, try pressing enter.

(root@nebula) [(none)]>
```

用户还可以使用完全限定域名（**FQDN**）连接数据库，域名格式为 `<cluster-name>-graphd.<cluster-namespace>.svc.<CLUSTER_DOMAIN>`：

```
kubectl run -ti --image vesoft/nebula-console:v3.0.0 --restart=Never -- <nebula_console_name> -addr <cluster_name>-graphd-svc.default.svc.cluster.local -port <service_port> -u <username> -
p <password>
```

`CLUSTER_DOMAIN` 的默认值为 `cluster.local` 。

20.7.3 通过 `NodePort` 在 Nebula Graph 集群外部连接 Nebula Graph 数据库

用户可创建 `NodePort` 类型的 Service，通过节点 IP 和暴露的节点端口，从集群外部访问集群内部的服务。用户也可以使用云厂商（例如 Azure、AWS 等）提供的负载均衡服务，设置 Service 的类型为 `LoadBalancer` 。

`NodePort` 类型的 Service 通过标签选择器 `spec.selector` 将前端的请求转发到带有标签 `app.kubernetes.io/cluster: <cluster-name>` 、`app.kubernetes.io/component: graphd` 的 Graphd pod 中。

2022 vesoft Inc.

操作步骤如下：

1. 创建名为 `graphd-nodeport-service.yaml` 的文件。YAML 文件内容如下：

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app.kubernetes.io/cluster: nebula
    app.kubernetes.io/component: graphd
    app.kubernetes.io/managed-by: nebula-operator
    app.kubernetes.io/name: nebula-graph
  name: nebula-graphd-svc-nodeport
  namespace: default
spec:
  externalTrafficPolicy: Local
  ports:
  - name: thrift
    port: 9669
    protocol: TCP
    targetPort: 9669
  - name: http
    port: 19669
    protocol: TCP
    targetPort: 19669
  selector:
    app.kubernetes.io/cluster: nebula
    app.kubernetes.io/component: graphd
    app.kubernetes.io/managed-by: nebula-operator
    app.kubernetes.io/name: nebula-graph
  type: NodePort
```

- Nebula Graph 默认使用 `9669` 端口为客户端提供服务。`19669` 为 Graph 服务端口号。
- `targetPort` 的值为映射至 Pod 的端口，可自定义。

2. 执行以下命令使 Service 服务在集群中生效。

```
kubectl create -f graphd-nodeport-service.yaml
```

3. 查看 Service 中 Nebula Graph 映射至集群节点的端口。

```
kubectl get services
```

返回：

```
NAME                          TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)                                        AGE
nebula-graphd-svc             ClusterIP   10.98.213.34   <none>        9669/TCP,19669/TCP,19670/TCP                   23h
nebula-graphd-svc-nodeport    NodePort    10.107.153.129 <none>        9669:32236/TCP,19669:31674/TCP,19670:31057/TCP 24h
nebula-metad-headless         ClusterIP   None           <none>        9559/TCP,19559/TCP,19560/TCP                   23h
nebula-storaged-headless      ClusterIP   None           <none>        9779/TCP,19779/TCP,19780/TCP,9778/TCP          23h
```

NodePort 类型的 Service 中，映射至集群节点的端口为 `32236`。

4. 使用节点 IP 和上述映射的节点端口连接 Nebula Graph。

```
kubectl run -ti --image vesoft/nebula-console:v3.0.0 --restart=Never -- <nebula_console_name> -addr <node_ip> -port <node_port> -u <username> -p <password>
```

示例如下：

```
kubectl run -ti --image vesoft/nebula-console:v3.0.0 --restart=Never -- nebula-console2 -addr 192.168.8.24 -port 32236 -u root -p vesoft
If you don't see a command prompt, try pressing enter.

(root@nebula) [(none)]>
```

- `--image`：为连接 Nebula Graph 的工具 Nebula Console 的镜像。
- `<nebula-console>`：自定义的 Pod 名称。本示例为 `nebula-console2`。
- `-addr`：Nebula Graph 集群中任一节点 IP 地址。本示例为 `192.168.8.24`。
- `-port`：Nebula Graph 映射至节点的端口。本示例为 `32236`。
- `-u`：Nebula Graph 账号的用户名。未启用身份认证时，可以使用任意已存在的用户名（默认为 root）。
- `-p`：用户名对应的密码。未启用身份认证时，密码可以填写任意字符。

2022 vesoft Inc.

20.7.4 通过 Ingress 在 Nebula Graph 集群外部连接 Nebula Graph 数据库

Nginx Ingress 是 Kubernetes Ingress 的一个实现。Nginx Ingress 通过 Watch 机制感知 Kubernetes 集群的 Ingress 资源，将 Ingress 规则生成 Nginx 配置，使 Nginx 能够转发 7 层流量。

用户可以通过 HostNetwork 和 DaemonSet 组合的模式使用 Nginx Ingress 从集群外部连接 Nebula Graph 集群。

由于使用 HostNetwork，Nginx Ingress 的 Pod 就不能被调度在同一个节点上。为了避免监听端口冲突，可以事先选择一些节点并将其标记为边缘节点，专门用于部署 Nginx Ingress。然后 Nginx Ingress 以 DaemonSet 模式部署在这些节点上。

由于 Ingress 不支持 TCP 或 UDP 服务，为此 nginx-ingress-controller 使用 `--tcp-services-configmap` 和 `--udp-services-configmap` 参数指向一个 ConfigMap，该 ConfigMap 中的键指需要使用的外部端口，值指要公开的服务的格式，值的格式为 `<命名空间/服务名称>:<服务端口>` 。

例如指向名为 `tcp-services` 的 ConfigMap 的配置如下：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: tcp-services
  namespace: nginx-ingress
data:
  # update
  9769: "default/nebula-graphd-svc:9669"
```


2022 vesoft Inc.

操作步骤如下：

1. 创建名为 `nginx-ingress-daemonset-hostnetwork.yaml` 的文件。

   单击 nginx-ingress-daemonset-hostnetwork.yaml 查看完整的 YAML 示例内容。

   > **Note**
   >
   > 上述 YAML 中的资源对象均使用 `nginx-ingress` 命名空间。用户可执行 `kubectl create namesapce nginx-ingress` 创建命名空间，或者自定义其他命名空间。

2. 为任一节点（本示例使用的节点名为 `worker2`，IP 为 `192.168.8.160`）打上标签，以运行上述 YAML 文件中名为 `nginx-ingress-controller` 的 DaemonSet。

   ```
   kubectl label node worker2 nginx-ingress=true
   ```

3. 执行以下命令使 Nginx Ingress 在集群中生效。

   ```
   kubectl create -f nginx-ingress-daemonset-hostnetwork.yaml
   ```

   返回：

   ```
   configmap/nginx-ingress-controller created
   configmap/tcp-services created
   serviceaccount/nginx-ingress created
   serviceaccount/nginx-ingress-backend created
   clusterrole.rbac.authorization.k8s.io/nginx-ingress created
   clusterrolebinding.rbac.authorization.k8s.io/nginx-ingress created
   role.rbac.authorization.k8s.io/nginx-ingress created
   rolebinding.rbac.authorization.k8s.io/nginx-ingress created
   service/nginx-ingress-controller-metrics created
   service/nginx-ingress-default-backend created
   service/nginx-ingress-proxy-tcp created
   daemonset.apps/nginx-ingress-controller created
   ```

   成功部署 Nginx Ingress 后，由于 Nginx Ingress 中配置的网络类型为 `hostNetwork`，因此用户可通过部署了 Nginx Ingress 的节点的 IP（`192.168.8.160`）和外部端口（`9769`）访问 Nebula Graph 服务。

4. 执行以下命令部署连接 Nebula Graph 服务的 Console 并通过宿主机 IP（本示例为 `192.168.8.160`）和上述配置的外部端口访问 Nebula Graph 服务。

   ```
   kubectl run -ti --image vesoft/nebula-console:v3.0.0 --restart=Never -- <nebula_console_name> -addr <host_ip> -port <external_port> -u <username> -p <password>
   ```

   示例：

   ```
   kubectl run -ti --image vesoft/nebula-console:v3.0.0 --restart=Never -- nebula-console -addr 192.168.8.160 -port 9769 -u root -p vesoft
   ```

   - `--image`：为连接 Nebula Graph 的工具 Nebula Console 的镜像。
   - `<nebula-console>`：自定义的 Pod 名称。本示例为 `nebula-console`。
   - `-addr`：部署 Nginx Ingress 的节点 IP，本示例为 `192.168.8.160`。
   - `-port`：外网访问使用的的端口。本示例设置为 `9769`。
   - `-u`：Nebula Graph 账号的用户名。未启用身份认证时，可以使用任意已存在的用户名（默认为 root）。
   - `-p`：用户名对应的密码。未启用身份认证时，密码可以填写任意字符。

   如果返回以下内容，说明成功连接数据库：

   ```
   If you don't see a command prompt, try pressing enter.

   (root@nebula) [(none)]>
   ```

最后更新: June 20, 2023

2022 vesoft Inc.

20.8 故障自愈

Nebula Operator 调用 Nebula Graph 集群提供的接口，动态地感知服务是否正常运行。当 Nebula Graph 集群中某一组件停止运行时，Nebula Operator 会自动地进行容错处理。本文通过删除 Nebula Graph 集群中 1 个 Storage 服务 Pod，模拟集群故障为例，说明 Nebular Operator 如何进行故障自愈。

20.8.1 前提条件

安装 Nebula Operator

20.8.2 操作步骤

1. 创建 Nebula Graph 集群。具体步骤参考使用 Kubectl 部署 Nebula Graph 集群或者使用 Helm 部署 Nebula Graph 集群。
2. 待所有 Pods 都处于 `Running` 状态时，模拟故障，删除名为 `<cluster_name>-storaged-2` Pod。

```
kubectl delete pod <cluster-name>-storaged-2 --now
```

`<cluster_name>` 为 Nebula Graph 集群的名称。

3. Nebula Operator 自动创建名为 `<cluster-name>-storaged-2` 的 Pod，以修复故障。

执行 `kubectl get pods` 查看 `<cluster-name>-storaged-2` Pod 的创建状态。

```
...
nebula-cluster-storaged-1        1/1     Running            0          5d23h
nebula-cluster-storaged-2        0/1     ContainerCreating  0          1s
...
```

```
...
nebula-cluster-storaged-1        1/1     Running     0          5d23h
nebula-cluster-storaged-2        1/1     Running     0          4m2s
...
```

当 `<cluster-name>-storaged-2` 的状态由 `ContainerCreating` 变为 `Running` 时，说明自愈成功。

最后更新: June 20, 2023

2022 vesoft Inc.

20.9 常见问题

20.9.1 Nebula Operator 支持 v1.x 版本的 Nebula Graph 吗?

不支持，因为 v1.x 版本的 Nebula Graph 不支持 DNS，而 Nebula Operator 需要使用 DNS。

20.9.2 Nebula Operator 是否支持滚动升级 Nebula Graph?

只支持升级 2.5.x 版本的 Nebula Graph 至 2.6.x。

20.9.3 使用本地存储是否可以保证集群稳定性?

无法保证。使用本地存储意味着 Pod 被绑定到一个特定的节点，Nebula Operator 目前不支持在绑定的节点发生故障时进行故障转移。

20.9.4 扩缩容集群时，如何确保稳定性?

建议提前备份数据，以便故障发生时回滚数据。

最后更新: June 20, 2023

2022 vesoft Inc.

21. 图计算

21.1 算法简介

图计算可以检测图结构，例如图中社区的检测、图的划分等，也可以揭示各个点之间关联关系的内在特征，例如点的中心性、相似性等。本文介绍相关算法和参数。

> **Note**
>
> 本文仅介绍 Nebula Analytics 的参数，Nebula Algorithm 的参数请先参见对应的算法文件。

> **Note**
>
> 执行图计算时不仅需要设置算法的参数，对数据源也有要求。数据源需要包含起点和终点。PageRank、DegreeWithTime、SSSP、APSP、LPA、HANP、Louvain 算法还需要包含权重（weight）。

- 如果数据源来自 HDFS，需要指定 CSV 文件，包含 `src` 和 `dst` 列，部分算法还需要包含 `weight` 列。
- 如果数据源来自 Nebula Graph，需要指定边类型，该类型的边提供 `src` 和 `dst` 列，部分算法还需要指定边类型的某个属性作为 `weight` 列。

21.1.1 节点重要度算法

PageRank

PageRank（页面排序）算法根据点之间的关系（边）计算点的相关性和重要性，通常使用在搜索引擎页面排名中。如果一个网页被很多其他网页链接，说明这个网页比较重要（PageRank 值较高）；如果一个 PageRank 值很高的网页链接到其他网页，那么被链接到的网页的 PageRank 值会提高。

参数说明如下。

- Nebula Analytics
- 传入参数

参数	默认值	说明
`ITERATIONS`	`10`	最大迭代次数。
`IS_DIRECTED`	`true`	是否考虑边的方向。如果设置为 `false`，系统会自动添加反向边。
`EPS`	`0.0001`	收敛精度，两轮迭代的结果差值小于这个值，结束迭代。
`DAMPING`	`0.85`	阻尼系数，访问页面后的跳转概率。

- 输出参数

参数	类型	说明
`VID`	创建图空间时 `vid_type` 决定	点 ID。
`VALUE`	double	点的 PageRank 值。

KCore

KCore 算法用于计算出没有小于 K 度的点组成的子图，通常使用在社区发现、金融风控等场景。其计算结果是判断点重要性最常用的参考值之一，体现了点的传播能力。

2022 vesoft Inc.

参数说明如下。

- Nebula Analytics
- 传入参数

参数	默认值	说明
`TYPE`	`vertex`	计算类型。取值：`vertex`、`subgraph`。`vertex` 表示为每个点计算核心度，`subgraph` 表示计算邻居。
`KMIN`	`1`	范围计算时设置 K 的最小值。仅在 `TYPE` = `subgraph` 时生效。
`KMAX`	`1000000`	范围计算时设置 K 的最大值。仅在 `TYPE` = `subgraph` 时生效。

- `TYPE=vertex` 时的输出参数

参数	类型	说明
`VID`	创建图空间时 `vid_type` 决定	点 ID。
`VALUE`	int	输出点的核心度。

- `TYPE=subgraph` 时的输出参数

参数	类型	说明
`VID`	创建图空间时 `vid_type` 决定	点 ID。
`VALUE`	与 `VID` 类型相同	输出点的邻居。

DegreeCentrality（NStepDegree）

DegreeCentrality（度中心性） 算法用于查找图中的流行点。度中心性测量来自点的传入或传出（或两者）关系的数量，具体取决于关系投影的方向。一个点的度越大就意味着这个点的度中心性越高，该点在网络中就越重要。

> **Note**
>
> Nebula Analytics 仅粗略估算度中心性。

2022 vesoft Inc.

参数说明如下。

- Nebula Analytics
- 传入参数

参数	默认值	说明
`STEP`	`3`	计算度数。`-1` 表示无穷大。
`BITS`	`6`	用于基数估计的 hyperloglog 位宽。
`TYPE`	`both`	计算的边的方向。取值：`in`、`out`、`both`。

- `TYPE=both` 时的输出参数

参数	类型	说明
`VID`	创建图空间时 `vid_type` 决定	点 ID。
`BOTH_DEGREE`	int	输出点的双向度中心性。
`OUT_DEGREE`	int	输出点的出方向度中心性。
`IN_DEGREE`	int	输出点的入方向度中心性。

- `TYPE=out` 时的输出参数

参数	类型	说明
`VID`	创建图空间时 `vid_type` 决定	点 ID。
`OUT_DEGREE`	int	输出点的出方向度中心性。

- `TYPE=in` 时的输出参数

参数	类型	说明
`VID`	创建图空间时 `vid_type` 决定	点 ID。
`IN_DEGREE`	int	输出点的入方向度中心性。

DegreeWithTime

DegreeWithTime 算法是基于边的时间范围统计邻居，查找出图中的流行点。

> **Note**
>
> 仅 Nebula Analytics 支持该算法。

2022 vesoft Inc.

参数说明如下。

- 传入参数

参数	默认值	说明
`TYPE`	`both`	计算的边的方向。取值：`in`、`out`、`both`。
`BEGIN_TIME`	-	起始时间。格式为`yyyy-MM-dd HH:mm:ss.SSS`。
`END_TIME`	-	结束时间。格式为`yyyy-MM-dd HH:mm:ss.SSS`。

- `TYPE=both`时的输出参数

参数	类型	说明
`VID`	创建图空间时`vid_type`决定	点 ID。
`BOTH_DEGREE`	int	输出点的双向流行度。
`OUT_DEGREE`	int	输出点的出方向流行度。
`IN_DEGREE`	int	输出点的入方向流行度。

- `TYPE=out`时的输出参数

参数	类型	说明
`VID`	创建图空间时`vid_type`决定	点 ID。
`OUT_DEGREE`	int	输出点的出方向流行度。

- `TYPE=in`时的输出参数

参数	类型	说明
`VID`	创建图空间时`vid_type`决定	点 ID。
`IN_DEGREE`	int	输出点的入方向流行度。

BetweennessCentrality

BetweennessCentrality（中介中心性）算法是一种检测点对图中信息流的影响量的方法，用于查找从图的一部分到另一部分时作为桥梁的点。每个点都会根据通过该点的最短路径的数量获得一个分数，即中介中心性分数。

2022 vesoft Inc.

参数说明如下。

- Nebula Analytics
- 传入参数

参数	默认值	说明
`ITERATIONS`	`10`	最大迭代次数。
`IS_DIRECTED`	`true`	是否考虑边的方向。如果设置为 `false`，系统会自动添加反向边。
`CHOSEN`	`-1`	选取的点ID，`-1` 表示随机选。
`CONSTANT`	`2`	系数。

- 输出参数

参数	类型	说明
`VID`	创建图空间时 `vid_type` 决定	点 ID。
`VALUE`	double	点的中介中心性分数。

ClosenessCentrality

ClosenessCentrality（紧密中心性）算法用于计算一个点到所有其他可达点的最短距离的平均值的倒数。值越大，点在图中的位置越靠近中心，也可以用来衡量信息从该点传输到其他点的时间长短。

参数说明如下。

- Nebula Analytics
- 传入参数

参数	默认值	说明
`IS_DIRECTED`	`true`	是否考虑边的方向。如果设置为 `false`，系统会自动添加反向边。
`NUM_SAMPLES`	`10`	采样的点数量。

- 输出参数

参数	类型	说明
`VID`	创建图空间时 `vid_type` 决定	点 ID。
`VALUE`	double	点的紧密中心性分数。

21.1.2 路径算法

APSP

APSP（全图最短路径）算法用于寻找图中两点之间的所有最短路径。

> **Note**
>
> 仅 Nebula Analytics 支持该算法。

2022 vesoft Inc.

参数说明如下。

- 输出参数

参数	类型	说明
VID1	创建图空间时 vid_type 决定	起点的 ID。
VID2	创建图空间时 vid_type 决定	终点的 ID。
DISTANCE	double	输出 VID1 到 VID2 的距离。

SSSP

SSSP（单源最短路径）算法用于计算给定的一个点（起始点）出发到其余各点的最短路径长度。通常使用在网络路由、路径设计等场景。

参数说明如下。

- Nebula Analytics
- 传入参数

参数	默认值	说明
ROOT	-	起始点的 VID。

- 输出参数

参数	类型	说明
VID	创建图空间时 vid_type 决定	点 ID。
DISTANCE	double	输出 ROOT 到 VID 的距离。

BFS

BFS（广度优先遍历）算法是一种基础的图遍历算法，它给定一个起始点，以递增的跳数访问其他点，即先遍历点的所有相邻点，再往相邻点的相邻点延伸。

参数说明如下。

- Nebula Analytics
- 传入参数

参数	默认值	说明
IS_DIRECTED	true	是否考虑边的方向。如果设置为 false，系统会自动添加反向边。
ROOT	-	起始点的 VID。

- 输出参数

参数	类型	说明
ROOT	创建图空间时 vid_type 决定	起始点的 ID。
VISITED	int	输出 ROOT 访问过的点数量。

2022 vesoft Inc.

21.1.3 社区发现算法

LPA

LPA（标签传播）算法是一种基于图的半监督学习方法，其基本思路是用已标记点的标签信息去预测未标记点的标签信息。利用样本间的关系建图，点包括已标注和未标注数据，其边表示两个点的相似度，点的标签按相似度传递给其他点。标签数据就像是一个源头，可以对无标签数据进行标注，点的相似度越大，标签越容易传播。

参数说明如下。

- Nebula Analytics
- 传入参数

参数	默认值	说明
`ITERATIONS`	`10`	最大迭代次数。
`IS_DIRECTED`	`true`	是否考虑边的方向。如果设置为 `false`，系统会自动添加反向边。
`IS_CALC_MODULARITY`	`false`	是否计算模块度。

- 输出参数

参数	类型	说明
`VID`	创建图空间时 `vid_type` 决定	点的 ID。
`LABEL`	与 `VID` 类型相同	输出标签相同的点的 ID。

HANP

HANP（Hop Attenuation & Node Preference）算法是LPA算法的优化算法，考虑了标签的其他信息，例如度的信息、距离信息等，同时在传播时引入了衰减系数，防止过渡传播。

参数说明如下。

- Nebula Analytics
- 传入参数

参数	默认值	说明
`ITERATIONS`	`10`	最大迭代次数。
`IS_DIRECTED`	`true`	是否考虑边的方向。如果设置为 `false`，系统会自动添加反向边。
`PREFERENCE`	`1.0`	对邻居节点度的偏向性。`m>0` 表示偏向节点度高的邻居，`m<0` 表示偏向节点度低的邻居，`m=0` 表示不考虑邻居节点度。
`HOP_ATT`	`0.1`	衰减因子。取值范围 `0` ~ `1`。值越大衰减的越快，可以传递的次数越少。

- 输出参数

参数	类型	说明
`VID`	创建图空间时 `vid_type` 决定	点的 ID。
`LABEL`	与 `VID` 类型相同	输出标签相同的点的 ID。

ConnectedComponent

ConnectedComponent（联通分量）算法用于计算出图中的一个子图，当中所有节点都相互连接。考虑路径方向的为强联通分量（strongly connected component），不考虑路径方向的为弱联通分量（weakly connected component）。

2022 vesoft Inc.

> **Note**
>
> Nebula Analytics 仅支持弱联通分量。

参数说明如下。

- Nebula Analytics
- 传入参数

参数	默认值	说明
`IS_DIRECTED`	`true`	是否考虑边的方向。如果设置为 `false`，系统会自动添加反向边。
`IS_CALC_MODULARITY`	`false`	是否计算模块度。

- 输出参数

参数	类型	说明
`VID`	创建图空间时 `vid_type` 决定	点的 ID。
`LABEL`	与 `VID` 类型相同	输出标签相同的点的 ID。

Louvain

Louvain 算法是基于模块度的社区发现算法，该算法在效率和效果上都表现较好，并且能够发现层次性的社区结构，其优化目标是最大化整个社区网络的模块度。模块度用于区分社区内和社区间链路密度的差异，是衡量每个点划分社区的好坏。通常情况下，一个优秀的分群方法将会使得社区内部的模块度高于社区与社区之间。

参数说明如下。

- Nebula Analytics
- 传入参数

参数	默认值	说明
`IS_DIRECTED`	`true`	是否考虑边的方向。如果设置为 `false`，系统会自动添加反向边。
`OUTER_ITERATION`	`20`	第一阶段最大迭代次数。
`INNER_ITERATION`	`10`	第二阶段最大迭代次数。
`IS_CALC_MODULARITY`	`false`	是否计算模块度。

- 输出参数

参数	类型	说明
`VID`	创建图空间时 `vid_type` 决定	点的 ID。
`LABEL`	与 `VID` 类型相同	输出标签相同的点的 ID。

21.1.4 图特征算法

TriangleCount

TriangleCount（三角计数）算法用于统计图中三角形个数。三角形越多，代表图中节点关联程度越高，组织关系越严密。

2022 vesoft Inc.

参数说明如下。

- Nebula Analytics
- 传入参数

参数	默认值	说明
`OPT`	`3`	计算类型。取值：`1`（统计整个图）、`2`（通过每个点统计）、`3`（列出所有三角形）。
`REMOVED_DUPLICATION_EDGE`	`true`	是否排除重复边。
`REMOVED_SELF_EDGE`	`true`	是否排除自环边。

- `OPT=1` 时的输出参数

参数	类型	说明
`COUNT`	int	输出全图的三角形数量。

- `OPT=2` 时的输出参数

参数	类型	说明
`VID`	创建图空间时 `vid_type` 决定	输出每个点的 ID。
`COUNT`	int	输出每个点的三角形数量。

- `OPT=3` 时的输出参数

参数	类型	说明
`VID1`	与 `VID` 类型相同	输出构成三角形的点 A 的 ID。
`VID2`	与 `VID` 类型相同	输出构成三角形的点 B 的 ID。
`VID3`	与 `VID` 类型相同	输出构成三角形的点 C 的 ID。

21.1.5 聚类算法

ClusteringCoefficient

ClusteringCoefficient（聚集系数）算法用于计算图中节点的聚集程度。在各类反映真实世界的网络结构，特别是社交网络结构中，各个点之间倾向于形成密度相对较高的网络群，也就是说，相对于在两个点之间随机连接而得到的网络，真实世界网络的聚集系数更高。

2022 vesoft Inc.

参数说明如下。

- Nebula Analytics
- 传入参数

参数	默认值	说明
`TYPE`	`local`	聚集类型。取值：`local`（为每个点计算聚集系数）、`global`（为全图计算聚集系数）。
`REMOVED_DUPLICATION_EDGE`	`true`	是否排除重复边。
`REMOVED_SELF_EDGE`	`true`	是否排除自环边。

- `TYPE=local` 时的输出参数

参数	类型	说明
`VID`	创建图空间时 `vid_type` 决定	点的 ID。
`VALUE`	double	输出每个点的聚集系数。

- `TYPE=global` 时的输出参数

参数	类型	说明
`VID`	创建图空间时 `vid_type` 决定	点的 ID。
`VALUE`	double	输出全图的聚集系数。只有一行数据。

21.1.6 相似度算法

Jaccard

Jaccard（杰卡德相似度）算法用于计算两个点（或集合）的相似程度，预测他们之间的关系。适用于社交网上的好友推荐、关系预测等场景。

参数说明如下。

- Nebula Analytics
- 传入参数

参数	默认值	说明
`IDS1`	-	若干个 VID 构成的集合A。多个 VID 之间用英文逗号（,）隔开。不可为空。
`IDS2`	-	若干个 VID 构成的集合B。多个 VID 之间用英文逗号（,）隔开。可以为空，为空时表示所有点。
`REMOVED_SELF_EDGE`	`true`	是否排除自环边。

- 输出参数

参数	类型	说明
`VID1`	创建图空间时 `vid_type` 决定	第一个点的 ID。
`VID2`	创建图空间时 `vid_type` 决定	第二个点的 ID。
`VALUE`	double	`VID1` 和 `VID2` 的相似度。

最后更新: June 20, 2023

2022 vesoft Inc.

21.2 Nebula Algorithm

Nebula Algorithm （简称 Algorithm）是一款基于 GraphX 的 Spark 应用程序，通过提交 Spark 任务的形式使用完整的算法工具对 Nebula Graph 数据库中的数据执行图计算，也可以通过编程形式调用 lib 库下的算法针对 DataFrame 执行图计算。

21.2.1 版本兼容性

Nebula Algorithm 版本和 Nebula Graph 内核的版本对应关系如下。

Nebula Algorithm 版本	**Nebula Graph** 版本
3.0-SNAPSHOT	nightly
3.0.0	3.1.0
2.6.x	2.6.x
2.5.0	2.5.0、2.5.1
2.1.0	2.0.0、2.0.1

21.2.2 前提条件

在使用 Algorithm 之前，用户需要确认以下信息：

- Nebula Graph 服务已经部署并启动。详细信息，参考 Nebula Graph 安装部署。
- Spark 版本为 2.4.x。
- Scala 版本为 2.11。
- （可选）如果用户需要在 Github 中克隆最新的 Algorithm，并自行编译打包，可以选择安装 Maven。

21.2.3 使用限制

- 直接提交算法包时，点 ID 的数据必须为整数，即点 ID 可以是 INT 类型，或者是 String 类型但数据本身为整数。
- 对于非整数的 String 类型数据，推荐使用调用算法接口的方式，可以使用 SparkSQL 的 `dense_rank` 函数进行编码，将 String 类型转换为 Long 类型。
- 图计算会输出点的数据集，算法结果会以DataFrame形式作为点的属性存储。用户可以根据业务需求，自行对算法结果做进一步操作，例如统计、筛选。

2022 vesoft Inc.

21.2.4 支持算法

Nebula Algorithm 支持的图计算算法如下。

算法名	说明	应用场景	属性名称	属性数据类型
PageRank	页面排序	网页排序、重点节点挖掘	pagerank	double/string
Louvain	鲁汶	社团挖掘、层次化聚类	louvain	int/string
KCore	K 核	社区发现、金融风控	kcore	int/string
LabelPropagation	标签传播	资讯传播、广告推荐、社区发现	lpa	int/string
Hanp	标签传播进阶版	社区发现、推荐	hanp	int/string
ConnectedComponent	弱联通分量	社区发现、孤岛发现	cc	int/string
StronglyConnectedComponent	强联通分量	社区发现	scc	int/string
ShortestPath	最短路径	路径规划、网络规划	shortestpath	string
TriangleCount	三角形计数	网络结构分析	trianglecount	int/string
GraphTriangleCount	全图三角形计数	网络结构及紧密程度分析	count	int
BetweennessCentrality	中介中心性	关键节点挖掘，节点影响力计算	betweenness	double/string
ClosenessCentrality	紧密中心性	关键节点挖掘、节点影响力计算	closeness	double/string
DegreeStatic	度统计	图结构分析	degree,inDegree,outDegree	int/string
ClusteringCoefficient	聚集系数	推荐、电信诈骗分析	clustercoefficient	double/string
Jaccard	杰卡德相似度计算	相似度计算、推荐	jaccard	string
BFS	广度优先遍历	层序遍历、最短路径规划	bfs	string
Node2Vec	-	图分类	node2vec	string

Note

如果需要将算法结果写入到 Nebula Graph 中，请确保对应图空间中的 Tag 有和上表对应的属性名称和数据类型。

21.2.5 实现方法

Nebula Algorithm 实现图计算的流程如下：

1. 利用 Nebula Spark Connector 从 Nebula Graph 数据库中读取图数据为 DataFrame。
2. 将 DataFrame 转换为 GraphX 的图。
3. 调用 GraphX 提供的图算法（例如 PageRank）或者自行实现的算法（例如 Louvain 社区发现）。

详细的实现方法可以参见相关 Scala 文件。

2022 vesoft Inc.

21.2.6 获取 Nebula Algorithm

编译打包

1. 克隆仓库 `nebula-algorithm` 。

```
$ git clone -b v3.0.0 https://github.com/vesoft-inc/nebula-algorithm.git
```

2. 进入目录 `nebula-algorithm` 。

```
$ cd nebula-algorithm
```

3. 编译打包。

```
$ mvn clean package -Dgpg.skip -Dmaven.javadoc.skip=true -Dmaven.test.skip=true
```

编译完成后，在目录 `nebula-algorithm/target` 下生成类似文件 `nebula-algorithm-3.x.x.jar` 。

Maven 远程仓库下载

下载地址

21.2.7 使用方法

调用算法接口（推荐）

`lib` 库中提供了 10 种常用图计算算法，用户可以通过编程调用的形式调用算法。

1. 在文件 `pom.xml` 中添加依赖。

```
<dependency>
    <groupId>com.vesoft</groupId>
    <artifactId>nebula-algorithm</artifactId>
    <version>3.0.0</version>
</dependency>
```

2. 传入参数调用算法（以 PageRank 为例）。更多调用示例请参见示例。

> **Note**
>
> 执行算法的 DataFrame 默认第一列是起始点，第二列是目的点，第三列是边权重（非 Nebula Graph 中的 Rank）。

```
val prConfig = new PRConfig(5, 1.0)
val louvainResult = PageRankAlgo.apply(spark, data, prConfig, false)
```

如果用户的节点 ID 是 String 类型，可以参考 PageRank 的示例。示例中进行了 ID 转换，将 String 类型编码为 Long 类型， 并在算法结果中将 Long 类型 ID 解码为原始的 String 类型。

2022 vesoft Inc.

直接提交算法包

> **Note**
>
> 使用封装好的算法包有一定的局限性，例如落库到 Nebula Graph 时，落库的图空间中创建的 Tag 的属性名称必须和代码内预设的名称保持一致。如果用户有开发能力，推荐使用第一种方法。

2022 vesoft Inc.

1. 设置配置文件。

```
{
    # Spark 相关配置
    spark: {
    app: {
        name: LPA
        # Spark 分片数量
        partitionNum:100
    }
    master:local
    }

    data: {
    # 数据源，可选值为 nebula、csv、json。
    source: nebula
    # 数据落库，即图计算的结果写入的目标，可选值为 nebula、csv、json。
    sink: nebula
    # 算法是否需要权重。
    hasWeight: false
    }

    # Nebula Graph 相关配置
    nebula: {
    # 数据源。Nebula Graph 作为图计算的数据源时，nebula.read 的配置才生效。
    read: {
        # 所有 Meta 服务的 IP 地址和端口，多个地址用英文逗号（,）分隔。格式："ip1:port1,ip2:port2"。
        # 使用 docker-compose 部署，端口需要填写 docker-compose 映射到外部的端口
        # 可以用`docker-compose ps`查看
        metaAddress: "192.168.*.10:9559"
        # Nebula Graph 图空间名称
        space: basketballplayer
        # Nebula Graph Edge type，多个 labels 时，多个边的数据将合并。
        labels: ["serve"]
        # Nebula Graph 每个 Edge type 的属性名称，此属性将作为算法的权重列，请确保和 Edge type 对应。
        weightCols: ["start_year"]
    }

    # 数据落库。图计算结果落库到 Nebula Graph 时，nebula.write 的配置才生效。
    write:{
        # Graph 服务的 IP 地址和端口，多个地址用英文逗号（,）分隔。格式："ip1:port1,ip2:port2"。
        # 使用 docker-compose 部署，端口需要填写 docker-compose 映射到外部的端口
        # 可以用`docker-compose ps`查看
        graphAddress: "192.168.*.11:9669"
        # 所有 Meta 服务的 IP 地址和端口，多个地址用英文逗号（,）分隔。格式："ip1:port1,ip2:port2"。
        # 使用 docker-compose 部署，端口需要填写 docker-compose 映射到外部的端口
        # 可以用`docker-compose ps`查看
        metaAddress: "192.168.*.12:9559"
        user:root
        pswd:nebula
        # 在提交图计算任务之前需要自行创建图空间及 Tag
        # Nebula Graph 图空间名称
        space:nb
        # Nebula Graph Tag 名称，图计算结果会写入该 Tag。Tag 中的属性名称固定如下：
        # PageRank：pagerank
        # Louvain：louvain
        # ConnectedComponent：cc
        # StronglyConnectedComponent：scc
        # LabelPropagation：lpa
        # ShortestPath：shortestpath
        # DegreeStatic：degree、inDegree、outDegree
        # KCore：kcore
        # TriangleCount：tranglecpunt
        # BetweennessCentrality：betweennedss
        tag:pagerank
    }
    }

    local: {
    # 数据源。图计算的数据源为 csv 文件或 json 文件时，local.read 的配置才生效。
    read:{
        filePath: "hdfs://127.0.0.1:9000/edge/work_for.csv"
        # 如果 CSV 文件没有表头，使用 [_c0, _c1, _c2, ..., _cn] 表示其表头，有表头或者是 json 文件时，直接使用表头名称即可。
        # 起始点 ID 列的表头。
        srcId:"_c0"
        # 目的点 ID 列的表头。
        dstId:"_c1"
        # 权重列的表头
        weight: "_c2"
        # csv 文件是否有表头
        header: false
        # csv 文件的分隔符
        delimiter:","
    }

    # 数据落库。图计算结果落库到 csv 文件或 text 文件时，local.write 的配置才生效。
    write:{
        resultPath:/tmp/
    }

    algorithm: {
    # 需要执行的算法，可选值为：
    # pagerank、louvain、connectedcomponent、labelpropagation、shortestpaths、
```


2022 vesoft Inc.

```
    # degreestatic、kcore、stronglyconnectedcomponent、trianglecount、
    # betweenness、graphtriangleCount。
    executeAlgo: pagerank

    # PageRank 参数
    pagerank: {
        maxIter: 10
        resetProb: 0.15
    }

    # Louvain 参数
    louvain: {
        maxIter: 20
        internalIter: 10
        tol: 0.5
    }

    # ...
    }
    }
}
```

2. 提交图计算任务。

```
${SPARK_HOME}/bin/spark-submit --master <mode> --class com.vesoft.nebula.algorithm.Main <nebula-algorithm-3.0.0.jar_path> -p <application.conf_path>
```

示例：

```
${SPARK_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.algorithm.Main /root/nebula-algorithm/target/nebula-algorithm-3.0-SNAPSHOT.jar -p /root/nebula-algorithm/src/main/
resources/application.conf
```

21.2.8 视频

- 图计算工具——Nebula Algorithm 介绍（2 分 36 秒）

最后更新: June 20, 2023

2022 vesoft Inc.

21.3 Nebula Analytics

Nebula Analytics 是一款高性能图计算框架工具，支持对 Nebula Graph 数据库中的数据执行图分析。

Enterpriseonly

仅企业版支持本功能。

21.3.1 适用场景

支持将数据源为 Nebula Graph 集群、HDFS 上的 CSV 文件或本地 CSV 文件中的数据导入 Nebula Analytics，并将图计算结果输出至 Nebula Graph 集群、HDFS 上的 CSV 文件或本地 CSV 文件。

21.3.2 使用限制

输入和输出均为 Nebula Graph 集群时，图计算结果只能输出到数据源所在的图空间。

21.3.3 版本兼容性

Nebula Analytics 版本和 Nebula Graph 内核的版本对应关系如下。

Nebula Analytics 版本	**Nebula Graph** 版本
3.2.0	3.1.0
1.0.x	3.0.x
0.9.0	2.6.x

2022 vesoft Inc.

21.3.4 支持算法

Nebula Analytics 支持的图计算算法如下。

算法名	说明	分类
APSP	全图最短路径	路径
SSSP	单源最短路径	路径
BFS	广度优先遍历	路径
PageRank	页面排序	节点重要度
KCore	K核	节点重要度
DegreeCentrality	度中心性	节点重要度
DegreeWithTime	基于边的时间范围统计邻居	节点重要度
BetweennessCentrality	中介中心性	节点重要度
ClosenessCentrality	紧密中心性	节点重要度
TriangleCount	三角计数	图特征
LPA	标签传播	社区发现
HANP	标签传播进阶版	社区发现
WCC	弱联通分量	社区发现
LOUVAIN	鲁汶	社区发现
Clustering Coefficient	聚集系数	聚类
Jaccard	杰卡德相似度	相似度

21.3.5 安装 Nebula Analytics

在多个机器安装由多个 Nebula Analytics 服务构成的集群时，需要安装路径相同，并设置节点间 SSH 免密登录。

```
sudo rpm -i nebula-analytics-3.2.0-centos.x86_64.rpm  --prefix /home/xxx/nebula-analytics
```


2022 vesoft Inc.

21.3.6 使用方法

安装完成后，用户可以设置不同算法的参数，然后执行脚本，即可获得算法的结果，并导出为指定格式。

2022 vesoft Inc.

1. 选择 Nebula Analytics 集群的任一节点，进入目录 `scripts` 。

```
$ cd scripts
```


2022 vesoft Inc.

2. 确认数据源和输出路径。配置方法如下：

- 数据源为 Nebula Graph 集群

a. 修改配置文件 `nebula.conf`，设置 Nebula Graph 集群相关信息。

```
# 连接 Nebula Graph 时的重试次数。
--retry=3
# 要读取或写入的图空间名称。
--space=baskeyballplayer

# 读取 Nebula Graph 设置
# Nebula Graph 的 metad leader服务地址，例如为192.168.8.101。
--meta_server_addrs=192.168.8.101:9559
# 要读取的边的名称。
--edges=LIKES
# 要读取的作为边的权重属性的名称。可以是属性名，也可以是 _rank。
#--edge_data_fields
# 每次扫描读取的行数。
--read_batch_size=10000

# 写回 Nebula Graph 设置
# Nebula Graph 的 graphd 服务地址。
--graph_server_addrs=192.168.8.100:9669
# Nebula Graph 的登录用户名。
--user=root
# Nebula Graph 的登录密码。
--password=nebula
# 写回 Nebula Graph 时采用的模式：insert 和 update。
--mode=insert
# 写回到 Nebula Graph 的 Tag 名称。
--tag=pagerank
# 写回到 Nebula Graph 的 Tag 对应的属性名称。
--prop=pr
# 写回到 Nebula Graph 的 Tag 对应的属性的类型。
--type=double
# 写回时，每次写入的行数。
--write_batch_size=1000
# 写回失败的数据所存储的文件。
--err_file=/home/xxx/analytics/err.txt
```

b. 修改需要使用的算法脚本，例如 `run_pagerank.sh`，设置相关参数。

```
# 集群所有机器所运行的进程数之和，推荐每台机器为 1 或者 NUMA 架构的 node 数。
WNUM=3
# 每个进程的线程数，推荐最大设置为机器的硬件线程数。
WCORES=4
# 数据源路径
# 可以通过文件 nebula.conf 设置从 Nebula Graph 读取：
INPUT=${INPUT:="nebula:$PROJECT/scripts/nebula.conf"}
# 也可以通过本地或 HDFS 上的 CSV 文件读取：
# #INPUT=${INPUT:="$PROJECT/data/graph/v100_e2150_ua_c3.csv"}

# 图计算结果输出路径
# 可以输出至 Nebula Graph 集群，如果数据源也为 Nebula Graph，结果会输出至 nebula.conf 指定的图空间。
OUTPUT=${OUTPUT:="nebula:$PROJECT/scripts/nebula.conf"}
# 也可以输出至本地或 HDFS 上的 CSV 文件：
# OUTPUT=${OUTPUT:='hdfs://192.168.8.100:9000/_test/output'}

# true 为有向图，false 为无向图。
IS_DIRECTED=${IS_DIRECTED:=true}
# 是否进行 ID 编码
NEED_ENCODE=${NEED_ENCODE:=true}
# 数据源的点 ID 类型，例如：string、int32、int64。
VTYPE=${VTYPE:=int32}
# 编码类型。distributed 为分布式点ID编码，single 为单机点 ID 编码。
ENCODER=${ENCODER:="distributed"}
# PageRank 算法的参数。不同算法的参数不同。
EPS=${EPS:=0.0001}
DAMPING=${DAMPING:=0.85}
# 迭代次数
ITERATIONS=${ITERATIONS:=100}
```


2022 vesoft Inc.

- 数据源为本地或 HDFS 上的 CSV 文件

  修改需要使用的算法脚本，例如 `run_pagerank.sh`，设置相关参数。

```
# 集群所有机器所运行的进程数之和，推荐每台机器为 1 或者 NUMA 架构的 node 数。
WNUM=3
# 每个进程的线程数，推荐最大设置为机器的硬件线程数。
WCORES=4
# 数据源路径
# 可以通过文件 nebula.conf 设置从 Nebula Graph 读取：
# INPUT=${INPUT:="nebula:$PROJECT/scripts/nebula.conf"}
# 也可以通过本地或 HDFS 上的 CSV 文件读取：
INPUT=${INPUT:="$PROJECT/data/graph/v100_e2150_ua_c3.csv"}

# 图计算结果输出路径
# 可以输出至 Nebula Graph 集群，如果数据源也为 Nebula Graph，结果会输出至 nebula.conf 指定的图空间。
# OUTPUT=${OUTPUT:="nebula:$PROJECT/scripts/nebula.conf"}
# 也可以输出至本地或 HDFS 上的 CSV 文件：
OUTPUT=${OUTPUT:='hdfs://192.168.8.100:9000/_test/output'}

# true 为有向图，false 为无向图。
IS_DIRECTED=${IS_DIRECTED:=true}
# 是否进行 ID 编码
NEED_ENCODE=${NEED_ENCODE:=true}
# 数据源的点 ID 类型，例如：string、int32、int64。
VTYPE=${VTYPE:=int32}
# 编码类型。distributed 为分布式点ID编码，single 为单机点 ID 编码。
ENCODER=${ENCODER:="distributed"}
# PageRank 算法的参数。不同算法的参数不同。
EPS=${EPS:=0.0001}
DAMPING=${DAMPING:=0.85}
# 迭代次数
ITERATIONS=${ITERATIONS:=100}
```

3. 修改配置文件 `cluster`，设置执行算法的 Nebula Analytics 集群节点和任务分配权重。

```
# Nebula Analytics 集群节点 IP 地址:任务分配权重
192.168.8.200:1
192.168.8.201:1
192.168.8.202:1
```

4. 执行算法脚本。例如:

```
./run_pagerank.sh
```

5. 在输出路径查看计算结果。

- 输出至 Nebula Graph 集群，请根据 `nebula.conf` 的设置查看计算结果。
- 输出至 HDFS 上的 CSV 文件或本地 CSV 文件，请根据图计算脚本内的 `OUTPUT` 设置查看计算结果，计算结果为 `.gz` 格式的压缩文件。

最后更新: June 20, 2023

2022 vesoft Inc.

21.4 Nebula Explorer Workflow

Nebula Explorer 提供 Workflow 功能进行可视化图计算。

详情参见工作流简介。

Enterpriseonly

Explorer 仅在企业版提供。

最后更新: June 20, 2023

2022 vesoft Inc.

21.5 Dag Controller

Dag Controller 是一款任务编排调度工具，可以编排调度有向无环图（DAG）类型的作业，该作业由多个任务组成，且任务之间存在先后关系，组成一个有向无环图（DAG）。

Dag Controller 可以结合 Nebula Analytics 进行复杂的图计算。例如 Dag Controller 将算法调用请求发送给 Nebula Analytics，Nebula Analytics 保存结果至 Nebula Graph 或 HDFS，Dag Controller 再将上次的计算结果作为下一个算法任务的输入创建新的任务。

本文介绍如何使用 Dag Controller。

> **Enterpriseonly**
>
> 仅企业版支持本功能。

21.5.1 前提条件

- 已部署 2.2.x 或以上版本的 HDFS。
- 已安装 1.8 版本的 JDK。

21.5.2 准备工作

不同的环境安装包和命令略有区别，本文的准备工作如下：

- 操作系统为 CentOS 7。
- 如果 Nebula Analytics 和 Dag Controller 分别部署在多台机器，请确保机器网络互通。
- 如果 Nebula Analytics 是分布式架构，请确保路径、端口等配置相同。

21.5.3 注意事项

- BFS、SSSP 算法需要对参数 `root` 做校验，只支持一个上游组件，且必须要指定行和列。存在多个文件时，随机取一个，找不到行、列或者文件时直接报错。
- 相似度算法，不限制上游组件的格式，但是需要指定列。存在多个文件时，随机叠加然后取前 N 条数据进行处理，指定了行和列、或者指定列不存在的话直接报错。

21.5.4 部署 Nebula Analytics

1. 安装 libatomic、psmisc。

```
$ sudo yum -y install libatomic psmisc
```

2. 安装 Nebula Analytics。

```
$ sudo rpm -ivh <analytics_package_name> --prefix <install_path>
$ sudo chown <user>:<user> -R <install path>
```

例如：

```
$ sudo rpm -ivh nebula-analytics-3.2.0-centos.x86_64.rpm --prefix=/home/vesoft/nebula-analytics
$ sudo chown vesoft:vesoft -R /home/vesoft/nebula-analytics
```

3. 配置 `set_env.sh` 文件，路径为 `nebula-analytics/scripts/set_env.sh`。配置正确的 Hadoop 路径和 JDK 路径。如果有多台机器，请确保路径一致。

```
export HADOOP_HOME=<hadoop_path>
export JAVA_HOME=<java_path>
```


2022 vesoft Inc.

21.5.5 部署 Dag Controller

1. 配置 Dag Controller 机器 SSH 免密登录 Nebula Analytics 机器，以及 Nebula Analytics 集群内所有节点间的 SSH 相互免密登录。

   例如机器 A（Dag Controller）通过 SSH 免密登录至 Nebula Analytics 集群 B 中的机器 B-1。请在机器 A 上执行如下命令：

   ```
   //执行后按提示生成密钥，默认按回车即可。
   $ ssh-keygen -t rsa

   //将机器 A 的公钥文件安装到机器 B-1 对应的用户下，即可从机器 A 免密登录机器 B-1。
   $ ssh-copy-id -i ~/.ssh/id_rsa.pub <B_user>@<B_IP>
   ```

   按同样方法设置 A 免密登录机器 B-2、B-3 等，以及集群 B 内所有机器的互相免密登录。

2. 添加以下内容至 `~/.bash_profile` 文件内，执行 `source ~/.bash_profile` 使其生效。

   ```
   eval $(ssh-agent)
   ssh-add ~/.ssh/id_rsa
   ```

3. 安装 Dag Controller。

   ```
   $ sudo rpm -ivh <analytics_package_name> --prefix <install_path>
   $ sudo chown <user>:<user> -R <install path>
   ```

   例如：

   ```
   $ sudo rpm -ivh dag-ctrl-3.2.0-centos.x86_64.rpm --prefix=/home/vesoft/dag-ctrl
   $ sudo chown vesoft:vesoft -R /home/vesoft/dag-ctrl
   ```

4. 配置 `dag-ctrl-api.yaml` 文件，路径为 `dag-ctrl/etc/dag-ctrl-api.yaml` 。配置 Nebula Analytics 机器的用户名及端口，如果有多台机器，请确保使用相同用户名。

   ```
   # Nebula Analytics 机器的用户名以及 SSH 端口。
   SSH:
     UserName: vesoft
     Port: 22

   #任务和作业的并行线程池大小。
   JobPool:
     Sleep: 3    # 3 秒检查一次有没有未执行的作业。
     Size: 3    # 同时可以执行 3 个作业。
   TaskPool:
     CheckStatusSleep: 1    # 1 秒检查一次任务状态。
     Size: 10    #同时可以执行 10 个任务。

   Dag:
     VarDataListMaxSize: 100    # 如果读取 HDFS 的列，则限制为每次 100 条数据。
   ```

5. 配置 `tasks.yaml` 文件，路径为 `dag-ctrl/etc/tasks.yaml` 。配置算法文件的具体路径（`exec_file` 参数），如果有多台机器，请确保路径一致。

6. 启动 Dag Controller。

   ```
   $ cd <dag_ctrl_install_path>
   $ ./scripts/start.sh
   ```

7. 查看 Dag Controller 的端口状态，确认是否启动成功。默认端口为 `9002`，在 `dag-ctrl-api.yaml` 文件内设置。

   ```
   $ netstat -aon | grep 9002
   ```

21.5.6 下一步

Nebula Analytics 和 Dag Controller 都配置并启动成功后，在 Nebula Explorer 上进行资源配置后即可进行复杂图计算。详情参见资源配置。

21.5.7 常见问题

如果 **Graph** 服务返回的查询结果数据量过大，会导致 **Dag Controller** 服务崩溃吗?

Dag Controller 服务仅仅提供调度功能，不会崩溃，但是数据量过大可能会导致 Nebula Analytics 服务读写 HDFS 或者 Nebula Graph 时内存不足而崩溃。

2022 vesoft Inc.

如果一个作业中的某个任务失败，能否从失败的任务开始重新执行?

暂不支持，只能整体重新执行。

如果任务结果保存很慢，或任务间数据传输很慢，如何加速?

Dag Controller 包含图查询组件和图计算组件。图查询是发送请求给 Graph 进程进行查询，因此只能增大 Graph 服务的内存进行加速；图计算是由 Nebula Analytics 提供的分布式节点进行计算，可以增大计算集群规模进行加速。

HDFS 服务器无法连接时，任务状态一直为 **running** 怎么办?

为 HDFS 连接设置超时时间、次数，配置如下：

```
<configuration>
<property>
    <name>ipc.client.connect.timeout</name>
    <value>3000</value>
</property>

<property>
    <name>ipc.client.connect.max.retries.on.timeouts</name>
    <value>3</value>
</property>
</configuration>
```

任务运行失败，报错 **Err:dial unix: missing address** 怎么办?

修改 `dag-ctrl/etc/dag-ctrl-api.yaml` 配置文件，配置 SSH 的 `UserName` 。

任务运行失败，报错 **bash: /home/xxx/nebula-analytics/scripts/run_algo.sh: No such file or directory** 怎么办?

修改 `dag-ctrl/etc/tasks.yaml` 配置文件，配置算法执行路径 `exec_file` 。

任务运行失败，报错 **/lib64/libm.so.6: version 'GLIBC_2.29' not found (required by /home/vesoft/jdk-18.0.1/jre/lib/amd64/server/libjvm.so)** 怎么办?

由于 JDK18 版本太新，而操作系统版本太旧， `YUM` 无法下载 `GLIBC_2.29` ，可以安装 JDK1.8，请同步修改 `nebula-analytics/scripts/set_env.sh` 中的 JDK 地址。

任务运行失败，报错 **handshake failed: ssh: unable to authenticate, attempted methods [none publickey], no supported methods remain** 怎么办?

重新配置 `.ssh` 文件夹及 `.ssh/authorized_keys` 文件的权限， `.ssh` 文件夹权限为 `744` ， `.ssh/authorized_keys` 文件权限为 `600` 。

任务运行失败，报错 **There are 0 Nebula Analytics available. clusterSize should be less than or equal to it** 怎么办?

可能是因为如下原因：

- 未配置 Nebula Analytics。请按本文档配置 Nebula Analytics。
- 已配置 Nebula Analytics，但是无法与 Dag Controller 联通。例如 地址错误、未配置 SSH、两个服务的启动用户不一致（导致 SSH 登录失败）等。

任务运行失败，报错 **broadcast.hpp:193] Check failed: (size_t)recv_bytes >= sizeof(chunk_tail_t) recv message too small: 0** 怎么办?

任务要处理的数据量过小，但是配置的计算节点数与进程数太多。需要在提交作业时设置较小的 `clusterSize` 和 `processes` 。

最后更新: June 20, 2023

2022 vesoft Inc.

22. Nebula Spark Connector

Nebula Spark Connector 是一个 Spark 连接器，提供通过 Spark 标准形式读写 Nebula Graph 数据的能力。Nebula Spark Connector 由 Reader 和 Writer 两部分组成。

- Reader

  提供一个 Spark SQL 接口，用户可以使用该接口编程读取 Nebula Graph 图数据，单次读取一个点或 Edge type 的数据，并将读取的结果组装成 Spark 的 DataFrame。

- Writer

  提供一个 Spark SQL 接口，用户可以使用该接口编程将 DataFrame 格式的数据逐条或批量写入 Nebula Graph。

更多使用说明请参见 Nebula Spark Connector。

22.1 适用场景

Nebula Spark Connector 适用于以下场景：

- 在不同的 Nebula Graph 集群之间迁移数据。
- 在同一个 Nebula Graph 集群内不同图空间之间迁移数据。
- Nebula Graph 与其他数据源之间迁移数据。
- 结合 Nebula Algorithm 进行图计算。

22.2 特性

Nebula Spark Connector 3.0.0版本特性如下：

- 提供多种连接配置项，如超时时间、连接重试次数、执行重试次数等。
- 提供多种数据配置项，如写入数据时设置对应列为点 ID、起始点 ID、目的点 ID 或属性。
- Reader 支持无属性读取和全属性读取。
- Reader 支持将 Nebula Graph 数据读取成 Graphx 的 VertexRDD 和 EdgeRDD，支持非 Long 型点 ID。
- 统一了 SparkSQL 的扩展数据源，统一采用 DataSourceV2 进行 Nebula Graph 数据扩展。
- 支持 `insert` 、 `update` 和 `delete` 三种写入模式。 `insert` 模式会插入（覆盖）数据， `update` 模式仅会更新已存在的数据， `delete` 模式只删除数据。
- 支持与 Nebula Graph 之间的 SSL 加密连接。

22.3 更新说明

Release notes

2022 vesoft Inc.

22.4 获取 Nebula Spark Connector

22.4.1 编译打包

> **Note**
>
> 安装 Spark 2.4.x 版本。

1. 克隆仓库 `nebula-spark-connector` 。

```
$ git clone -b v3.0.0 https://github.com/vesoft-inc/nebula-spark-connector.git
```

2. 进入目录 `nebula-spark-connector` 。

```
$ cd nebula-spark-connector/nebula-spark-connector
```

3. 编译打包。

```
$ mvn clean package -Dmaven.test.skip=true -Dgpg.skip -Dmaven.javadoc.skip=true
```

编译完成后，在目录 `nebula-spark-connector/nebula-spark-connector/target/` 下生成类似文件 `nebula-spark-connector-3.0.0-SHANPSHOT.jar` 。

22.4.2 Maven 远程仓库下载

下载地址

22.5 使用方法

使用 Nebula Spark Connector 读写 Nebula Graph 数据库时，只需要编写以下代码即可实现。

```
# 从 Nebula Graph 读取点边数据。
spark.read.nebula().loadVerticesToDF()
spark.read.nebula().loadEdgesToDF()

# 将 dataframe 数据作为点和边写入 Nebula Graph 中。
dataframe.write.nebula().writeVertices()
dataframe.write.nebula().writeEdges()
```

`nebula()` 接收两个配置参数，包括连接配置和读写配置。

22.5.1 从 Nebula Graph 读取数据

```
val config = NebulaConnectionConfig
  .builder()
  .withMetaAddress("127.0.0.1:9559")
  .withConenctionRetry(2)
  .withExecuteRetry(2)
  .withTimeout(6000)
  .build()

val nebulaReadVertexConfig: ReadNebulaConfig = ReadNebulaConfig
  .builder()
  .withSpace("test")
  .withLabel("person")
  .withNoColumn(false)
  .withReturnCols(List("birthday"))
  .withLimit(10)
  .withPartitionNum(10)
  .build()
val vertex = spark.read.nebula(config, nebulaReadVertexConfig).loadVerticesToDF()

val nebulaReadEdgeConfig: ReadNebulaConfig = ReadNebulaConfig
  .builder()
  .withSpace("test")
  .withLabel("knows")
  .withNoColumn(false)
  .withReturnCols(List("degree"))
  .withLimit(10)
  .withPartitionNum(10)
```


2022 vesoft Inc.

```
  .build()
val edge = spark.read.nebula(config, nebulaReadEdgeConfig).loadEdgesToDF()
```

- `NebulaConnectionConfig` 是连接 Nebula Graph 的配置，说明如下。

参数	是否必须	说明
`withMetaAddress`	是	所有 Meta 服务的地址，多个地址用英文逗号（,）隔开，格式为 `ip1:port1,ip2:port2,...`。读取数据不需要配置 `withGraphAddress`。
`withConnectionRetry`	否	Nebula Java Client 连接 Nebula Graph 的重试次数。默认值为 `1`。
`withExecuteRetry`	否	Nebula Java Client 执行查询语句的重试次数。默认值为 `1`。
`withTimeout`	否	Nebula Java Client 请求响应的超时时间。默认值为 `6000`，单位：毫秒（ms）。

- `ReadNebulaConfig` 是读取 Nebula Graph 数据的配置，说明如下。

参数	是否必须	说明
`withSpace`	是	Nebula Graph 图空间名称。
`withLabel`	是	Nebula Graph 图空间内的 Tag 或 Edge type 名称。
`withNoColumn`	否	是否不读取属性。默认值为 `false`，表示读取属性。取值为 `true` 时，表示不读取属性，此时 `withReturnCols` 配置无效。
`withReturnCols`	否	配置要读取的点或边的属性集。格式为 `List(property1,property2,...)`，默认值为 `List()`，表示读取全部属性。
`withLimit`	否	配置 Nebula Java Storage Client 一次从服务端读取的数据行数。默认值为 1000。
`withPartitionNum`	否	配置读取 Nebula Graph 数据时 Spark 的分区数。默认值为 100。该值的配置最好不超过图空间的的分片数量（partition_num）。

22.5.2 向 Nebula Graph 写入数据

Note

DataFrame 中的列会自动作为属性写入 Nebula Graph。

```
val config = NebulaConnectionConfig
  .builder()
  .withMetaAddress("127.0.0.1:9559")
  .withGraphAddress("127.0.0.1:9669")
  .withConenctionRetry(2)
  .build()

val nebulaWriteVertexConfig: WriteNebulaVertexConfig = WriteNebulaVertexConfig
  .builder()
  .withSpace("test")
  .withTag("person")
  .withVidField("id")
  .withVidPolicy("hash")
  .withVidAsProp(true)
  .withUser("root")
  .withPasswd("nebula")
  .withBatch(512)
  .build()
df.write.nebula(config, nebulaWriteVertexConfig).writeVertices()

val nebulaWriteEdgeConfig: WriteNebulaEdgeConfig = WriteNebulaEdgeConfig
  .builder()
  .withSpace("test")
  .withEdge("friend")
  .withSrcIdField("src")
  .withSrcPolicy(null)
  .withDstIdField("dst")
  .withDstPolicy(null)
  .withRankField("degree")
  .withSrcAsProperty(true)
  .withDstAsProperty(true)
  .withRankAsProperty(true)
  .withUser("root")
```


2022 vesoft Inc.

```
    .withPasswd("nebula")
    .withBatch(512)
    .build()
df.write.nebula(config, nebulaWriteEdgeConfig).writeEdges()
```

默认写入模式为 `insert`，可以通过 `withWriteMode` 配置修改为 `update`：

```
val config = NebulaConnectionConfig
    .builder()
    .withMetaAddress("127.0.0.1:9559")
    .withGraphAddress("127.0.0.1:9669")
    .build()
val nebulaWriteVertexConfig = WriteNebulaVertexConfig
    .builder()
    .withSpace("test")
    .withTag("person")
    .withVidField("id")
    .withVidAsProp(true)
    .withBatch(512)
    .withWriteMode(WriteMode.UPDATE)
```


2022 vesoft Inc.

```
  .build()
df.write.nebula(config, nebulaWriteVertexConfig).writeVertices()
```

- `NebulaConnectionConfig` 是连接 Nebula Graph 的配置，说明如下。

参数	是否必须	说明
`withMetaAddress`	是	所有 Meta 服务的地址，多个地址用英文逗号（,）隔开，格式为 `ip1:port1,ip2:port2,...` 。
`withGraphAddress`	是	Graph 服务的地址，多个地址用英文逗号（,）隔开，格式为 `ip1:port1,ip2:port2,...` 。
`withConnectionRetry`	否	Nebula Java Client 连接 Nebula Graph 的重试次数。默认值为 `1` 。

- `WriteNebulaVertexConfig` 是写入点的配置，说明如下。

参数	是否必须	说明
`withSpace`	是	Nebula Graph 图空间名称。
`withTag`	是	写入点时需要关联的 Tag 名称。
`withVidField`	是	DataFrame 中作为点 ID 的列。
`withVidPolicy`	否	写入点 ID 时，采用的映射函数，Nebula Graph 仅支持 HASH。默认不做映射。
`withVidAsProp`	否	DataFrame 中作为点 ID 的列是否也作为属性写入。默认值为 `false` 。如果配置为 `true` ，请确保 Tag 中有和 `VidField` 相同的属性名。
`withUser`	否	Nebula Graph 用户名。若未开启身份验证，无需配置用户名和密码。
`withPasswd`	否	Nebula Graph 用户名对应的密码。
`withBatch`	是	一次写入的数据行数，默认值为 `512` 。当 `withWriteMode` 为 `update` 时，该参数的最大值为 `512` 。
`withWriteMode`	否	写入模式。可选值为 `insert` 和 `update` 。默认为 `insert` 。

2022 vesoft Inc.

- `WriteNebulaEdgeConfig` 是写入边的配置，说明如下。

参数	是否必须	说明
`withSpace`	是	Nebula Graph 图空间名称。
`withEdge`	是	写入边时需要关联的 Edge type 名称。
`withSrcIdField`	是	DataFrame 中作为起始点的列。
`withSrcPolicy`	否	写入起始点时，采用的映射函数，Nebula Graph 仅支持 HASH。默认不做映射。
`withDstIdField`	是	DataFrame 中作为目的点的列。
`withDstPolicy`	否	写入目的点时，采用的映射函数，Nebula Graph 仅支持 HASH。默认不做映射。
`withRankField`	否	DataFrame 中作为 rank 的列。默认不写入 rank。
`withSrcAsProperty`	否	DataFrame 中作为起始点的列是否也作为属性写入。默认值为 `false`。如果配置为 `true`，请确保 Edge type 中有和 `SrcIdField` 相同的属性名。
`withDstAsProperty`	否	DataFrame 中作为目的点的列是否也作为属性写入。默认值为 `false`。如果配置为 `true`，请确保 Edge type 中有和 `DstIdField` 相同的属性名。
`withRankAsProperty`	否	DataFrame 中作为 rank 的列是否也作为属性写入。默认值为 `false`。如果配置为 `true`，请确保 Edge type 中有和 `RankField` 相同的属性名。
`withUser`	否	Nebula Graph 用户名。若未开启身份验证，无需配置用户名和密码。
`withPasswd`	否	Nebula Graph 用户名对应的密码。
`withBatch`	是	一次写入的数据行数，默认值为 `512`。当 `withWriteMode` 为 `update` 时，该参数的最大值为 `512`。
`withWriteMode`	否	写入模式。可选值为 `insert` 和 `update`。默认为 `insert`。

22.5.3 示例代码

详细的使用方式参见 示例代码。

最后更新: June 20, 2023

2022 vesoft Inc.

23. Nebula Flink Connector

Nebula Flink Connector 是一款帮助 Flink 用户快速访问 Nebula Graph 的连接器，支持从 Nebula Graph 图数据库中读取数据，或者将其他外部数据源读取的数据写入 Nebula Graph 图数据库。

更多使用说明请参见 Nebula Flink Connector。

23.1 适用场景

Nebula Flink Connector 适用于以下场景：

- 在不同的 Nebula Graph 集群之间迁移数据。
- 在同一个 Nebula Graph 集群内不同图空间之间迁移数据。
- Nebula Graph 与其他数据源之间迁移数据。

23.2 更新说明

Release notes

最后更新: June 20, 2023

2022 vesoft Inc.

24. Nebula Bench

Nebula Bench 是一款利用 LDBC 数据集对 Nebula Graph 进行性能测试的工具。

24.1 适用场景

- 生成测试数据并导入 Nebula Graph。
- 对 Nebula Graph 集群进行性能测试。

24.2 更新说明

Release

详细使用说明请参见 Nebula Bench。

最后更新: June 20, 2023

2022 vesoft Inc.

25. 附录

25.1 Nebula Graph 3.1.0 release notes

25.1.1 优化

- 支持在表达式中使用模式，如：`MATCH (v:player) WHERE (v)-[:like]->() RETURN v`。#3997
- 支持用 `CLEAR SPACE` 清除图空间数据并保留 Schema 信息。#3989
- 支持在匹配模式中重复点的别名，如：`MATCH (v)-->(v)`。#3929
- 优化 `SUBGRAPH` 和 `FIND PATH` 从而提高性能。#3871 #4095
- 优化路径以减少冗余路径和时间复杂度。#4126
- 优化获取属性的方式进而优化 `MATCH` 语句的性能。#3750
- 优化 `GO` 、`YIELD` 子句，减少不必要的属性读取。#3974
- 支持获取属性时 Filter 及 `LIMIT` 下推。 #3844 #3839
- `maxHop` 在匹配可变长度路径中是可选的。#3881
- 使用 `DROP SPACE` 之后图空间将进行物理删除。#3913
- 优化日期时间/日期/时间中数字解析。#3797
- 添加 `toSet` 函数，将 `LIST` 或 `SET` 转换为 `SET`。#3594
- 支持使用 nGQL 来显示服务的 HTTP 端口，并禁用 HTTP2 端口。#3808
- 限制单用户、单机器连接数据库的会话数量。#3729
- 优化存储启动时的等待机制，保证与 Meta 服务及时连接。#3971
- 当节点存在多条路径，某条数据路径对应磁盘故障时，不再需要重建整个节点。#4131
- 优化 Job 管理。#3976 #4045 #4001
- 支持在作业管理中管理 `DOWNLOAD` 、`INGEST SST` 文件。#3994
- 支持显示失败作业的错误码。#4067
- 支持禁用 OS 页面缓存，只在共享环境中使用块缓存和 Nebula 存储缓存，以避免应用程序之间的内存占用干扰。#3890
- 更新 KV 分离阈值的默认值（从 0 到 100）。#3879
- 支持 gflag 设置表达式深度上限，方便调整适配不同的机器环境。#3722
- 新增 `KILL QUERY` 的权限检查。当启用身份验证时，具有 GOD 角色的用户可以终止所有查询，而具有其他角色的用户只能终止自己的查询。#3896
- 新增 distcc、sccache 等编译方式的支持。#3896
- meta dump 工具支持更多可 dump 的表。#3870
- 存储层将写操作（`INSERT VERTEX` 或者 `INSERT EDGE`）的并发控制，从报错并要求客户端重试，改为内部排队，以便客户端更简单适配。#3926

25.1.2 缺陷修复

- 修复 `LOOKUP` 中使用函数调用作为过滤器的一部分导致的服务崩溃问题。#4111
- 修复 `IN` 表达式中的属性没有索引绑定时的崩溃问题。#3986
- 修复并发扫描点或者边时导致 Storage 服务崩溃的问题。#4190
- 修复 `MATCH` 语句的聚合子句中，使用模式表达式时崩溃的问题。#4180
- 修复获取 `profile` 查询的 JSON 结果导致的崩溃问题。#3998
- 修复 Lambda 函数中的 `async` 接口运行完毕且 `threadManager` 中的任务未执行时的崩溃问题。#4000
- 修复 `GROUP BY` 输出的缺陷。#4128
- 修复 `SHOW HOSTS` 有时不能显示版本的缺陷。#4116

2022 vesoft Inc.

- 修复 `id(n) == $var`， `id(n) IN [$var]`， `id(n) == $var.foo.bar`， `id(n) IN $var.foo.bar` 参数化的缺陷。#4024
- 修复 `MATCH...WHERE` 中出现错误路径方向的缺陷。#4091
- 修复 `WHERE` 子句同时引用多 `MATCH` 变量结果显示不正确的缺陷。 #4143
- 修复优化规则的缺陷。#4146
- 修复节点处理 Raft 快照失败的缺陷。#4019
- 修复节点接收快照后无法接受更多日志的缺陷。#3909
- 修复快照中不包含不带 Tag 的点的缺陷。#4189
- 修复同一个 Tag 或者 Edge 的版本超过 255 后读取最新版本的 Schema 失败的缺陷。#4023
- 修复 `SHOW STATS` 不统计不带 Tag 的点的缺陷。#3967
- 修复有时时间戳获取错误的缺陷。#3958
- 修复可以为 `root` 用户分配图空间中的其他角色的缺陷。#3868
- 修复词法分析器中的列索引重复计数的缺陷。#3626

25.1.3 历史版本

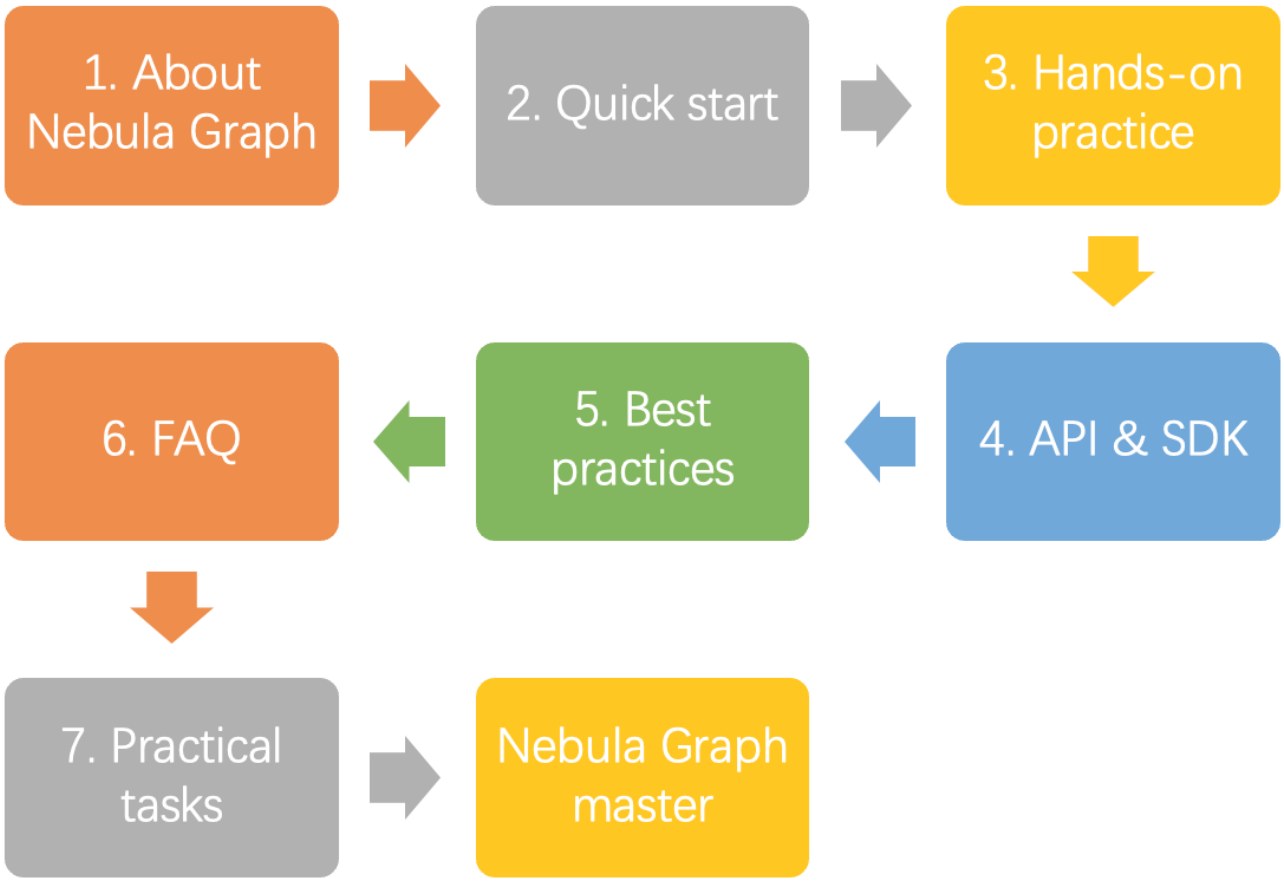
历史版本

最后更新: June 20, 2023

2022 vesoft Inc.

25.2 Nebula Graph 学习路径

本文介绍 Nebula Graph 学习路径，用户可以通过路径中的文档及视频由浅入深地学习图数据库 Nebula Graph。



25.2.1 1. 关于 Nebula Graph

1.1 什么是 **Nebula Graph**?

文档	视频	**PPT**
什么是 Nebula Graph	Nebula Graph 介绍、万亿级别的图数据库 Nebula Graph	Nebula Graph 3年回顾

1.2 图相关术语

视频
图世界的那些概念、术语
图数据库简述

1.3 数据模型

文档
数据模型

2022 vesoft Inc.

1.4 路径

文档	视频
路径	路径类型

1.5 产品架构

文档	视频
Meta 服务	-
Graph 服务	-
Storage 服务	-

25.2.2 2. 快速入门

2.1 安装 **Nebula Graph**

文档	视频
使用 RPM/DEB 包	-
使用 TAR 包	-
使用 Docker	-
使用源码	图数据库入门教程（零）通过编译源码来安装 Nebula Graph
-	如何选择部署方式?

2.2 启动 **Nebula Graph**

文档
启停 Nebula Graph

2.3 连接 **Nebula Graph**

文档
连接 Nebula Graph

2.4 使用 **nGQL** 命令

文档
nGQL 命令汇总

25.2.3 3. 进阶操作

3.1 部署多机集群

文档
使用 RPM/DEB 包部署 Nebula Graph 多机集群

2022 vesoft Inc.

3.2 升级集群版本

文档

升级 Nebula Graph

3.3 配置Nebula

文档

配置 Meta

配置 Graph

配置 Storage

配置 Linux 内核

3.4 配置日志

文档

日志配置

3.5 运维与管理

- 账号鉴权和授权

  文档

  本地身份验证

  OpenLDAP

  管理用户

  内置角色

- 平衡分片分布

  文档

  Storage 负载均衡

- 监控

  文档

  Nebula 指标

  RocksDB 统计数据

- 数据快照

  文档

  创建快照

- SSL 加密

  文档

  SSL 加密

2022 vesoft Inc.

3.6 性能调优

文档	视频
图建模	Nebula 高性能图 schema 设计
系统建模	-
Compaction 策略	-

3.7 周边工具

- 云

Cloud 版本	文档	视频

2022 vesoft Inc.

|阿里云版|Nebula Graph Cloud 阿里云版|Nebula Graph Cloud 阿里云版介绍|

- 可视化

可视化工具	文档	视频
数据可视化	Nebula Studio	Nebula Studio 图探索功能和Nebula Studio 可视化建模
数据监控和运维	Nebula Dashboard 企业版和 Nebula Dashboard 社区版	可视化监控 Nebula Dashboard
数据分析	Nebula Explorer 企业版	可视化图探索 Nebula Explorer

- 数据导入与导出

导入与导出	文档	视频
数据导入	Nebula Importer	Nebula Importer
数据导入	Nebula Spark Connector	-
数据导入	Nebula Flink Connector	-
数据导入	Nebula Exchange 社区版	Nebula Graph 数据导入工具——Exchange、Exchange 导入 SST 数据
数据导出	Nebula Exchange 企业版	-

- 备份与恢复

文档	视频
Nebula BR	Nebula Graph 容灾备份工具 nebula-br

- 性能测试

文档
Nebula Bench

- 集群运维

文档	视频
Nebula Operator	Nebula Operator、 云原生的演进

- 图算法

文档	视频
Nebula Algorithm	Nebula Algorithm 介绍

- 客户端

文档
Nebula Console
Nebula CPP
Nebula Java
Nebula Python
Nebula Go

2022 vesoft Inc.

25.2.4 4. 高阶操作

文档

API & SDK

25.2.5 5. 最佳实践

文档 **&** 视频

LDBC 数据导入及 nGQL 实践

基于 Nebula Graph 的 Betweenness Centrality 算法

百亿级图数据在快手安全情报的应用与挑战

美团图数据库平台建设及业务实践

Akulaku 的智能风控实践

微信使用 Nebula Graph 的定制化开发

知乎使用 Nebula Graph 实践

Nebula Graph 在微众银行的数据治理业务的实践

图数据库在安全风控场景的应用 @BOSS 直聘

更多文档 、视频

25.2.6 6. 常见问题

文档

常见问题 FAQ

25.2.7 7. 实操作业

用户可以通过完成以下实操作业来检测是否玩转 Nebula Graph。

作业内容	参考
编译内核	使用源码安装 Nebula Graph
部署可视化工具 Studio、Dashboard、Explorer	部署 Studio、部署 Dashboard、部署 Explorer
使用 K6 压测 Nebula Graph	K6 在 Nebula Graph 上的压测实践
导入 LDBC 数据	LDBC 数据导入及 nGQL 实践
查询 LDBC 数据（点查、K度邻居查询、路径查询、子图查询）	LDBC 和 interactive-short-1.cypher

25.2.8 8. 通过 CI/CP 考试

图数据库 Nebula Graph 提供2个不同等级的图数据库技能认证考试：

- Nebula Graph Certified Insider (NGCI)：通过该考试证明用户对图数据库及 Nebula Graph 有基础的了解，是合格的 Nebula Graph 使用者。
- Nebula Graph Certified Professional (NGCP)：通过该考试证明用户对图数据库及 Nebula Graph 有深入的了解，是 Nebula Graph 专业人士。

考试用书《开源分布式图数据库 Nebula Graph 完全指南》

2022 vesoft Inc.

单击Nebula Graph 个人技能认证查看考试说明及入口。

最后更新: June 20, 2023

2022 vesoft Inc.

25.3 常见问题 FAQ

本文列出了使用 Nebula Graph 3.1.0 时可能遇到的常见问题，用户可以使用文档中心或者浏览器的搜索功能查找相应问题。

如果按照文中的建议无法解决问题，请到 Nebula Graph 论坛 提问或提交 GitHub issue。

25.3.1 关于本手册

为什么手册示例和系统行为不一致?

Nebula Graph 一直在持续开发，功能或操作的行为可能会有变化，如果发现不一致，请提交 issue 通知 Nebula Graph 团队。

> **Note**
>
> 如果发现本文档中的错误:

1. 用户可以点击页面顶部右上角的"铅笔"图标进入编辑页面。
2. 使用 Markdown 修改文档。完成后点击页面底部的 "Commit changes"，这会触发一个 GitHub pull request。
3. 完成 CLA 签署，并且至少 2 位 reviewer 审核通过即可合并。

25.3.2 关于历史兼容性

> **版本兼容性**
>
> Nebula Graph 3.1.0 与 历史版本 （包括 Nebula Graph 1.x 和 2.x) 的数据格式、客户端通信协议均双向不兼容。 使用老版本客户端连接新版本服务端，会导致服务进程退出。 数据格式升级参见升级 Nebula Graph 历史版本至当前版本。 客户端与工具均需要下载对应版本。

25.3.3 关于执行报错

如何处理错误信息 `SemanticError: Missing yield clause.`

从 Nebula Graph 3.0.0 开始，查询语句 `LOOKUP` 、`GO` 、`FETCH` 必须用 `YIELD` 子句指定输出结果。详情请参见YIELD。

如何处理错误信息 `Host not enough!`

从 3.0.0 版本开始，在配置文件中添加的 Storage 节点无法直接读写，配置文件的作用仅仅是将 Storage 节点注册至 Meta 服务中。必须使用 `ADD HOSTS` 命令后，才能正常读写 Storage 节点。详情参见管理 Storage 主机。

如何处理错误信息 `To get the property of the vertex in 'v.age', should use the format 'var.tag.prop'`

从 3.0.0 版本开始， `pattern` 支持同时匹配多个 Tag，所以返回属性时，需要额外指定 Tag 名称。即从 `RETURN 变量名.属性名` 改为 `RETURN 变量名.Tag名.属性名` 。

2022 vesoft Inc.

如何处理错误信息 `Storage Error E_RPC_FAILURE`

报错原因通常为 Graph 服务向 Storage 服务请求了过多的数据，导致 Storage 服务超时。请尝试以下解决方案：

- 修改配置文件：在 `nebula-graphd.conf` 文件中修改 `--storage_client_timeout_ms` 参数的值，以增加 Storage client 的连接超时时间。该值的单位为毫秒（ms）。例如，设置 `--storage_client_timeout_ms=60000` 。如果 `nebula-graphd.conf` 文件中未配置该参数，请手动增加。提示：请在配置文件开头添加--local_config=true 再重启服务。
- 优化查询语句：减少全库扫描型的查询，无论是否用 `LIMIT` 限制了返回结果的数量；用 GO 语句改写 MATCH 语句（前者有优化，后者无优化）。
- 检查 Storaged 是否发生过 OOM。(`dmesg |grep nebula`)。
- 为 Storage 服务器提供性能更好的 SSD 或者内存。
- 重试请求。

如何处理错误信息 `The leader has changed. Try again later`

已知问题，通常需要重试 1-N 次 (N==partition 数量）。原因为 meta client 更新 leader 缓存需要 1-2 个心跳或者通过错误触发强制更新。

编译 **Exchange**、**Connectors**、**Algorithm** 时无法下载 **SNAPSHOT** 包

现象：编译时提示 `Could not find artifact com.vesoft:client:jar:xxx-SNAPSHOT` 。

原因：本地 maven 没有配置用于下载 SNAPSHOT 的仓库。maven 中默认的 central 仓库用于存放正式发布版本，而不是开发版本（SNAPSHOT）。

解决方案：在 maven 的 setting.xml文件的 `profiles` 作用域内中增加以下配置：

```
<profile>
   <activation>
      <activeByDefault>true</activeByDefault>
   </activation>
   <repositories>
      <repository>
          <id>snapshots</id>
          <url>https://oss.sonatype.org/content/repositories/snapshots/</url>
          <snapshots>
             <enabled>true</enabled>
          </snapshots>
    </repository>
   </repositories>
</profile>
```

如何处理错误信息 `[ERROR (-1004)]: SyntaxError: syntax error near`？

大部分情况下，查询语句需要有 `YIELD` 或 `RETURN` ，请检查查询语句是否包含。

如何处理错误信息 `can't solve the start vids from the sentence`

查询引擎需要知道从哪些 VID 开始图遍历。这些开始图遍历的 VID，或者通过用户指定，例如：

```
> GO FROM ${vids} ...
> MATCH (src) WHERE id(src) == ${vids}
# 开始图遍历的 VID 通过如上办法指定
```

或者通过一个属性索引来得到，例如：

```
# CREATE TAG INDEX IF NOT EXISTS i_player ON player(name(20));
# REBUILD TAG INDEX i_player;

> LOOKUP ON player WHERE player.name == "abc" | ... YIELD ...
> MATCH (src) WHERE src.name == "abc" ...
# 通过点属性 name 的索引，来得到 VID
```

否则，就会抛出这样一个异常 `can't solve the start vids from the sentence` 。

2022 vesoft Inc.

如何处理错误信息 `Wrong vertex id type: 1001`

检查输入的 VID 类型是否是 `create space` 设置的 `INT64` 或 `FIXED_STRING(N)`。详情请参见 create space。

如何处理错误信息 `The VID must be a 64-bit integer or a string fitting space vertex id length limit.`

检查输入的 VID 是否超过限制长度。详情请参见 create space。

如何处理错误信息 `edge conflict` 或 `vertex conflict`

Storage 服务在毫秒级时间内多次收到插入或者更新同一点或边的请求时，可能返回该错误。请稍后重试。

如何处理错误信息 `RPC failure in MetaClient: Connection refused`

报错原因通常为 metad 服务状态异常，或是 metad 和 graphd 服务所在机器网络不通。请尝试以下解决方案：

- 在 metad 所在服务器查看下 metad 服务状态，如果服务状态异常，可以重新启动 metad 服务。
- 在报错服务器下使用 `telnet meta-ip:port` 查看网络状态。
- 检查配置文件中的端口配置，如果端口号与连接时使用的不同，改用配置文件中的端口或者修改配置。

如何处理 `nebula-graph.INFO` 中错误日志 `StorageClientBase.inl:214] Request to "x.x.x.x":9779 failed: N6apache6thrift9transport19TTransportExceptionE: Timed Out`

报错原因可能是查询的数据量比较大，storaged 处理超时。请尝试以下解决方法：

- 导入数据时，手动 compaction，加速读的速度。
- 增加 Graph 服务与 Storage 服务的 RPC 连接超时时间，在 `nebula-storaged.conf` 文件里面修改 `--storage_client_timeout_ms` 参数的值。该值的单位为毫秒（ms），默认值为 60000 毫秒。

如何处理 `nebula-storaged.INFO` 中错误日志 `MetaClient.cpp:65] Heartbeat failed, status:Wrong cluster!` 或者 `nebula-metad.INFO` 含有错误日志 `HBProcessor.cpp:54] Reject wrong cluster host "x.x.x.x":9771!`

报错的原因可能是用户修改了 metad 的 ip 或者端口信息，或者 storage 之前加入过其他集群。请尝试以下解决方法：

用户到 storage 部署的机器所在的安装目录（默认安装目录为 `/usr/local/nebula`）下面将 `cluster.id` 文件删除，然后重启 storaged 服务。

25.3.4 关于设计与功能

返回消息中 `time spent` 的含义是什么？

将命令 `SHOW SPACES` 返回的消息作为示例：

```
nebula> SHOW SPACES;
+--------------------+
| Name               |
+--------------------+
| "basketballplayer" |
+--------------------+
Got 1 rows (time spent 1235/1934 us)
```

- 第一个数字 `1235` 表示数据库本身执行该命令花费的时间，即查询引擎从客户端接收到一个查询，然后从存储服务器获取数据并执行一系列计算所花费的时间。
- 第二个数字 `1934` 表示从客户端角度看所花费的时间，即从客户端发送请求、接收结果，然后在屏幕上显示结果所花费的时间。

为什么在正常连接 **Nebula Graph** 后，`nebula-storaged` 进程的端口号一直显示红色？

`nebula-storaged` 进程的端口号的红色闪烁状态是因为 `nebula-storaged` 在启动流程中会等待 `nebula-metad` 添加当前 Storage 服务，当前 Storage 服务收到 Ready 信号后才会正式启动服务。从 3.0.0 版本开始，Meta 服务无法直接读写在配置文件中添加的 Storage 服务，配置文件的作用仅仅是将

2022 vesoft Inc.

Storage 服务注册至 Meta 服务中。用户必须使用 `ADD HOSTS` 命令后，才能使 Meta 服务正常读写 Storage 服务。更多信息，参见管理 Storage 主机。

为什么 **Nebula Graph** 的返回结果每行之间没有横线分隔了?

这是 Nebula Console 2.6.0 版本的变动造成的，不是 Nebula Graph 内核的变更，不影响返回数据本身的内容。

关于悬挂边

悬挂边 (Dangling edge) 是指一条边的起点或者终点在数据库中不存在。

Nebula Graph 3.1.0 的数据模型中，由于设计允许图中存在“悬挂边”； 没有 openCypher 中的 MERGE 语句。 对于悬挂边的保证完全依赖应用层面。 详见 INSERT VERTEX, DELETE VERTEX, INSERT EDGE, DELETE EDGE。

可以在 **`CREATE SPACE`** 时设置 **`replica_factor`** 为偶数（例如设置为 **2**）吗?

不要这样设置。

Storage 服务使用 Raft 协议（多数表决），为保证可用性，要求出故障的副本数量不能达到一半。

当机器数量为 1 时， `replica_factor` 只能设置为 `1`。

当机器数量足够时，如果 `replica_factor=2`，当其中一个副本故障时，就会导致系统无法正常工作；如果 `replica_factor=4`，只能有一个副本可以出现故障，这和 `replica_factor=3` 是一样。以此类推，所以 `replica_factor` 设置为奇数即可。

建议在生产环境中设置 `replica_factor=3`，测试环境中设置 `replica_factor=1`，不要使用偶数。

是否支持停止或者中断慢查询

支持。

详情请参见终止查询。

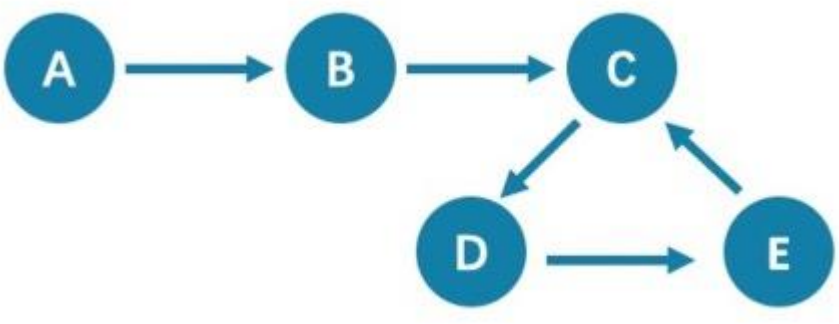
使用 **`GO`** 和 **`MATCH`** 执行相同语义的查询，查询结果为什么不同?

原因可能有以下几种：

- `GO` 查询到了悬挂边。
- `RETURN` 命令未指定排序方式。
- 触发了 Storage 服务中 `max_edge_returned_per_vertex` 定义的稠密点截断限制。
- 路径的类型不同，导致查询结果可能会不同。
- `GO` 语句采用的是 `walk` 类型，遍历时点和边可以重复。
- `MATCH` 语句兼容 openCypher，采用的是 `trail` 类型，遍历时只有点可以重复，边不可以重复。

因路径类型不同导致查询结果不同的示例图和说明如下。

2022 vesoft Inc.



从点 A 开始查询距离 5 跳的点，都会查询到点 C（`A->B->C->D->E->C`），查询 6 跳的点时，`GO` 语句会查询到点 D（`A->B->C->D->E->C->D`），因为边 `C->D` 可以重复查询，而 `MATCH` 语句查询为空，因为边不可以重复。

所以使用 `GO` 和 `MATCH` 执行相同语义的查询，可能会出现 `MATCH` 语句的查询结果比 `GO` 语句少。

关于路径的详细说明，请参见维基百科。

如何统计每种 **Tag** 有多少个点，每个 **Edge type** 有多少条边?

请参见 show-stats。

如何获取每种 **Tag** 的所有点，或者每种 **Edge type** 的所有边?

1. 建立并重建索引。

```
> CREATE TAG INDEX IF NOT EXISTS i_player ON player();
> REBUILD TAG INDEX i_player;
```

2. 使用 `LOOKUP` 或 `MATCH` 语句。例如:

```
> LOOKUP ON player;
> MATCH (n:player) RETURN n;
```

更多详情请参见 INDEX、LOOKUP 和 MATCH。

如何在不指定 **Tag/EdgeType** 的情况下，获取所有的点和边?

nGQL 没有该功能。

你必须先指定 Tag/EdgeType，或者用 `LIMIT` 子句限制返回数量，才能获取对应类型的所有的点和边。

例如执行 `MATCH (n) RETURN (n)`. 会返回错误 `Scan vertices or edges need to specify a limit number, or limit number can not push down.` 。

一个办法是使用 Nebula Algorithm.

或者指定各 Tag/Edge Type，然后再自己通过 `Union` 拼装。

能不能用中文字符做标识符，比如图空间、**Tag**、**Edge type**、属性、索引的名称?

能，详情参见关键字和保留字。

获取指定点的出度（或者入度）**?**

一个点的“出度”是指从该点出发的“边”的条数。入度，是指指向该点的边的条数。

```
nebula > MATCH (s)-[e]->() WHERE id(s) == "given" RETURN count(e); #出度
nebula > MATCH (s)<-[e]-() WHERE id(s) == "given" RETURN count(e); #入度
```


2022 vesoft Inc.

因为没有对出度和入度的进行特殊处理（例如索引或者缓存），这是一个较慢的操作。特别当遇到超级节点时，可能会发生OOM。

是否有办法快速获取“所有”点的出度和入度?

没有直接命令。

可以使用 Nebula Algorithm。

25.3.5 关于运维

运行日志文件过大时如何回收日志?

Nebula Graph 的运行日志默认在 `/usr/local/nebula/logs/` 下，正常 INFO 级别日志文件为 `nebula-graphd.INFO`, `nebula-storaged.INFO`, `nebula-metad.INFO`，报警和错误级别后缀为 `.WARNING` 和 `.ERROR`。

Nebula Graph 使用 glog 打印日志。glog 没有日志回收的功能，用户可以：

- 使用 crontab 设置定期任务回收日志文件，详情请参见 Glog should delete old log files automatically。
- 使用 logrotate 实现日志轮询。使用 logrotate 管理日志前需修改相应 Nebula Graph 服务的配置，将 `timestamp_in_logfile_name` 参数的值改成 `false`。

如何查看 **Nebula Graph** 版本

服务运行时：`nebula-console` 中执行命令 `SHOW HOSTS META`，详见 SHOW HOSTS

服务未运行时：在安装路径的 `bin` 目录内，执行 `./<binary_name> --version` 命令，可以查看到 version 和 GitHub 上的 commit ID，例如：

```
$ ./nebula-graphd --version
```

- Docker Compose 部署

  查看 Docker Compose 部署的 Nebula Graph 版本，方式和编译安装类似，只是要先进入容器内部，示例命令如下：

  ```
  docker exec -it nebula-docker-compose_graphd_1 bash
  cd bin/
  ./nebula-graphd --version
  ```

- RPM/DEB 包安装

  执行 `rpm -qa |grep nebula` 即可查看版本。

如何扩缩容（仅限企业版）

- 使用 Dashboard（企业版），在可视化页面对 graphd 和 storaged 进行快速扩缩容，详情参见集群操作-扩缩容。
- 使用 Nebula Operator 扩缩容集群，详情参见使用 Kubectl 部署 Nebula Graph 集群和使用 Helm 部署 Nebula Graph 集群。

2022 vesoft Inc.

Nebula Graph 3.1.0 未提供运维命令以实现自动扩缩容，参考以下步骤：

- metad 的扩容和缩容： metad 不支持自动扩缩容。

  Note

  用户可以使用脚本工具 迁移 meta 服务，但是需要自行修改 Graph 服务和 Storage 服务的配置文件中的 Meta 设置。

- graphd 的缩容：将该 graphd 的 ip 从 client 的代码中移除，关闭该 graphd 进程。
- graphd 的扩容：在新机器上准备 graphd 二进制文件和配置文件，在配置文件中修改或增加已在运行的 metad 地址，启动 graphd 进程。
- storaged 的缩容：详情参见缩容命令。完成后关闭 storaged 进程。
- storaged 的扩容：在新机器上准备 storaged 二进制文件和配置文件，在配置文件中修改或增加已在运行的 metad 地址，然后注册 storaged 到 metad 并启动 storaged 进程。详情参见注册 Storage 服务。

  storaged 扩缩容之后，还需要运行 Balance Data 和 Balance Leader 命令。

修改 **Host** 名称后，旧的 **Host** 一直显示 `OFFLINE` 怎么办?

`OFFLINE` 状态的 Host 将在一天后自动删除。

25.3.6 关于连接

防火墙中需要开放哪些端口?

如果没有修改过配置文件 中预设的端口，请在防火墙中开放如下端口：

服务类型	端口
Meta	9559, 9560, 19559, 19560
Graph	9669, 19669, 19670
Storage	9777 ~ 9780, 19779, 19780

如果修改过配置文件中预设的端口，请找出实际使用的端口并在防火墙中开放它们。

周边工具各自使用不用的端口，请参考各工具文档。

如何测试端口是否已开放?

用户可以使用如下 telnet 命令检查端口状态：

```
telnet <ip> <port>
```

Note

如果无法使用 telnet 命令，请先检查主机中是否安装并启动了 telnet。

示例：

```
// 如果端口已开放：
$ telnet 192.168.1.10 9669
Trying 192.168.1.10...
Connected to 192.168.1.10.
Escape character is '^]'.

// 如果端口未开放：
$ telnet 192.168.1.10 9777
```


2022 vesoft Inc.

```
Trying 192.168.1.10...
telnet: connect to address 192.168.1.10: Connection refused
```

25.3.7 关于 License

Dashboard/Explorer/Nebula Graph 企业版的 **License** 一样吗?

不一样，Dashboard、Explorer、Nebula Graph 企业版的 License 相互独立，不可互用。

在 **Nebula Graph** 企业版的 **License** 使用有效期期间，将企业版 **Meta** 替换成社区版 **Meta** 后，可以和企业版的 **Graph**、**Storage**一起使用吗?

不可以，不支持企业版和社区版的 Nebula Graph 的混合部署。

Nebula Graph 企业版的 **License** 到期后，是否可以将`data`目录的数据替换至社区版的 **Nebula Graph** 中使用?

可以。企业版和社区版的数据可以通用。只有在都是社区版部署的服务中才可以正常使用替换后的数据。不支持企业版和社区版的混合部署，例如，不支持混合部署企业版的 Meta 服务和社区版的 Graph、Storage 服务。

License 过期前，是否有信息提示? **License** 过期后如何续期?

License 过期前，系统会发送过期提醒信息。

正式版的 License 和试用版的 License 的过期前的提醒时间不同。

- 正式版的 License 过期提醒:
- 过期前 30 天和过期当天，在启动服务时有过期提醒。
- 到期后有 14 天缓冲期。在缓冲期期间，有过期提示，用户能继续使用 Nebula Graph/Dashboard/Explorer。缓冲期结束后，服务会停机并无法启动。

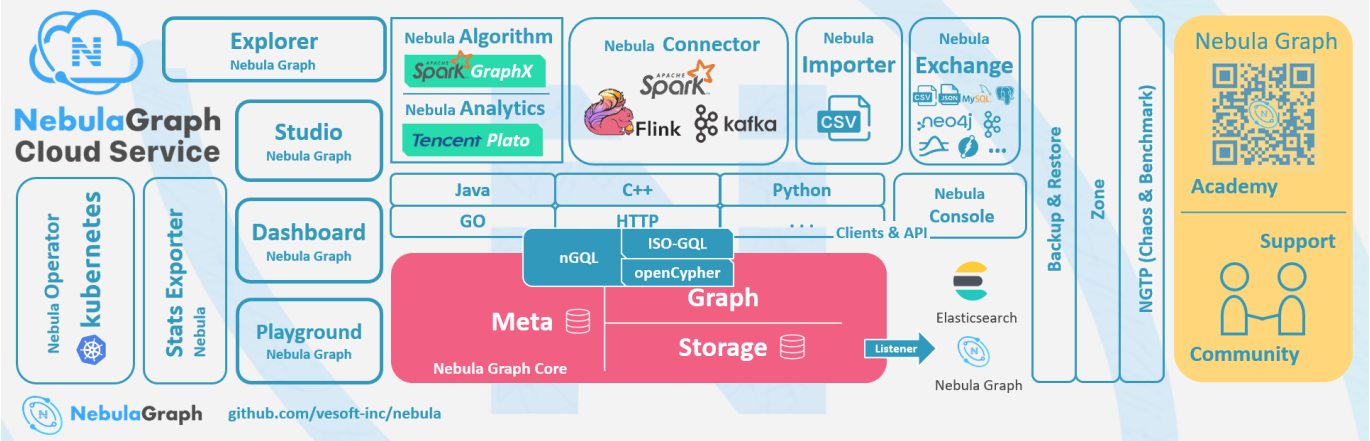
- 试用版的 License 过期提醒:
- 过期前 7 天和过期当天，在启动服务时有过期提醒。
- 无缓冲期。License 到期后，无法启动服务。

License 过期后，请及时联系销售（inqury@vesoft.com）更换新的证书。

最后更新: June 20, 2023

2022 vesoft Inc.

25.4 生态工具概览



Note

内核版本号命名规则为 `X.Y.Z`，表示 `大版本 X，中版本 Y，小版本 Z`。关于生命周期说明：

- 企业版：一个 `Y` 版本的开发周期为 6 个月，其维护与支持周期为 3 年；
- 社区版：一个 `Y` 版本的开发周期为 3 个月，其维护与支持周期至下一个 `Y` 版本发布；
- 文件命名中含有 `RC` 表示仅用于预览的非正式版本（`Release Candidate`），其维护周期仅至下个 RC 或者正式版本发布，其客户端、数据兼容性等均无保证。
- 文件命名中含有 `nightly` 、 `SNAPSHOT` 或者日期的为日常开发版本，无质量保证和维护周期。
- 版本号不一定连续，其数字大小仅代表偏序关系。

Compatibility

对于客户端的升级要求为：

- 内核从 `X.Y.Z1` 升级成 `X.Y.Z2`：表示内核完全前向兼容，通常用于 bugfix，建议尽快升级内核小版本。此时客户端可以不升级；
- 内核从 `X.Y1.*` 升级成 `X.Y2.*`：表示存在 API、语法、返回值部分不兼容，通常用于增加功能、提升性能、代码优化等。需要客户端相应升级至 `X.Y2.*`；
- 内核从 `X1.*.*` 升级成 `X2.*.*`：表示存储格式、API 、语法等存在大的不兼容。需要使用工具升级内核数据；客户端必须升级；
- 默认内核及客户端都不支持降级：不可从 `X.Y.Z2` 降级成 `X.Y.Z1`；
- 2.x 版本的生态工具完全不支持在 Nebula Graph 3.x 版本中使用。

25.4.1 Nebula Studio

Nebula Studio（简称 Studio）是一款可以通过 Web 访问的图数据库可视化工具，搭配 Nebula Graph DBMS 使用，提供构图、数据导入、编写 nGQL 查询、图探索等一站式服务。详情请参见什么是 Nebula Studio。

Note

Studio 版本发布节奏独立于 Nebula Graph 内核，其命名方式也不参照内核命名规则，两者兼容对应关系如下表。

Nebula Graph 版本	**Studio** 版本
v3.1.0	v3.3.0

2022 vesoft Inc.

25.4.2 Nebula Dashboard（社区版）

Nebula Dashboard（简称 Dashboard）是一款用于监控 Nebula Graph 集群中机器和服务状态的可视化工具。详情参见什么是 Nebula Dashboard。

Nebula Graph 版本	**Dashboard** 社区版本
v3.1.0	v1.1.2

25.4.3 Nebula Dashboard（企业版）

Nebula Dashboard（简称 Dashboard）是一款用于监控和管理 Nebula Graph 多集群中机器和服务状态的可视化工具，支持在可视化界面进行集群创建、集群导入、数据平衡、扩容缩容等操作。详情参见什么是 Nebula Dashboard。

Nebula Graph 版本	**Dashboard** 企业版本
v3.1.0	v3.1.0

25.4.4 Nebula Explorer

Nebula Explorer（简称 Explorer）是一款可以通过 Web 访问的图探索可视化工具，搭配 Nebula Graph 内核使用，用于与图数据进行可视化交互。即使没有图数据操作经验，用户也可以快速成为图专家。详情参见什么是 Nebula Explorer。

Nebula Graph 版本	**Explorer** 企业版本
v3.1.0	v3.1.0

25.4.5 Nebula Stats Exporter

nebula-stats-exporter将监控数据导入Prometheus。

Nebula Graph 版本	**Stats Exporter** 版本
v3.1.0	v3.1.0

25.4.6 Nebula Exchange

Nebula Exchange（简称 Exchange）是一款 Apache Spark™ 应用，用于在分布式环境中将集群中的数据批量迁移到 Nebula Graph 中，能支持多种不同格式的批式数据和流式数据的迁移。详情请参见什么是 Nebula Exchange。

Nebula Graph 版本	**Exchange** 版本	**Exchange** 企业版本
v3.1.0	v3.0.0	v3.0.0

25.4.7 Nebula Operator

Nebula Operator（简称 Operator）是用于在 Kubernetes 系统上自动化部署和运维 Nebula Graph 集群的工具。依托于 Kubernetes 扩展机制，Nebula Graph 将其运维领域的知识全面注入至 Kubernetes 系统中，让 Nebula Graph 成为真正的云原生图数据库。详情请参考什么是 Nebula Operator。

Nebula Graph 版本	**Operator** 版本
v3.1.0	v1.1.0

2022 vesoft Inc.

25.4.8 Nebula Importer

Nebula Importer（简称 Importer）是一款 Nebula Graph 的 CSV 文件导入工具。Importer 可以读取本地的 CSV 文件，然后导入数据至 Nebula Graph 图数据库中。详情请参见什么是 Nebula Importer。

Nebula Graph 版本	**Importer** 版本
v3.1.0	v3.1.0

25.4.9 Nebula Spark Connector

Nebula Spark Connector 是一个 Spark 连接器，提供通过 Spark 标准形式读写 Nebula Graph 数据的能力。Nebula Spark Connector 由 Reader 和 Writer 两部分组成。详情请参见什么是 Nebula Spark Connector。

Nebula Graph 版本	**Spark Connector** 版本
v3.1.0	v3.0.0

25.4.10 Nebula Flink Connector

Nebula Flink Connector 是一款帮助 Flink 用户快速访问 Nebula Graph 的连接器，支持从 Nebula Graph 图数据库中读取数据，或者将其他外部数据源读取的数据写入 Nebula Graph 图数据库。详情请参见什么是 Nebula Flink Connector。

Nebula Graph 版本	**Flink Connector** 版本
v3.1.0	v3.0.0

25.4.11 Nebula Algorithm

Nebula Algorithm（简称 Algorithm）是一款基于 GraphX 的 Spark 应用程序，通过提交 Spark 任务的形式使用完整的算法工具对 Nebula Graph 数据库中的数据执行图计算，也可以通过编程形式调用 lib 库下的算法针对 DataFrame 执行图计算。详情请参见什么是 Nebula Algorithm。

Nebula Graph 版本	**Algorithm** 版本
v3.1.0	v3.0.0

25.4.12 Nebula Analytics

Nebula Analytics是一款集成了开源高性能图计算框架 Plato 的应用程序，支持利用 Plato 对 Nebula Graph 数据库中的数据执行图计算。详情请参见什么是 Nebula Analytics。

Nebula Graph 版本	**Analytics** 版本
v3.1.0	v3.2.0

25.4.13 Nebula Console

Nebula Console 是 Nebula Graph 的原生 CLI 客户端。如何使用请参见 Nebula Console。

Nebula Graph 版本	**Console** 版本
v3.1.0	v3.0.0

2022 vesoft Inc.

25.4.14 Nebula Docker Compose

Docker Compose 可以快速部署 Nebula Graph 集群。如何使用请参见 Docker Compose 部署 Nebula Graph。

Nebula Graph 版本	**Docker Compose** 版本
v3.1.0	v3.1.0

25.4.15 Backup & Restore

Backup&Restore（简称 BR）是一款命令行界面（CLI）工具，可以帮助备份 Nebula Graph 的图空间数据，或者通过备份文件恢复数据。

Nebula Graph 版本	**BR** 版本
v3.1.0	v0.6.1

25.4.16 Nebula Bench

Nebula Bench 用于测试 Nebula Graph 的基线性能数据，使用 LDBC v0.3.3 的标准数据集。

Nebula Graph 版本	**Bench** 版本
v3.1.0	v1.2.0

25.4.17 API、SDK

Compatibility

选择与内核版本相同 `X.Y.*` 的最新版本。

Nebula Graph 版本	语言（**commit id**）
v3.1.0	C++
v3.1.0	Go
v3.1.0	Python
v3.1.0	Java
v3.1.0	HTTP

25.4.18 未发布

- Rust Client
- Node.js Client
- Object Graph Mapping Library (OGM, or ORM)
- 测试

最后更新: June 20, 2023

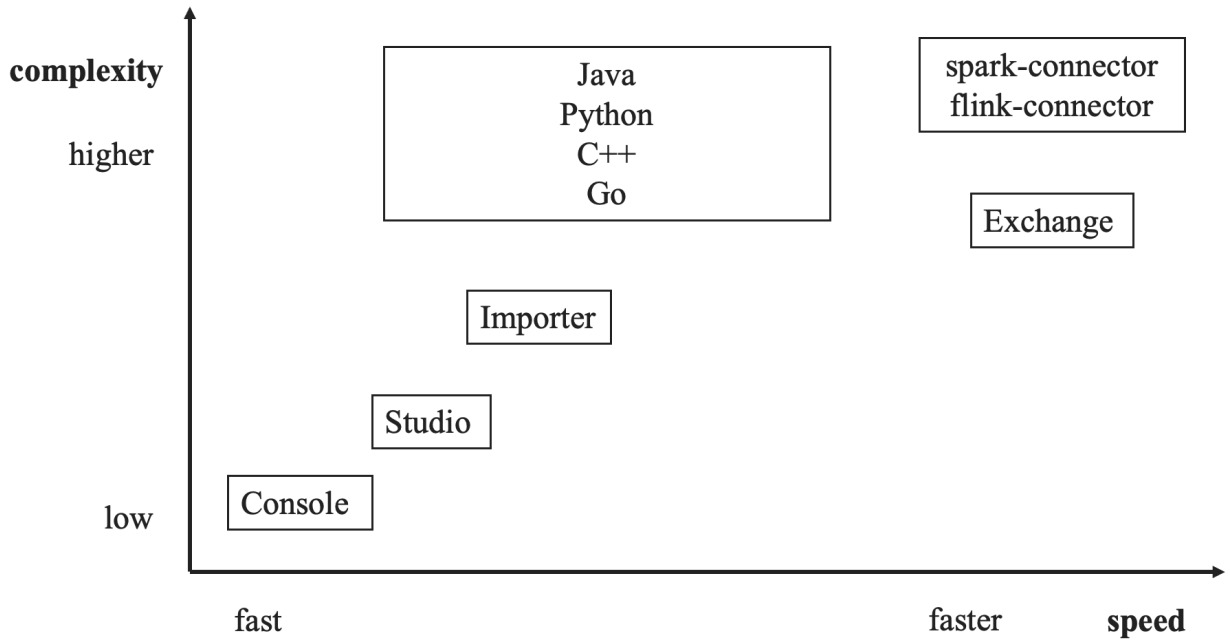
2022 vesoft Inc.

25.5 导入工具选择

有多种方式可以写入 Nebula Graph 3.1.0：

- 使用命令行 -f 的方式导入：可以导入少量准备好的 nGQL 文件，适合少量手工测试数据准备；
- 使用 studio 导入：可以用过浏览器导入本机多个 csv 文件，格式有限制；
- 使用 importer 导入：导入单机多个 csv 文件，大小没有限制，格式灵活；数据量十亿级以内；
- 使用 Exchange 导入：从 Neo4j, Hive, MySQL 等多种源分布式导入，需要有 Spark 集群；数据量十亿级以上
- 使用 Spark-connector/Flink-connector 导入：有相应组件 (Spark/Flink)，撰写少量代码；
- 使用 C++/GO/Java/Python SDK：编写程序的方式导入，需要有一定编程和调优能力。

下图给出了几种方式的定位：



最后更新: June 20, 2023

2022 vesoft Inc.

25.6 如何贡献代码和文档

25.6.1 开始之前

GitHub 或社区提交问题

欢迎为项目贡献任何代码或文档，但是建议先在 GitHub 或社区上提交一个问题，和大家共同讨论。

签署贡献者许可协议（**CLA**）

什么是 CLA?

签署协议链接：vesoft inc. Contributor License Agreement

单击按钮 **Sign in with GitHub to agree** 签署协议。

如果有任何问题，请发送邮件至 `info@vesoft.com` 。

25.6.2 修改单篇文档

Nebula Graph 文档以 Markdown 语言编写。单击文档标题右侧的铅笔图标即可提交修改建议。

该方法仅适用于修改单篇文档。

25.6.3 批量修改或新增文件

该方法适用于贡献代码、批量修改多篇文档或者新增文档。

Step 1：通过 **GitHub fork** 仓库

Nebula Graph 项目有很多仓库，以 nebula 仓库为例：

1. 访问 github.com/vesoft-inc/nebula。
2. 在右上角单击按钮 `Fork`，然后单击用户名，即可 fork 出 nebula 仓库。

Step 2：将分支克隆到本地

1. 定义本地工作目录。

```
# 定义工作目录。
working_dir=$HOME/Workspace
```

2. 将 `user` 设置为 GitHub 的用户名。

```
user={GitHub 用户名}
```

3. 克隆代码。

```
mkdir -p $working_dir
cd $working_dir
git clone https://github.com/$user/nebula.git
# 或：git clone git@github.com:$user/nebula.git

cd $working_dir/nebula
git remote add upstream https://github.com/vesoft-inc/nebula.git
# 或：git remote add upstream git@github.com:vesoft-inc/nebula.git

# 由于没有写访问权限，请勿推送至上游主分支。
git remote set-url --push upstream no_push

# 确认远程分支有效。
# 正确的格式为：
# origin    git@github.com:$(user)/nebula.git (fetch)
# origin    git@github.com:$(user)/nebula.git (push)
```


2022 vesoft Inc.

```
# upstream  https://github.com/vesoft-inc/nebula (fetch)
# upstream  no_push (push)
git remote -v
```

4. （可选）定义 pre-commit hook。

   请将 Nebula Graph 的 pre-commit hook 连接到 `.git` 目录。

   hook 将检查 commit，包括格式、构建、文档生成等。

```
cd $working_dir/nebula/.git/hooks
ln -s $working_dir/nebula/.linters/cpp/hooks/pre-commit.sh .
```

   pre-commit hook 有时候可能无法正常执行，用户必须手动执行。

```
cd $working_dir/nebula/.git/hooks
chmod +x pre-commit
```

Step 3：分支

1. 更新本地主分支。

```
cd $working_dir/nebula
git fetch upstream
git checkout master
git rebase upstream/master
```

2. 从主分支创建并切换分支：

```
git checkout -b myfeature
```

> **Note**
>
> 由于一个 PR 通常包含多个 commits，最终合入 upstream/master 分支时，我们会将这些 commits 挤压（squash）成一个 commit 进行合并。因此强烈建议创建一个独立的分支进行更改，这样在合入时才容易被挤压。合并后，这个分支可以被丢弃。如果未创建单独的分支，而是直接将 commits 提交至 origin/master，在合入时，可能会出现问题。 若未创建单独的分支（或是 origin/master 合并了其他的分支等），导致 origin/master 和 upstream/master 不一致时，用户可以使用 hard reset 强制两者进行一致。例如：
>
> ```
> git fetch upstream
> git checkout master
> git reset --hard upstream/master
> git push --force origin master
> ```

Step 4：开发

- 代码风格

2022 vesoft Inc.

Nebula Graph 采用 `cpplint` 来确保代码符合 Google 的代码风格指南。检查器将在提交代码之前执行。

- 单元测试要求

  请为新功能或 Bug 修复添加单元测试。

- 构建代码时开启单元测试

  详情请参见使用源码安装 Nebula Graph。

  > **Note**
  >
  > 请确保已设置 `-DENABLE_TESTING = ON` 启用构建单元测试。

- 运行所有单元测试

  在 `nebula` 根目录执行如下命令：

  ```
  cd nebula/build
  ctest -j$(nproc)
  ```

Step 5：保持分支同步

```
# 当处于 myfeature 分支时。
git fetch upstream
git rebase upstream/master
```

在其他贡献者将 PR 合并到基础分支之后，用户需要更新 head 分支。

Step 6：**Commit**

提交代码更改：

```
git commit -a
```

用户可以使用命令 `--amend` 重新编辑之前的代码。

Step 7：**Push**

需要审核或离线备份代码时，可以将本地仓库创建的分支 push 到 GitHub 的远程仓库。

```
git push origin myfeature
```

Step 8：创建 **pull request**

1. 访问 fork 出的仓库 `https://github.com/$user/nebula` （替换此处的用户名 `$user` ）。
2. 单击 `myfeature` 分支旁的按钮 `Compare & pull request` 。

Step 9：代码审查

pull request 创建后，至少需要两人审查。审查人员将进行彻底的代码审查，以确保变更满足存储库的贡献准则和其他质量标准。

25.6.4 添加测试用例

添加测试用例的方法参见 How to add test cases。

2022 vesoft Inc.

25.6.5 捐赠项目

Step 1：确认项目捐赠

通过邮件、微信、Slack 等方式联络 Nebula Graph 官方人员，确认捐赠项目一事。项目将被捐赠至 Nebula Contrib 组织下。

- 邮件地址：info@vesoft.com
- 微信：NebulaGraphbot
- Slack：Join Slack

Step 2：获取项目接收人信息

由 Nebula Graph 官方人员给出 Nebula Contrib 的项目接收者 ID。

Step 3：捐赠项目

由您将项目转移至本次捐赠的项目接受人，并由项目接收者将该项目转移至 Nebula Contrib 组织下。捐赠后，您将以 Maintain 角色继续主导社区项目的发展。

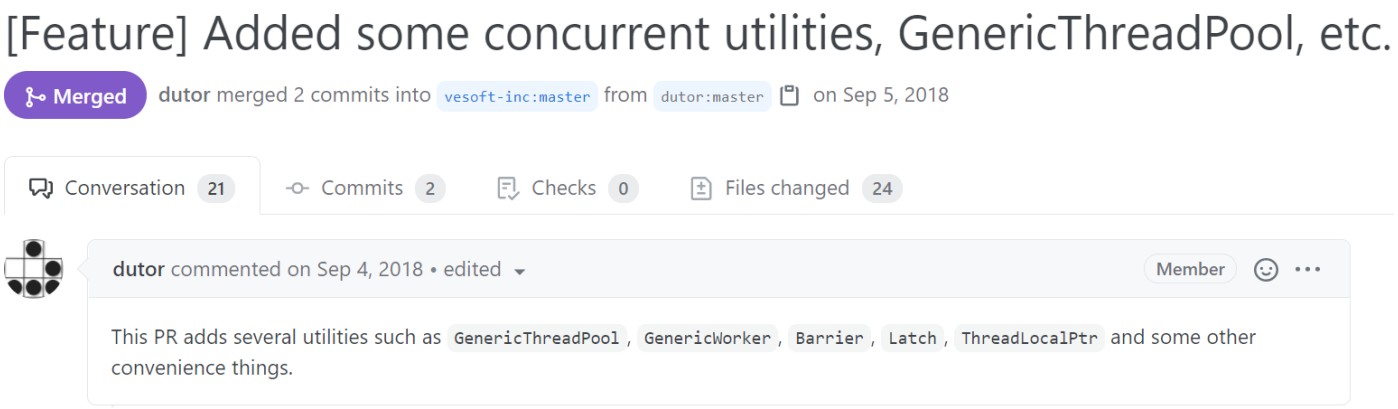
GitHub 上转移仓库的操作，请参见 Transferring a repository owned by your user account。

最后更新: June 20, 2023

2022 vesoft Inc.

25.7 Nebula Graph 年表

1. 2018.9.5 由 @dutor 提交了第一行代码。



2. 2019.5 发布了 v0.1.0 alpha 版本, 并开源。

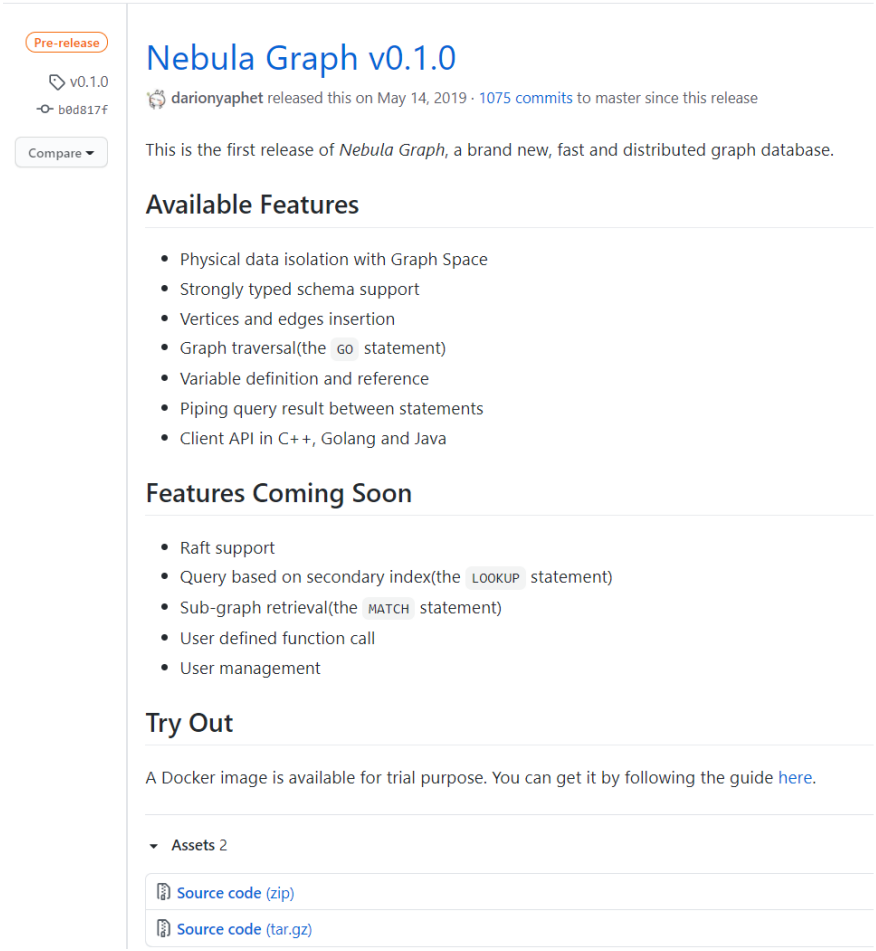
2022 vesoft Inc.





此后一年内陆续发布 v1.0.0-beta, v1.0.0-rc1, v1.0.0-rc2, v1.0.0-rc3, v1.0.0-rc4

2022 vesoft Inc.



Pre-release

v0.1.0

b0d817f

Compare

Nebula Graph v0.1.0

darionyaphet released this on May 14, 2019 · 1075 commits to master since this release

This is the first release of *Nebula Graph*, a brand new, fast and distributed graph database.

Available Features

- Physical data isolation with Graph Space
- Strongly typed schema support
- Vertices and edges insertion
- Graph traversal(the `GO` statement)
- Variable definition and reference
- Piping query result between statements
- Client API in C++, Golang and Java

Features Coming Soon

- Raft support
- Query based on secondary index(the `LOOKUP` statement)
- Sub-graph retrieval(the `MATCH` statement)
- User defined function call
- User management

Try Out

A Docker image is available for trial purpose. You can get it by following the guide here.

Assets 2

Source code (zip)

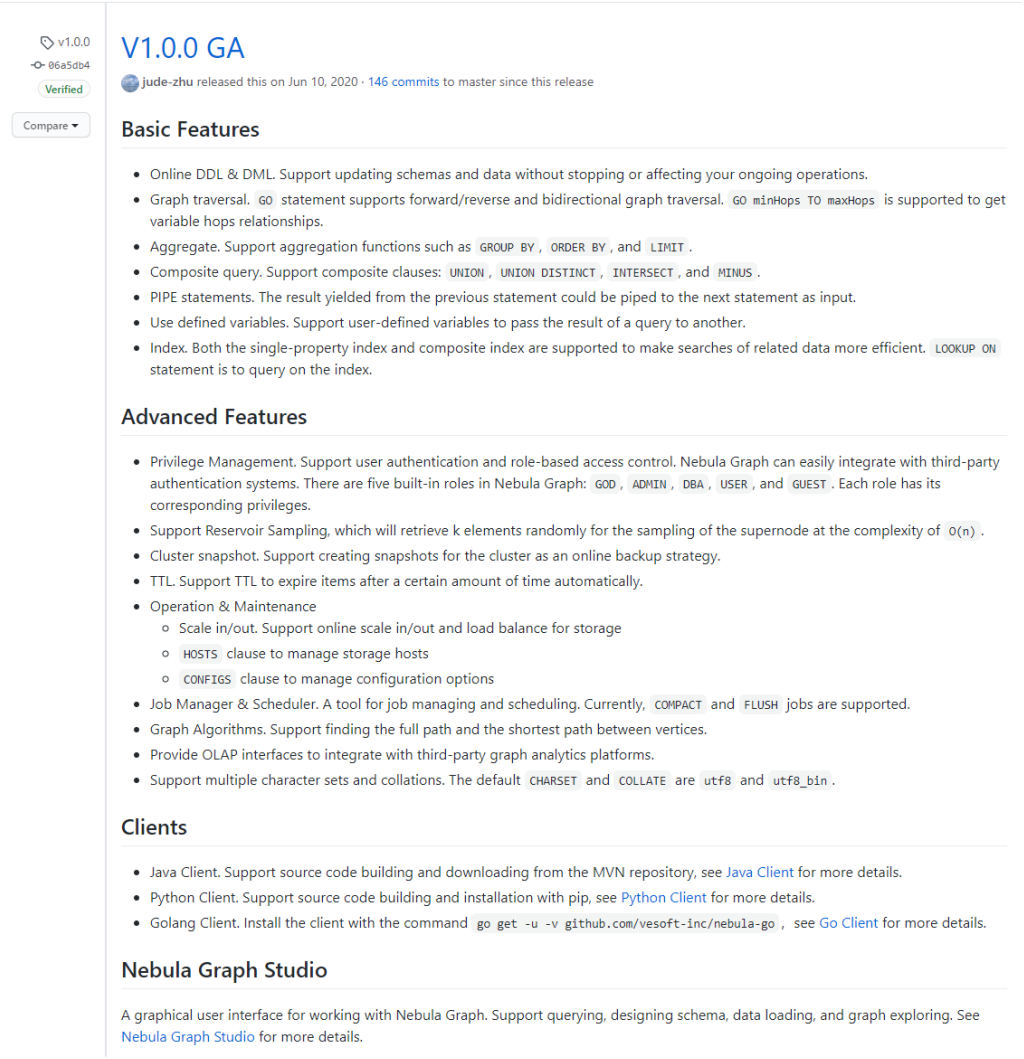
Source code (tar.gz)

3. 2019.7 在 HBaseCon 第一次公开亮相[1]@dangleptr

   image

4. 2020.3 在 v1.0 开发的收尾阶段，启动了 v2.0 项目的研发
5. 2020.6 发布了第一个正式大版本 v1.0.0 GA

2022 vesoft Inc.



v1.0.0
06a5db4
Verified
Compare

V1.0.0 GA

jude-zhu released this on Jun 10, 2020 · 146 commits to master since this release

Basic Features

- Online DDL & DML. Support updating schemas and data without stopping or affecting your ongoing operations.
- Graph traversal. `GO` statement supports forward/reverse and bidirectional graph traversal. `GO minHops TO maxHops` is supported to get variable hops relationships.
- Aggregate. Support aggregation functions such as `GROUP BY`, `ORDER BY`, and `LIMIT`.
- Composite query. Support composite clauses: `UNION`, `UNION DISTINCT`, `INTERSECT`, and `MINUS`.
- PIPE statements. The result yielded from the previous statement could be piped to the next statement as input.
- Use defined variables. Support user-defined variables to pass the result of a query to another.
- Index. Both the single-property index and composite index are supported to make searches of related data more efficient. `LOOKUP ON` statement is to query on the index.

Advanced Features

- Privilege Management. Support user authentication and role-based access control. Nebula Graph can easily integrate with third-party authentication systems. There are five built-in roles in Nebula Graph: `GOD`, `ADMIN`, `DBA`, `USER`, and `GUEST`. Each role has its corresponding privileges.
- Support Reservoir Sampling, which will retrieve k elements randomly for the sampling of the supernode at the complexity of `O(n)`.
- Cluster snapshot. Support creating snapshots for the cluster as an online backup strategy.
- TTL. Support TTL to expire items after a certain amount of time automatically.
- Operation & Maintenance
  - Scale in/out. Support online scale in/out and load balance for storage
  - `HOSTS` clause to manage storage hosts
  - `CONFIGS` clause to manage configuration options
- Job Manager & Scheduler. A tool for job managing and scheduling. Currently, `COMPACT` and `FLUSH` jobs are supported.
- Graph Algorithms. Support finding the full path and the shortest path between vertices.
- Provide OLAP interfaces to integrate with third-party graph analytics platforms.
- Support multiple character sets and collations. The default `CHARSET` and `COLLATE` are `utf8` and `utf8_bin`.

Clients

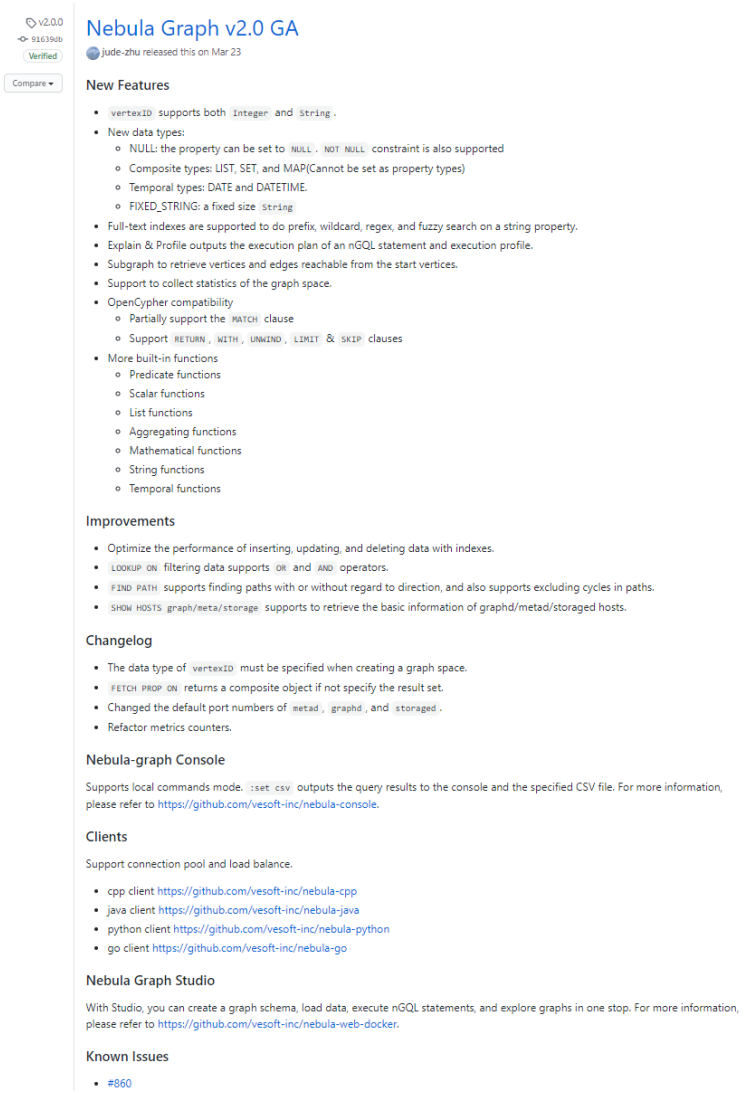
- Java Client. Support source code building and downloading from the MVN repository, see Java Client for more details.
- Python Client. Support source code building and installation with pip, see Python Client for more details.
- Golang Client. Install the client with the command `go get -u -v github.com/vesoft-inc/nebula-go`, see Go Client for more details.

Nebula Graph Studio

A graphical user interface for working with Nebula Graph. Support querying, designing schema, data loading, and graph exploring. See Nebula Graph Studio for more details.

6. 2021.3 发布了第二个大版本 v2.0 GA

2022 vesoft Inc.



v2.0.0
91639db
Verified
Compare

Nebula Graph v2.0 GA

jude-zhu released this on Mar 23

New Features

- `vertexID` supports both `Integer` and `String`.
- New data types:
  - NULL: the property can be set to `NULL`. `NOT NULL` constraint is also supported
  - Composite types: LIST, SET, and MAP(Cannot be set as property types)
  - Temporal types: DATE and DATETIME.
  - FIXED_STRING: a fixed size `String`
- Full-text indexes are supported to do prefix, wildcard, regex, and fuzzy search on a string property.
- Explain & Profile outputs the execution plan of an nGQL statement and execution profile.
- Subgraph to retrieve vertices and edges reachable from the start vertices.
- Support to collect statistics of the graph space.
- OpenCypher compatibility
  - Partially support the `MATCH` clause
  - Support `RETURN`, `WITH`, `UNWIND`, `LIMIT` & `SKIP` clauses
- More built-in functions
  - Predicate functions
  - Scalar functions
  - List functions
  - Aggregating functions
  - Mathematical functions
  - String functions
  - Temporal functions

Improvements

- Optimize the performance of inserting, updating, and deleting data with indexes.
- `LOOKUP ON` filtering data supports `OR` and `AND` operators.
- `FIND PATH` supports finding paths with or without regard to direction, and also supports excluding cycles in paths.
- `SHOW HOSTS graph/meta/storage` supports to retrieve the basic information of graphd/metad/storaged hosts.

Changelog

- The data type of `vertexID` must be specified when creating a graph space.
- `FETCH PROP ON` returns a composite object if not specify the result set.
- Changed the default port numbers of `metad`, `graphd`, and `storaged`.
- Refactor metrics counters.

Nebula-graph Console

Supports local commands mode. `:set csv` outputs the query results to the console and the specified CSV file. For more information, please refer to https://github.com/vesoft-inc/nebula-console.

Clients

Support connection pool and load balance.

- cpp client https://github.com/vesoft-inc/nebula-cpp
- java client https://github.com/vesoft-inc/nebula-java
- python client https://github.com/vesoft-inc/nebula-python
- go client https://github.com/vesoft-inc/nebula-go

Nebula Graph Studio

With Studio, you can create a graph schema, load data, execute nGQL statements, and explore graphs in one stop. For more information, please refer to https://github.com/vesoft-inc/nebula-web-docker.

Known Issues

- #860

7. 2021.8 发布 v2.5.0
8. 2021.10 发布 v2.6.0
9. 2022.2 发布 v3.0.0
10. 2022.4 发布 v3.1.0

---

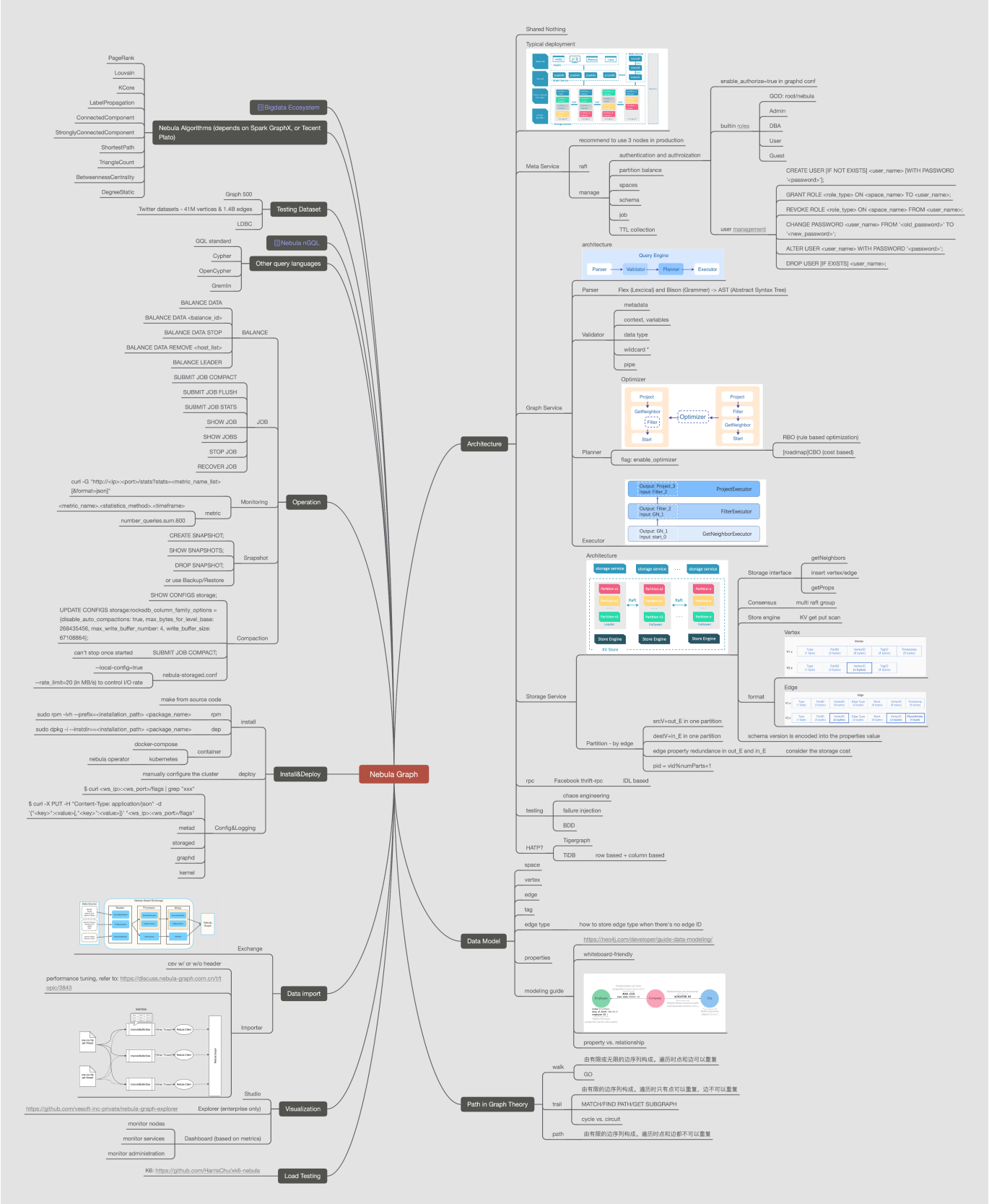
1. Nebula Graph 1.x 版本支持 RocksDB 和 HBase 两种主要的后端，但在 Nebula Graph 2.x 版本取消了默认对 HBase 的支持。 ↩

最后更新: June 20, 2023

2022 vesoft Inc.

25.8 思维导图

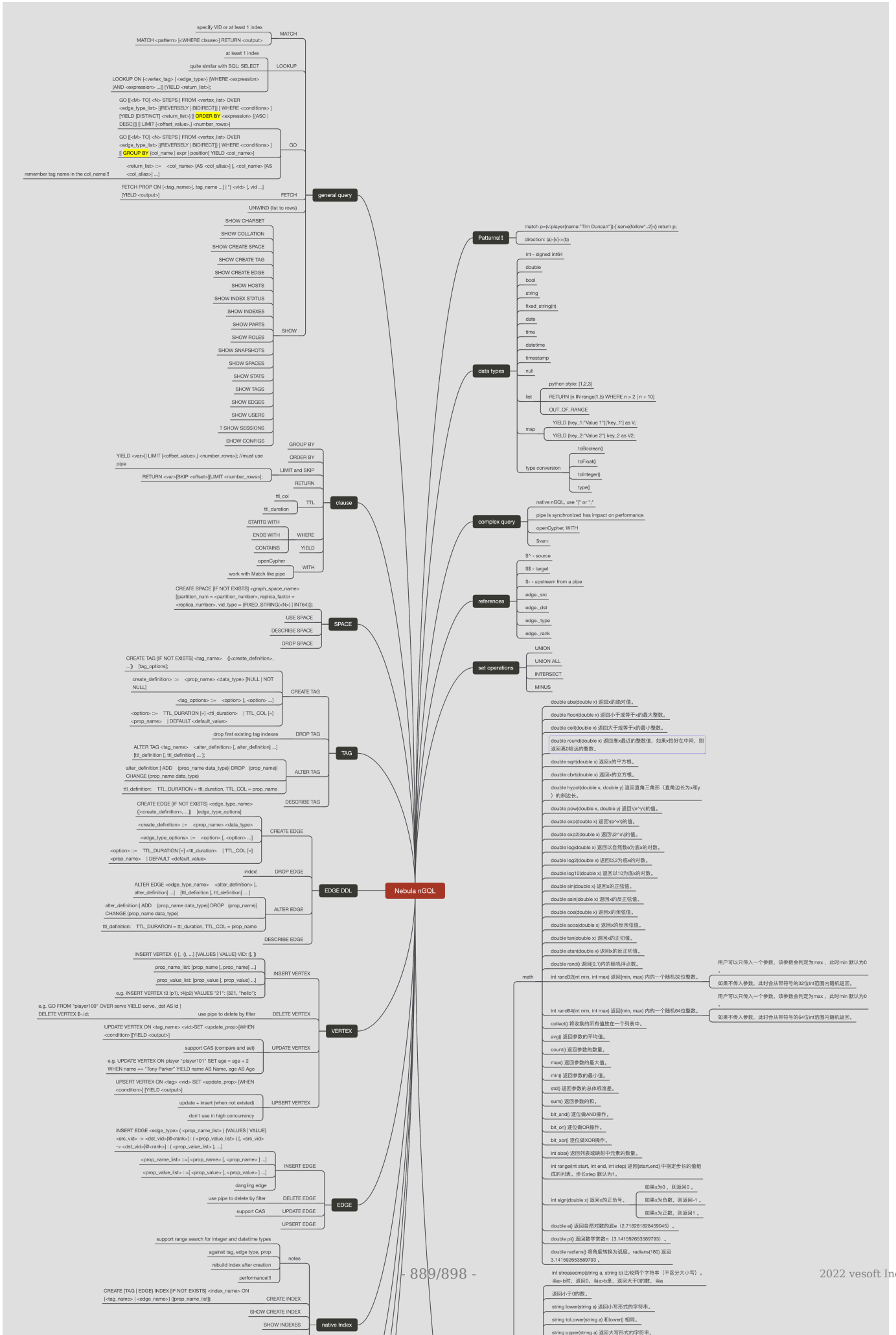
以下给出 Nebula Graph 结构框架的思维导图，用户可以点击 并查看大图。



2022 vesoft Inc.

以下给出 nGQL 的思维导图，用户可以点击 并查看大图。

2022 vesoft Inc.



2022 vesoft Inc.

最后更新: June 20, 2023

2022 vesoft Inc.

25.9 错误码

Nebula Graph 运行出现问题时，会返回错误码。本文介绍错误码的详细信息。

2022 vesoft Inc.

Note

- 如果出现错误但没有返回错误码，或错误码描述不清，请在论坛或 GitHub 反馈。
- 返回 `0` 表示执行成功。

2022 vesoft Inc.

错误码	说明
-1	连接断开
-2	无法建立连接
-3	RPC 失败
-4	Raft leader 变更
-5	图空间不存在
-6	Tag 不存在
-7	Edge type不存在
-8	索引不存在
-9	边属性不存在
-10	Tag 属性不存在
-11	当前角色不存在
-12	当前配置不存在
-13	当前主机不存在
-15	listener 不存在
-16	当前分区不存在
-17	key 不存在
-18	用户不存在
-19	统计信息不存在
-20	没有找到当前服务
-21	drainer 不存在
-22	drainer 客户端不存在
-24	备份失败
-25	备份的表为空
-26	备份表失败
-27	multiget 无法获得所有数据
-28	重建索引失败
-29	密码无效
-30	无法获得绝对路径
-1001	身份验证失败
-1002	无效会话
-1003	会话超时
-1004	语法错误
-1005	执行错误
-1006	语句为空
-1008	权限不足

2022 vesoft Inc.

错误码	说明
-1009	语义错误
-1010	超出最大连接数
-1011	访问存储失败（仅有部分请求成功）
-2001	主机不存在
-2002	主机已经存在
-2003	无效主机
-2004	当前命令、语句、功能不支持
-2007	配置项不能改变
-2008	参数与 meta 数据冲突
-2009	无效的参数
-2010	错误的集群
-2011	listener 冲突
-2021	存储数据失败
-2022	存储段非法
-2023	无效的数据均衡计划
-2024	集群已经处于数据均衡状态
-2025	没有正在运行的数据均衡计划
-2026	缺少有效的主机
-2027	已经损坏的数据均衡计划
-2029	缺少有效的 drainer
-2030	回收用户角色失败
-2031	无效的分区数量
-2032	无效的副本因子
-2033	无效的字符集
-2034	无效的字符排序规则
-2035	字符集和字符排序规则不匹配
-2040	生成快照失败
-2041	写入块数据失败
-2044	增加新的任务失败
-2045	停止任务失败
-2046	保存任务信息失败
-2047	数据均衡失败
-2048	当前任务还没有完成
-2049	任务报表失效
-2050	当前任务不在图空间内

2022 vesoft Inc.

错误码	说明
-2051	当前任务需要恢复
-2065	无效的任务
-2066	备份终止（正在创建索引）
-2067	备份时图空间不存在
-2068	备份恢复失败
-2069	会话不存在
-2070	获取集群信息失败
-2071	获取集群信息时无法获取绝对路径
-2072	获取集群信息时无法获得 agent
-2073	query 未找到
-2074	agent 没有汇报心跳
-2080	无效变量
-2081	变量值和类型不匹配
-3001	选举时无法达成共识
-3002	key 已经存在
-3003	数据类型不匹配
-3004	无效的字段值
-3005	无效的操作
-3006	当前值不允许为空
-3007	字段非空或者没有默认值时，字段值必须设置
-3008	取值超出了当前类型的范围
-3010	数据冲突
-3011	写入被延迟
-3021	不正确的数据类型
-3022	VID 长度无效
-3031	无效的过滤器
-3032	无效的字段更新
-3033	无效的 KV 存储
-3034	peer 无效
-3035	重试次数耗光
-3036	leader 转换失败
-3037	无效的统计类型
-3038	VID 无效
-3040	加载元信息失败
-3041	生成 checkpoint 失败

2022 vesoft Inc.

错误码	说明
-3042	生成 checkpoint 被阻塞
-3043	数据被过滤
-3044	无效的数据
-3045	并发写入同一条边发生冲突
-3046	并发写入同一个点发生冲突
-3047	锁已经失效
-3051	无效的任务参数
-3052	用户取消了任务
-3053	任务执行失败
-3060	执行计划被清除
-3061	客户端和服务端版本不兼容
-3062	获取 ID 序号失败
-3070	收到请求时心跳流程未完成
-3071	收到旧 leader 的过时心跳（已选举出新的 leader）
-3073	并发写入时与后到的请求发生冲突
-3500	未知的分区
-3501	raft 日志落后
-3502	raft 日志过期
-3503	心跳信息已经过期
-3504	未知的追加日志
-3511	等待快照完成
-3512	发送快照过程出错
-3513	无效的接收端
-3514	Raft 没有启动
-3515	Raft 已经停止
-3516	错误的角色
-3521	写入 WAL 失败
-3522	主机已经停止
-3523	请求数量过多
-3524	持久化快照失败
-3525	RPC 异常
-3526	没有发现 WAL 日志
-3527	主机暂停
-3528	写入被堵塞
-3529	缓存溢出

2022 vesoft Inc.

错误码	说明
-3530	原子操作失败
-3531	leader 租约过期
-3532	Raft 已经同步数据
-4001	drainer 日志落后
-4002	drainer 日志过期
-4003	drainer 数据存储无效
-4004	图空间不匹配
-4005	分区不匹配
-4006	数据冲突
-4007	请求冲突
-4008	数据非法
-5001	缓存配置错误
-5002	空间不足
-5003	没有命中缓存
-5005	写缓存失败
-7001	机器节点数超出限制
-7002	解析证书失败
-8000	未知错误

最后更新: June 20, 2023

2022 vesoft Inc.



https://docs.nebula-graph.com.cn/3.1.0

2022 vesoft Inc.