Nebula Graph Database 内核手册

v2.0.0

吴敏,周瑶,张莹,梁振亚

Table of contents

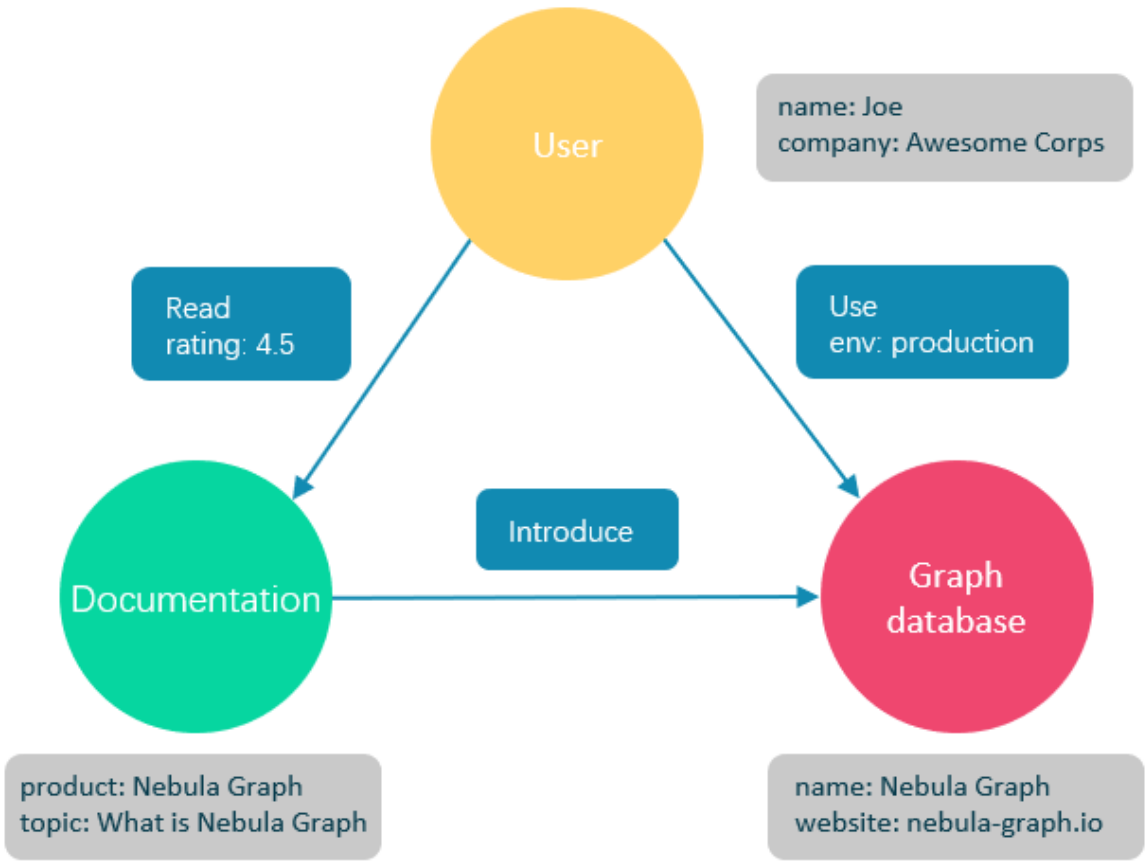
1. 简介

1.1 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，能够承载数十亿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。

1.1.1 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。它可以将图形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。



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作为一个典型的原生图数据库，允许将丰富的关系存储为边，边的类型和属性可以直接附加到边上。

1.1.2 Nebula Graph的优势

开源

Nebula Graph是在Apache 2.0和Commons Clause 1.0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 [https://github.com/vesoft-inc/nebula-graph]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 [https://discuss.nebula-graph.io/t/nebula-graph-1-0-benchmark-report/581]。

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 [#2.quick-start/6.useful-links/:]。

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 [#7.data-security/1.authentication/1.authentication/:]。

生态多样化

Nebula Graph开放了越来越多的原生工具，例如Nebula Graph Studio [https://github.com/vesoft-inc/nebula-web-docker]、nebula-console [https://github.com/vesoft-inc/nebula-console]、Nebula Graph Exchange [https://github.com/vesoft-inc/nebula-java/tree/master/tools/exchange]等。

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，也称为nGQL，是一种声明性的、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 [https://github.com/vesoft-inc/nebula-docs/tree/master/docs-2.0/3.ngql-guide]。

灵活数据建模

您可以轻松地在Nebula Graph中建立数据模型，不必将数据强制转换为关系表之类的结构，而且可以自由增加、更新和删除属性。详情请参见数据模型 [#1.introduction/2.data-model/:]。

广受欢迎

腾讯、vivo、美团和京东数科等科技巨头都在使用Nebula Graph。详情请参见Nebula Graph官网 [https://nebula-graph.com.cn/]。

1.1.3 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，建议您使用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形建模，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形数据，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。

1.2 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。

1.2.1 数据结构

Nebula Graph数据模型使用5种数据结构来保存数据：

- **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。
  - 点必须有至少一个标签（Tag）。

- **标签（tag）**

  标签可以用于对点进行区分。具有相同标签的点共享相同的属性定义。

- **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：

  - 一条边有且仅有一个边类型。
  - 起点、边类型（edge type）、rank、终点可以唯一表示一条边。
  - 边是有指向的。`->`表示边的指向，边可以沿任意方向遍历。
  - 边必须有rank。rank是一个不可更改的、用户分配的、64位有符号整数。通过它才能识别两个点之间具有相同边类型的边。边按它们的rank排序，值大的边排在前面。默认rank为0。

- **边类型（edge type）**

  边类型用于对边进行区分。具有相同边类型的边共享相同的属性定义。

- **属性（properties）**

  属性是指以键值对（key-value）形式存储点或边的相关信息。

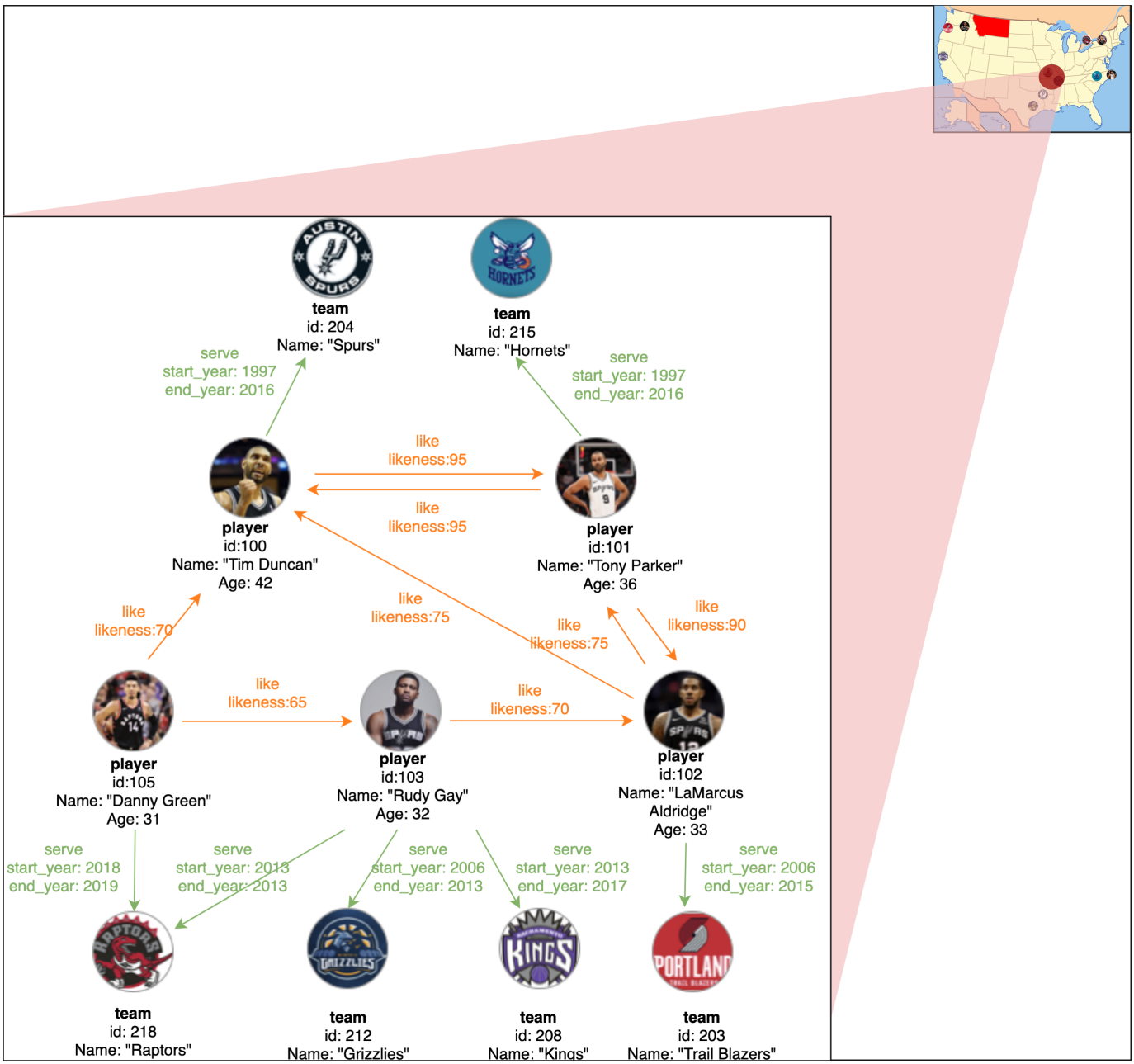
1.2.2 有向属性图

Nebula Graph将数据存储在有向属性图中。有向属性图是指点和边构成的图，这些边是有方向的。有向属性图表示为：

$$\mathbf{G = < V, E, P_V, P_E >}$$

- **V**是点的集合。
- **E**是有向边的集合。
- $\mathbf{P_V}$ 是点的属性。
- $\mathbf{P_E}$ 是边的属性。

通过下面的示例图来介绍有向属性图的基本概念。



图片展示了一组关于NBA球员和球队的数据。有两种类型的点（**player**、**name**）和两种类型的边（**serve**、**like**）。

下表为图片数据模型的详细信息。

类型	名称	属性名（数据类型）	说明
tag	**player**	name （string） age （int）	表示NBA球员。
tag	**team**	name （string）	表示NBA球队。
edge type	**serve**	start_year （int） end_year （int）	表示NBA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	**like**	likeness （int）	表示NBA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。

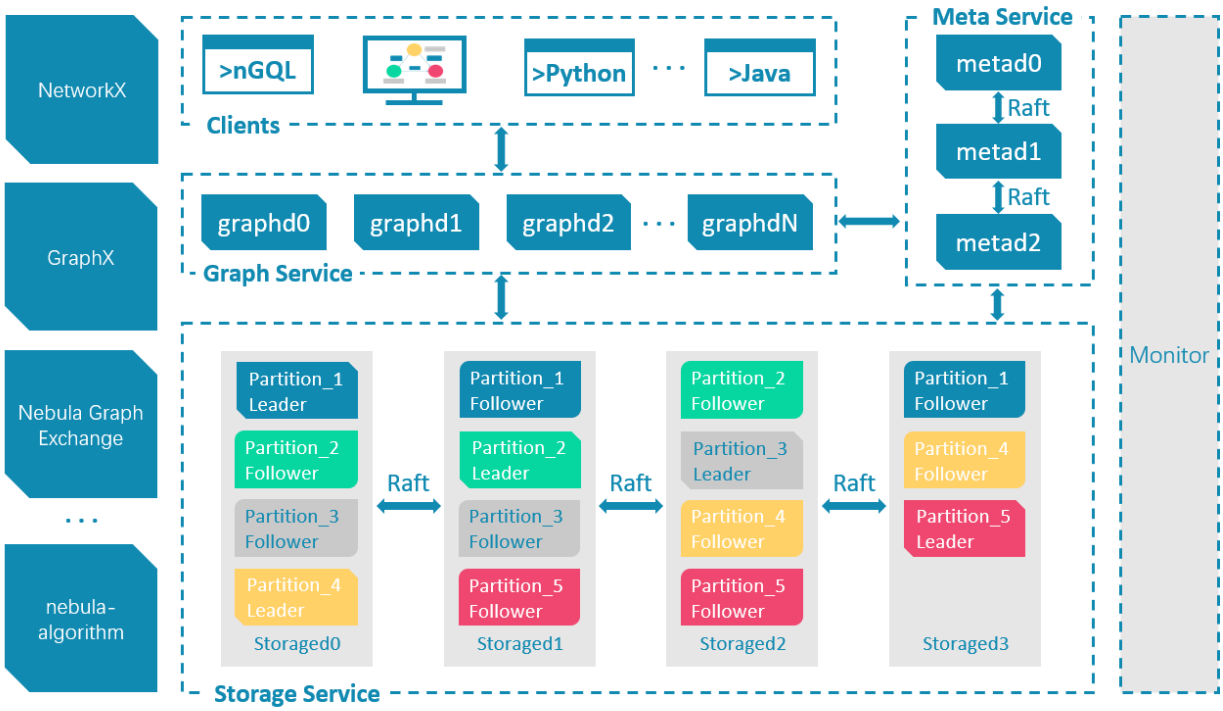
1.3 Nebula Graph架构

1.3.1 架构总览

Nebula Graph由三种服务构成：Graph服务、Meta服务和Storage服务。

每个服务都有可执行的二进制文件和对应进程，您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。



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 [#1.introduction/3.nebula-graph-architecture/2.meta-service/:]。

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。如果使用Nebula Graph Cloud [https://cloud.nebula-graph.com.cn/]，可以进一步节约成本。

- 开放更多可能性

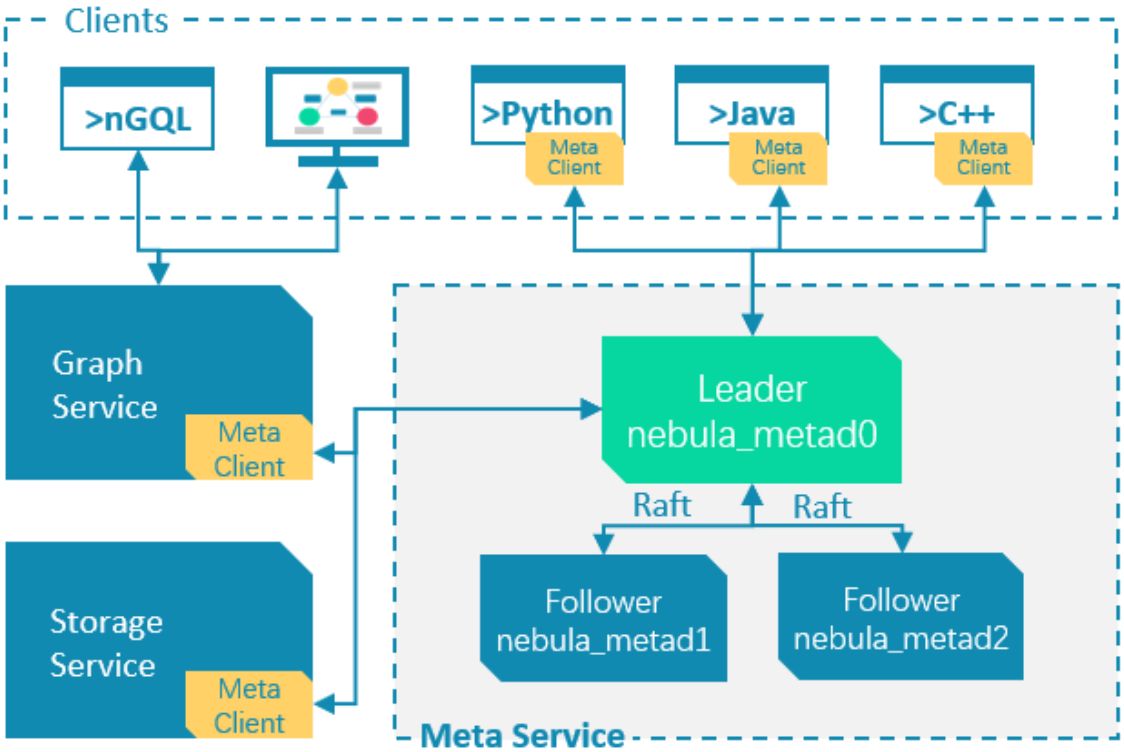
  基于分离架构的特性，Graph服务可以在多种存储引擎上单独运行，Storage服务也可以为多种计算引擎提供服务。

Graph服务和Storage服务的详细说明，请参见Graph服务 [#1.introduction/3.nebula-graph-architecture/3.graph-service/:]和Storage服务 [#1.introduction/3.nebula-graph-architecture/4.storage-service/:]。

1.3.2 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，您可以根据场景配置nebula-metad进程数量：

- 测试环境中，您可以在每个机器上配置1个或3个nebula-metad进程。
- 生产环境中，建议您在每个机器上都配置3个nebula-metad进程以保证高可用。

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。

> **说明**：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。

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。

更多Nebula Graph的访问控制说明，请参见身份验证 [#7.data-security/1.authentication/1.authentication/:]。

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包括标签、边类型、标签属性和边类型属性。

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 [#1.introduction/2.data-model/:]。

管理基于TTL的数据回收

Meta服务提供基于TTL（time to live）的自动数据回收和空间回收。

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。

1.3.3 Graph服务

> **说明**：编写该文档是下一个Nebula Graph技术文档工程师的培训任务。如果您想了解Graph服务，请参见Nebula Graph 2.0 Query Engine [https://nebula-graph.io/posts/nebula-query-engine-introduction/]。

1.3.4 Storage服务

> **说明**：该文档正用于招聘测试中，因此官方版本将在四月底发布。如果你想加入我们的团队，请随时联系我们 [https://discuss.nebula-graph.io/]。如果感兴趣，你也可以对这个话题提交贡献 [https://github.com/vesoft-inc/nebula-docs/tree/master/docs-2.0]。
>
> 参考文档
>
> - Nebula Graph's Storage Engine [https://nebula-graph.io/posts/nebula-graph-storage-engine-overview/]
> - Architecture overview [#1.introduction/3.nebula-graph-architecture/1.architecture-overview/:]
> - Meta Service [#1.introduction/3.nebula-graph-architecture/2.meta-service/:]

2. 快速入门

2.1 常见问题FAQ

2.1.1 关于openCypher兼容性

nGQL兼容openCypher 9吗？

nGQL部分兼容openCypher 9。

在Nebula Graph Issues [https://github.com/vesoft-inc/nebula-graph/issues]中已经列出已知的不兼容项。如果您发现这种类型的新问题，请提交问题并附带 `incompatible` 标签。您可以通过关键字 `compatibility` 搜索兼容性问题。

下面是nGQL和openCypher 9的主要差异（由于设计原因而不兼容）。

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（ `CREATE` 、 `MERGE` 等）和部分 `MATCH` 语句。

> **说明**：请注意openCypher 9 [http://www.opencypher.org/]和Cypher [https://neo4j.com/developer/cypher/]在语法和许可上有一些不同。例如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暂不支持事务。

哪里可以找到更多nGQL的示例？

您可以在Nebula Graph GitHub的features [https://github.com/vesoft-inc/nebula-graph/tree/master/tests/tck/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: Match seek by tag

  Background: Prepare space
    Given a graph with space named "nba"

  Scenario: seek by empty tag index
    When executing query:
      """
      MATCH (v:bachelor)
      RETURN id(v) AS vid
      """
    Then the result should be, in any order:
      | vid          |
      | 'Tim Duncan' |
    And no side effects
```

```
    When executing query:
      """
      MATCH (v:bachelor)
      RETURN id(v) AS vid, v.age AS age
      """
    Then the result should be, in any order:
      | vid          | age |
      | 'Tim Duncan' | 42  |
    And no side effects
```

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。
`Then`	描述执行 `When` 内语句的预期返回结果。如果您的返回结果和文档不同，请提交issue [https://github.com/vesoft-inc/nebula-graph/issues]通知Nebula Graph团队。
`And`	描述执行 `When` 内语句的副作用。
`@skip`	跳过这个示例。通常表示测试代码还没有准备好。

欢迎您增加更多实用场景示例 [https://docs.nebula-graph.io/1.1/manual-EN/4.contributions/how-to-contribute/]，成为Nebula Graph贡献者。

2.1.2 关于数据模型

Nebula Graph支持 W3C 的RDF（SPARQL 或 GraphQL）吗？

不支持。

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

Nebula Graph的查询语言也不支持SPARQL和GraphQL。

2.1.3 关于执行

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 |
+------+
| nba  |
```

```
+------+
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。

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。

如果 `replica_factor=2`，当其中一个副本故障时，就会导致系统无法工作；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。

建议您在生产环境中设置 `replica_factor=3`，测试环境中设置 `replica_factor=1`，不要使用偶数。

如何处理报错`[ERROR (-7)]: SyntaxError: syntax error near`？

大部分情况下，查询语句需要有 `YIELD` 或 `RETURN`，请检查您的查询语句是否包含。

如何统计每种Tag有多少个点，每个边类型有多少条边?

请参见show-stats [#3.ngql-guide/7.general-query-statements/6.show/14.show-stats/:]。

如何获取每种Tag的所有点，或者每种边类型的所有边?

1. 建立并重建索引

```
> CREATE TAG INDEX i_player ON player();
> REBUILD TAG INDEX i_player;
```

1. 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 [#3.ngql-guide/14.native-index-statements/1.create-native-index/:]、LOOKUP [#3.ngql-guide/7.general-query-statements/5.lookup/:]和MATCH [#3.ngql-guide/7.general-query-statements/2.match/:]。

Error `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 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`。

Error `Storage Error: The VID must be a 64-bit integer or a string.`

检查输入的 vid 是否是 create space 设置的整型或者 `fix_string(N)`. 如果是字符串类型，检查长度是否超过 `N` (默认为 `8`). 见 create space [#3.ngql-guide/9.space-statements/1.create-space/:].

2.1.4 关于运维

日志文件过大时如何回收日志？

Nebula Graph使用 glog [https://github.com/google/glog] 打印日志。glog 没有日志回收的功能，您可以使用 crontab 设置定期任务回收日志文件，详情请参见Glog should delete old log files automaticly [https://github.com/google/glog/issues/423]。

2.1.5 关于本手册

为什么手册示例和系统行为不一致？

Nebula Graph 2.0一直在持续开发，功能或操作的行为可能会有变化，如果您发现不一致，请提交issue [https://github.com/vesoft-inc/nebula-graph/issues]通知Nebula Graph团队。

2.1.6 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 [#5.configurations-and-logs/1.configurations/1.configurations/:]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。

如何测试端口是否已开放

您可以使用如下telnet命令检查端口状态：

```
telnet <ip> <port>
```

> **说明：** 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
Trying 192.168.1.10...
telnet: connect to address 192.168.1.10: Connection refused
```

如何查看Nebula Graph版本

1. 使用 `<binary_path> --version` 命令查看相应服务二进制文件的Git Commit ID。

   例如，要查看Graph服务的版本，进入Graph服务对应的二进制文件 `nebula-graphd` 所在目录，运行 `./nebula-graphd --version` 。

   ```
   ./nebula-graphd --version
   nebula-graphd version Git: ab4f683, Build Time: Mar 24 2021 02:17:30
   This source code is licensed under Apache 2.0 License, attached with Common Clause Condition 1.0.
   ```

2. 在GitHub Commits [https://github.com/vesoft-inc/nebula-graph/commits/master]页搜索该Commit ID，获取Commit的提交时间。

3. 对比Commit提交时间和GitHub Releases [https://github.com/vesoft-inc/nebula-graph/releases]页的版本发布时间，即可确定Nebula Graph的版本。

2.2 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。

1. Docker Compose部署Nebula Graph [#2.quick-start/2.deploy-nebula-graph-with-docker-compose/:]
2. 连接Nebula Graph [#2.quick-start/3.connect-to-nebula-graph/:]
3. 基础操作语法 [#2.quick-start/4.nebula-graph-crud/:]

以下是其他推荐您阅读的文档。在您尝试过快速入门的操作后，您可能会需要阅读它们来进一步了解和使用Nebula Graph。

- 常见问题FAQ [#2.quick-start/0.FAQ/:]
- 部署Nebula Graph集群 [#4.deployment-and-installation/deploy-nebula-graph-cluster/:]
- 一些常用链接 [#2.quick-start/6.useful-links/:]
- Compaction [#8.service-tuning/compaction/:]

2.3 Docker Compose部署Nebula Graph

有多种方式可以部署Nebula Graph，但是使用Docker Compose通常是最快的方式。

2.3.1 阅读指南

如果您是带着如下问题阅读本文，可以直接单击问题跳转查看对应的说明。

- 部署Nebula Graph之前需要做什么准备工作？[#2.quick-start/2.deploy-nebula-graph-with-docker-compose/:_2]
- 如何通过Docker Compose快速部署Nebula Graph？[#2.quick-start/2.deploy-nebula-graph-with-docker-compose/:nebula_graph]
- 如何检查Nebula Graph服务的状态和端口？[#2.quick-start/2.deploy-nebula-graph-with-docker-compose/:nebula_graph_1]
- 如何检查Nebula Graph服务的数据和日志？[#2.quick-start/2.deploy-nebula-graph-with-docker-compose/:nebula_graph_2]
- 如何停止Nebula Graph服务？[#2.quick-start/2.deploy-nebula-graph-with-docker-compose/:nebula_graph_3]
- 如何通过其他方式部署Nebula Graph？[#2.quick-start/2.deploy-nebula-graph-with-docker-compose/:_3]

2.3.2 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 [https://docs.docker.com/engine/install/]
Docker Compose	最新版本	Install Docker Compose [https://docs.docker.com/compose/install/]
Git	最新版本	Download Git [https://git-scm.com/download/]

- 如果您使用非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 [https://docs.docker.com/engine/install/linux-postinstall/#manage-docker-as-a-non-root-user]。
- 启动主机上的Docker服务。
- 如果您已经通过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，需要您删除目录`nebula-docker-compose/data`。

  > **说明**：如果您需要备份服务数据，请参见使用BR备份数据 [#2.quick-start/6.useful-links/:] TODO: 未和v2.0.0一起发布。

2.3.3 部署和连接Nebula Graph

1. 通过Git克隆 `nebula-docker-compose` 仓库的 `master` 分支到您的主机。

   > **禁止**： `master` 分支包含最新的Nebula Graph开发版本的Docker Compose解决方案。请**不要**在生产环境使用此版本。

   ```
   $ git clone https://github.com/vesoft-inc/nebula-docker-compose.git
   ```

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。

   > **说明**: 如果长期未更新镜像，请先更新Nebula Graph镜像 [#2.quick-start/2.deploy-nebula-graph-with-docker-compose/:nebula_graphdocker]和Nebula Console镜像 [#2.quick-start/2.deploy-nebula-graph-with-docker-compose/:nebula_console]。

   ```
   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
   ```

   > **说明**: 上述服务的更多信息，请参见架构总览 [#1.introduction/3.nebula-graph-architecture/1.architecture-overview/:]。

4. 连接Nebula Graph。

   1. 使用Nebula Console镜像启动一个容器，并连接到Nebula Graph服务所在的网络（nebula-docker-compose_nebula-net）中。

      ```
      $ docker run --rm -ti --network nebula-docker-compose_nebula-net --entrypoint=/bin/sh vesoft/nebula-console:v2-nightly
      ```

      > **说明**：您的本地网络可能和示例中的 `nebula-docker-compose_nebula-net` 不同，请使用如下命令查看。

      ```
      $ docker network  ls
      NETWORK ID          NAME                                DRIVER              SCOPE
      a74c312b1d16        bridge                              bridge              local
      dbfa82505f0e        host                                host                local
      ed55ccf356ae        nebula-docker-compose_nebula-net    bridge              local
      93ba48b4b288        none                                null                local
      ```

   1. 通过Nebula Console连接Nebula Graph。

      ```
      docker> nebula-console -u user -p password --address=graphd --port=9669
      ```

      > **说明**：默认情况下，身份认证功能是关闭的，可以使用任意用户名和密码登录。如果想使用身份认证，请参见身份认证 [#7.data-security/1.authentication/1.authentication/:]。

1. 执行如下命令检查 `nebula-storaged` 进程状态。

```
nebula> SHOW HOSTS;
+-------------+------+----------+--------------+----------------------+------------------------+
| Host        | Port | Status   | Leader count | Leader distribution  | Partition distribution |
+-------------+------+----------+--------------+----------------------+------------------------+
| "storaged0" | 9779 | "ONLINE" | 0            | "No valid partition" | "No valid partition"   |
+-------------+------+----------+--------------+----------------------+------------------------+
| "storaged1" | 9779 | "ONLINE" | 0            | "No valid partition" | "No valid partition"   |
+-------------+------+----------+--------------+----------------------+------------------------+
| "storaged2" | 9779 | "ONLINE" | 0            | "No valid partition" | "No valid partition"   |
+-------------+------+----------+--------------+----------------------+------------------------+
| "Total"     |      |          | 0            |                      |                        |
+-------------+------+----------+--------------+----------------------+------------------------+
```

5. 执行两次 `exit` 可以退出容器。

2.3.4 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
Name                     Command                       State                                                Ports
-------------------------------------------------------------------------------------------------------------------------------------------------------
nebula-docker-compose_graphd1_1     ./bin/nebula-graphd --flag ...   Up (health: starting)   13000/tcp, 13002/tcp, 0.
0.0.0:33295->19669/tcp, 0.0.0.0:33291->19670/tcp,
                                                                                             3699/tcp, 0.0.0.0:33298->9669/tcp
nebula-docker-compose_graphd2_1     ./bin/nebula-graphd --flag ...   Up (health: starting)   13000/tcp, 13002/tcp, 0.
0.0.0:33285->19669/tcp, 0.0.0.0:33284->19670/tcp,
                                                                                             3699/tcp, 0.0.0.0:33286->9669/tcp
nebula-docker-compose_graphd_1      ./bin/nebula-graphd --flag ...   Up (health: starting)   13000/tcp, 13002/tcp, 0.
0.0.0:33288->19669/tcp, 0.0.0.0:33287->19670/tcp,
                                                                                             3699/tcp, 0.0.0.0:9669->9669/tcp
nebula-docker-compose_metad0_1      ./bin/nebula-metad --flagf ...   Up (health: starting)   11000/tcp, 11002/tcp, 0.
0.0.0:33276->19559/tcp, 0.0.0.0:33275->19560/tcp,
                                                                                             45500/tcp, 45501/tcp, 0.
0.0.0:33278->9559/tcp
nebula-docker-compose_metad1_1      ./bin/nebula-metad --flagf ...   Up (health: starting)   11000/tcp, 11002/tcp, 0.
0.0.0:33279->19559/tcp, 0.0.0.0:33277->19560/tcp,
                                                                                             45500/tcp, 45501/tcp, 0.
0.0.0:33281->9559/tcp
nebula-docker-compose_metad2_1      ./bin/nebula-metad --flagf ...   Up (health: starting)   11000/tcp, 11002/tcp, 0.
0.0.0:33282->19559/tcp, 0.0.0.0:33280->19560/tcp,
                                                                                             45500/tcp, 45501/tcp, 0.
0.0.0:33283->9559/tcp
nebula-docker-compose_storaged0_1   ./bin/nebula-storaged --fl ...   Up (health: starting)   12000/tcp, 12002/tcp, 0.
0.0.0:33290->19779/tcp, 0.0.0.0:33289->19780/tcp,
                                                                                             44500/tcp, 44501/tcp, 0.
0.0.0:33294->9779/tcp
nebula-docker-compose_storaged1_1   ./bin/nebula-storaged --fl ...   Up (health: starting)   12000/tcp, 12002/tcp, 0.
0.0.0:33296->19779/tcp, 0.0.0.0:33292->19780/tcp,
                                                                                             44500/tcp, 44501/tcp, 0.
0.0.0:33299->9779/tcp
nebula-docker-compose_storaged2_1   ./bin/nebula-storaged --fl ...   Up (health: starting)   12000/tcp, 12002/tcp, 0.
0.0.0:33297->19779/tcp, 0.0.0.0:33293->19780/tcp,
                                                                                             44500/tcp, 44501/tcp, 0.
0.0.0:33300->9779/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。

2.3.5 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
```

2.3.6 停止Nebula Graph服务

您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_storaged0_1 ... done
Stopping nebula-docker-compose_graphd1_1   ... done
Stopping nebula-docker-compose_graphd_1    ... done
Stopping nebula-docker-compose_storaged1_1 ... done
Stopping nebula-docker-compose_graphd2_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_storaged0_1 ... done
Removing nebula-docker-compose_graphd1_1   ... done
Removing nebula-docker-compose_graphd_1    ... done
Removing nebula-docker-compose_storaged1_1 ... done
Removing nebula-docker-compose_graphd2_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
```

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

2.3.7 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:v2-nightly
    ...
    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。

如何更新Nebula Graph服务的Docker镜像？

在目录 `nebula-docker-compose` 内执行命令 `docker-compose pull`，可以更新Graph服务、Storage服务和Meta服务的镜像。

执行命令`docker-compose pull`报错`ERROR: toomanyrequests`

您可能遇到如下错误：

```
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
```

以上错误表示您已达到Docker Hub的速率限制。解决方案请参见Understanding Docker Hub Rate Limiting [https://www.docker.com/increase-rate-limit]。

如何更新Nebula Console？

执行如下命令可以更新Nebula Console客户端镜像。

```
docker pull vesoft/nebula-console:v2-nightly
```

如何升级Nebula Graph？

更新Nebula Graph的Docker镜像并重启服务：

1. 在目录 `nebula-docker-compose` 内，执行命令 `docker-compose pull` 更新Nebula Graph的Docker镜像。
2. 执行命令 `docker-compose down` 停止Nebula Graph服务。
3. 执行命令 `docker-compose up -d` 启动Nebula Graph服务。

为什么更新nebula-docker-compose仓库（Nebula Graph 2.0.0-RC）后，无法通过端口3699连接Nebula Graph？

在Nebula Graph 2.0.0-RC版本，默认端口从 `3699` 改为 `9669` 。请使用 `9669` 端口连接，或修改配置文件 `docker-compose.yaml` 内的端口。

为什么更新nebula-docker-compose仓库后，无法访问数据？（2021年01月04日）

如果您在2021年01月04日后更新过nebula-docker-compose仓库，而且之前已经有数据，请修改文件 `docker-compose.yaml` ，将端口修改为之前使用的端口。详情请参见修改默认端口 [https://github.com/vesoft-inc/nebula-docker-compose/commit/2a612f1c4f0e2c31515e971b24b355b3be69420a]。

为什么更新nebula-docker-compose仓库后，无法访问数据？（2021年01月27日）

2021年01月27日修改了数据格式，无法兼容之前的数据，请执行命令 `docker-compose down -v` 删除所有本地数据。

2.3.8 相关文档

- 使用源码安装部署Nebula Graph [#4.deployment-and-installation/2.compile-and-install-nebula-graph/1.install-nebula-graph-by-compiling-the-source-code/:]
- 使用RPM或DEB安装包安装Nebula Graph [#4.deployment-and-installation/2.compile-and-install-nebula-graph/2.install-nebula-graph-by-rpm-or-deb/:]
- 多种方式连接Nebula Graph [#2.quick-start/3.connect-to-nebula-graph/:]

2.4 管理Nebula Graph服务

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

`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。

2.4.1 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。

2.4.2 启动Nebula Graph服务

非容器部署

执行如下命令启动Nebula Graph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
```

容器部署

在 `nebula-docker-compose/` 目录内执行如下命令启动Nebula Graph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
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
```

2.4.3 停止Nebula Graph服务

> **禁止**：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
```

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
```

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

2.4.4 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：

```
$ 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: Running as 26601, Listening on 9559
  [INFO] nebula-graphd: Running as 26644, Listening on 9669
  [INFO] nebula-storaged: Running as 26709, Listening on 9779
  ```

- 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[https://discuss.nebula-graph.com.cn/]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
  ```

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/` ，您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，您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`)，最后登录容器排查问题。

```
nebula-docker-compose]$ docker ps
CONTAINER ID   IMAGE                               COMMAND                  CREATED          STATUS
PORTS                                                                                 NAMES
2a6c56c405f5   vesoft/nebula-graphd:v2-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:v2-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:v2-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:v2-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:v2-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:v2-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:v2-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:v2-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:v2-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

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

2.5 连接Nebula Graph

Nebula Graph支持多种类型客户端，包括CLI客户端、GUI客户端和流行编程语言开发的客户端。本文将概述Nebula Graph客户端，并介绍如何使用原生CLI客户端Ne'bu'la。

2.5.1 Nebula Graph客户端

您可以使用已支持的客户端或者命令行工具 [#2.quick-start/6.useful-links/:] 来连接Nebula Graph数据库.

2.5.2 使用Nebula Console连接Nebula Graph

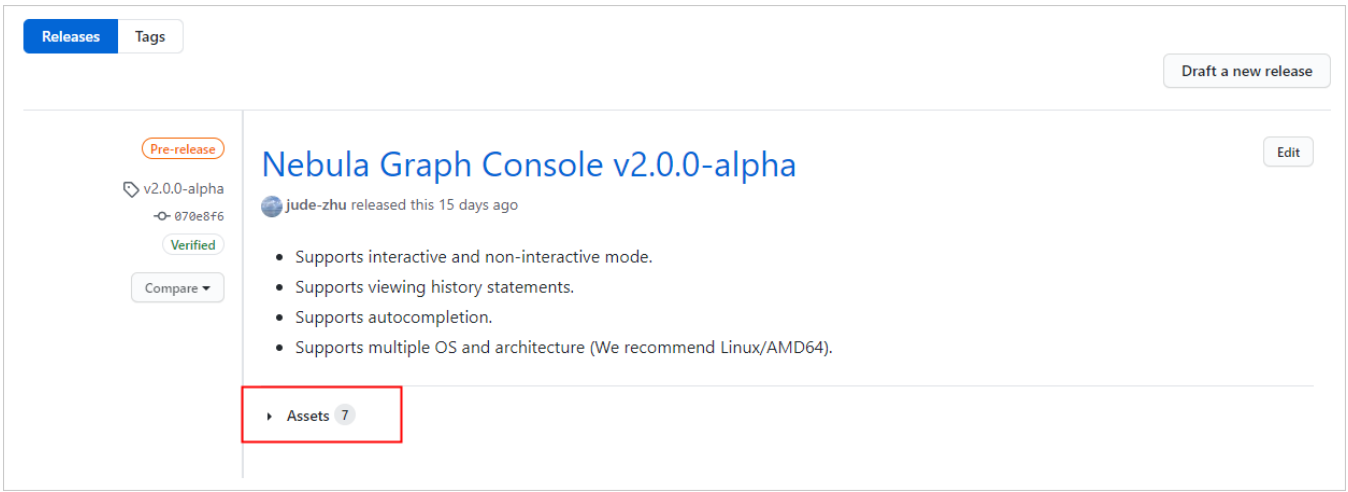
前提条件

- Nebula Graph服务已启动。如何启动服务，请参见启动和停止Nebula Graph服务 [#2.quick-start/5.start-stop-service/:]。
- 运行Nebula Console的机器和运行Nebula Graph的服务器网络互通。

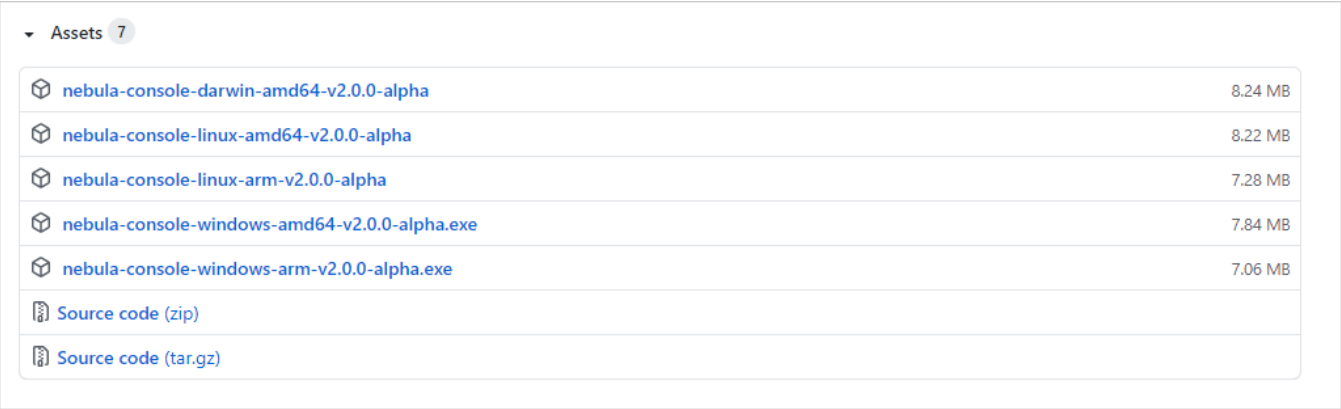
操作步骤

1. 在nebula-console [https://github.com/vesoft-inc/nebula-console/releases]页面，确认需要的版本，单击**Assets**。

   > **说明**：建议您选择**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` 。

   > **说明**：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。

   > **说明**：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。

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`	显示帮助菜单。
`-addr`	设置要连接的graphd服务的IP地址。默认地址为127.0.0.1。
`-port`	设置要连接的graphd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，用户名可以填写任意字符。
`-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。

您可以在项目仓库 [https://github.com/vesoft-inc/nebula-console/tree/v2.0.0-ga]找到更多细节.

2.5.3 Nebula Console导出模式

导出模式开启时，Nebula Console会导出所有请求的结果到CSV格式文件中。关闭导出模式会停止导出。使用语法如下：

> **说明**：
>
> - 命令不区分大小写。
> - CSV格式文件保存在当前工作目录中，即Linux命令 `pwd` 显示的目录。

- 开启导出模式

```
nebula> :SET CSV <your_file.csv>
```

- 关闭导出模式

```
nebula> :UNSET CSV
```

2.5.4 使用Nebula Console断开连接

您可以使用 `: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` 。

```
nebula> :QUIT

Bye root!
```

2.5.5 常见问题

如何通过源码安装Nebula Console？

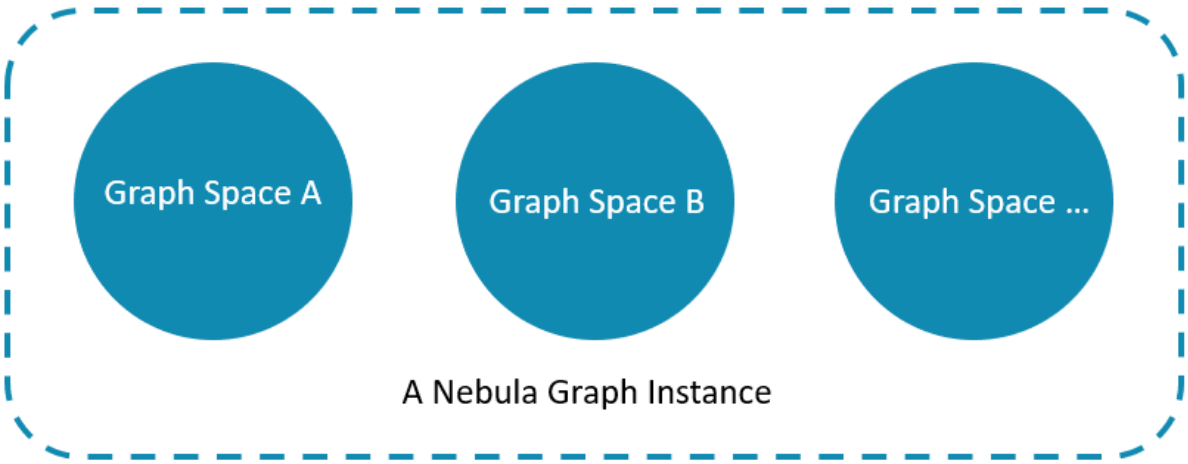
下载和编译Nebula Console的最新源码，请参见GitHub nebula console [https://github.com/vesoft-inc/nebula-console#build-nebula-graph-console]页面的说明。

2.6 基础操作语法

本文介绍Nebula Graph基础操作的语法。

2.6.1 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，您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。

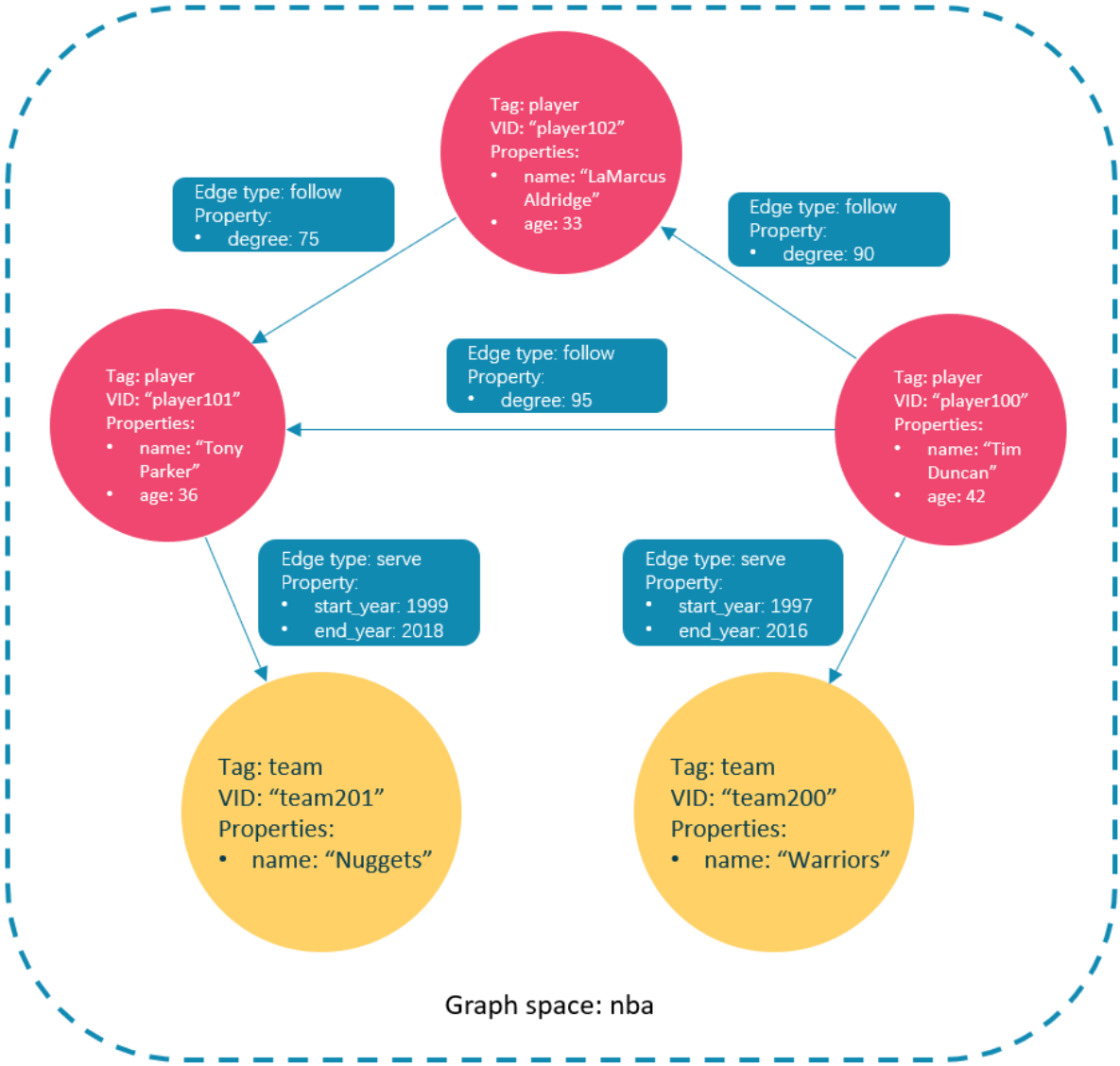


为了在图空间中插入数据，需要为图数据库定义一个Schema。Nebula Graph的Schema是由如下几部分组成。

组成部分	说明
点（vertex）	表示现实世界中的实体。一个点可以有一个或多个标签。
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，定义了一组描述边类型的属性。

更多信息，请参见数据结构 [#1.introduction/2.data-model/:]。

本文将使用下图的数据集演示基础操作的语法。



2.6.2 检查Nebula Graph集群的机器状态

首先建议您检查机器状态，确保所有的Storage服务连接到了Meta服务。执行命令 `SHOW HOSTS` 查看机器状态。

```
nebula> SHOW HOSTS;
+-------------+-----------+-----------+--------------+----------------------+------------------------+
| Host        | Port      | Status    | Leader count | Leader distribution  | Partition distribution |
+-------------+-----------+-----------+--------------+----------------------+------------------------+
| "storaged0" | 9779      | "ONLINE"  | 0            | "No valid partition" | "No valid partition"   |
+-------------+-----------+-----------+--------------+----------------------+------------------------+
| "storaged1" | 9779      | "ONLINE"  | 0            | "No valid partition" | "No valid partition"   |
+-------------+-----------+-----------+--------------+----------------------+------------------------+
| "storaged2" | 9779      | "ONLINE"  | 0            | "No valid partition" | "No valid partition"   |
+-------------+-----------+-----------+--------------+----------------------+------------------------+
| "Total"     | __EMPTY__ | __EMPTY__ | 0            | __EMPTY__            | __EMPTY__              |
+-------------+-----------+-----------+--------------+----------------------+------------------------+
Got 4 rows (time spent 1061/2251 us)
```

在返回结果中，查看**Status**列，可以看到所有Storage服务都在线。

异步实现创建和修改

Nebula Graph中执行如下创建和修改操作，是异步实现的，需要在下一个心跳周期才同步数据。

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

> **说明**：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 [#5.configurations-and-logs/1.configurations/1.configurations/:]。

为确保数据同步，后续操作能顺利进行，可采取以下方法之一：

- 执行`SHOW`或`DESCRIBE`命令检查相应对象的状态，确保创建或修改已完成。如果没有完成，请等待几秒重试。
- 等待2个心跳周期（20秒）。

2.6.3 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}];
```

参数	说明
partition_num	指定图空间的分片数量。建议设置为5倍的集群硬盘数量。例如集群中有3个硬盘，建议设置15个分片。
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。由于需要进行基于quorum的选举，副本数量必须是**奇数**。
vid_type	指定点ID的数据类型。可选值为`FIXED_STRING(<N>)`和`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。默认值为`FIXED_STRING(8)`。

- 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 nba 的图空间。

```
ngql
nebula> CREATE SPACE nba(partition_num=15, replica_factor=1, vid_type=fixed_string(30));
 Execution succeeded (time spent 2817/3280 us)
```

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      | Status    | Leader count | Leader distribution | Partition distribution |
+-------------+-----------+-----------+--------------+----------------------+------------------------+
| "storaged0" | 9779      | "ONLINE"  | 5            | "nba:5"             | "nba:5"                |
+-------------+-----------+-----------+--------------+----------------------+------------------------+
| "storaged1" | 9779      | "ONLINE"  | 5            | "nba:5"             | "nba:5"                |
+-------------+-----------+-----------+--------------+----------------------+------------------------+
| "storaged2" | 9779      | "ONLINE"  | 5            | "nba:5"             | "nba:5"                |
+-------------+-----------+-----------+--------------+----------------------+------------------------+
| "Total"     | __EMPTY__ | __EMPTY__ | 15           | "nba:15"            | "nba:15"               |
+-------------+-----------+-----------+--------------+----------------------+------------------------+
Got 4 rows (time spent 1633/2867 us)
```

   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[#8.service-tuning/load-balance/:]。

3. 选择图空间 nba 。

```
nebula[(none)]> USE nba;
Execution succeeded (time spent 1229/2318 us)
```

   您可以执行命令 SHOW SPACES 查看创建的图空间。

```
nebula> SHOW SPACES;
+-------+
| Name  |
+-------+
| "nba" |
+-------+
Got 1 rows (time spent 977/2000 us)
```

2.6.4 创建标签和边类型

nGQL语法

```
CREATE {TAG | EDGE} {<tag_name> | <edge_type>}(<property_name> <data_type>
[, <property_name> <data_type> ...]);
```

示例

创建标签 player 和 team ，以及边类型 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);
Execution succeeded (time spent 20708/22071 us)

Wed, 24 Feb 2021 03:47:01 EST

nebula> CREATE TAG team(name string);
Execution succeeded (time spent 5643/6810 us)

Wed, 24 Feb 2021 03:47:59 EST

nebula> CREATE EDGE follow(degree int);
Execution succeeded (time spent 12665/13934 us)

Wed, 24 Feb 2021 03:48:07 EST

nebula> CREATE EDGE serve(start_year int, end_year int);
Execution succeeded (time spent 5858/6870 us)

Wed, 24 Feb 2021 03:48:16 EST
```

2.6.5 插入点和边

您可以使用 `INSERT` 语句，基于现有的标签插入点，或者基于现有的边类型插入边。

nGQL语法

- 插入点

  ```
  INSERT VERTEX <tag_name> (<property_name>[, <property_name>...])
  [, <tag_name> (<property_name>[, <property_name>...]), ...]
  {VALUES | VALUE} <vid>: (<property_value>[, <property_value>...])
  [, <vid>: (<property_value>[, <property_value>...];
  ```

  `VID` 是Vertex ID的缩写， `VID` 在一个图空间中是唯一的。

- 插入边

  ```
  INSERT EDGE <edge_type> (<property_name>[, <property_name>...])
  {VALUES | VALUE} <src_vid> -> <dst_vid>[@<rank>] : (<property_value>[, <property_value>...])
  [, <src_vid> -> <dst_vid>[@<rank> : (<property_name>[, <property_name>...]), ...];
  ```

示例

- 插入代表NBA球员和球队的点。

  ```
  ```

```
nebula> INSERT VERTEX player(name, age) VALUES "player100":("Tim Duncan", 42);
Execution succeeded (time spent 28196/30896 us)

Wed, 24 Feb 2021 03:55:08 EST

nebula> INSERT VERTEX player(name, age) VALUES "player101":("Tony Parker", 36);
Execution succeeded (time spent 2708/3834 us)

Wed, 24 Feb 2021 03:55:20 EST

nebula> INSERT VERTEX player(name, age) VALUES "player102":("LaMarcus Aldridge", 33);
Execution succeeded (time spent 1945/3294 us)

Wed, 24 Feb 2021 03:55:32 EST

nebula> INSERT VERTEX team(name) VALUES "team200":("Warriors"), "team201":("Nuggets");
Execution succeeded (time spent 2269/3310 us)

Wed, 24 Feb 2021 03:55:47 EST
```

- 插入代表NBA球员和球队之间关系的边。

```
nebula> INSERT EDGE follow(degree) VALUES "player100" -> "player101":(95);
Execution succeeded (time spent 3362/4542 us)

Wed, 24 Feb 2021 03:57:36 EST

nebula> INSERT EDGE follow(degree) VALUES "player100" -> "player102":(90);
Execution succeeded (time spent 2974/4274 us)

Wed, 24 Feb 2021 03:57:44 EST

nebula> INSERT EDGE follow(degree) VALUES "player102" -> "player101":(75);
Execution succeeded (time spent 1891/3096 us)

Wed, 24 Feb 2021 03:57:52 EST

nebula> INSERT EDGE serve(start_year, end_year) VALUES "player100" -> "team200":(1997, 2016), "player101" -> "team201":
(1999,  2018);
Execution succeeded (time spent 6064/7104 us)

Wed, 24 Feb 2021 03:58:01 EST
```

2.6.6 查询数据

- GO [#3.ngql-guide/7.general-query-statements/3.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 [#3.ngql-guide/7.general-query-statements/4.fetch/:]语句可以获得点或边的属性。
- LOOKUP [#3.ngql-guide/7.general-query-statements/5.lookup/:]语句是基于索引 [#2.quick-start/4.nebula-graph-crud/:_14]的，和 `WHERE` 子句一起使用，查找符合特定条件的数据。
- MATCH [#3.ngql-guide/7.general-query-statements/2.match/:]语句是查询图数据最常用的，但是它依赖索引 [#2.quick-start/4.nebula-graph-crud/:_14]去匹配Nebula Graph中的数据模型。

nGQL语法

- `GO`

```
GO [[<M> TO] <N> STEPS ] FROM <vertex_list>
OVER <edge_type_list> [REVERSELY] [BIDIRECT]
[WHERE <expression> [AND | OR expression ...])]
YIELD [DISTINCT] <return_list>;
```

- `FETCH`
  - 查询标签属性

```
FETCH PROP ON {<tag_name> | <tag_name_list> | *} <vid_list>
[YIELD [DISTINCT] <return_list>];
```

  - 查询边属性

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

- `LOOKUP`

```
LOOKUP ON {<tag_name> | <edge_type>}
WHERE <expression> [AND expression ...])]
[YIELD <return_list>];
```

- `MATCH`

```
MATCH <pattern> [<WHERE clause>] RETURN <output>;
```

GO语句示例

- 从VID为 `player100` 的球员开始，沿着边 `follow` 找到连接的球员。

```
nebula> GO FROM "player100" OVER follow;
+-------------+
| follow._dst |
+-------------+
| "player101" |
+-------------+
| "player102" |
```

```
+-------------+
Got 2 rows (time spent 12097/14220 us)
```

- 从VID为 `player100` 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 "player100" OVER follow WHERE $$.player.age >= 35 \
                ->     YIELD $$.player.name AS Teammate, $$.player.age AS Age;
+---------------+-----+
| Teammate      | Age |
+---------------+-----+
| "Tony Parker" | 36  |
+---------------+-----+
Got 1 rows (time spent 8206/9335 us)
```

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 `player100` 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 "player100" OVER follow YIELD follow._dst AS id | \
                GO FROM $-.id OVER serve YIELD $$.team.name AS Team, \
                $^.player.name AS Player;
+-----------+---------------+
| Team      | Player        |
+-----------+---------------+
| "Nuggets" | "Tony Parker" |
+-----------+---------------+
Got 1 rows (time spent 5055/8203 us)
```

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

> **说明**：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。

```
ngql
 nebula> $var = GO FROM "player100" OVER follow YIELD follow._dst AS id; \
 GO FROM $var.id OVER serve YIELD $$.team.name AS Team, \
 $^.player.name AS Player;
 +---------+-------------+
 | Team    | Player      |
 +---------+-------------+
 | Nuggets | Tony Parker |
 +---------+-------------+
 Got 1 rows (time spent 3103/3711 us)
```

FETCH语句示例

查询VID为 `player100` 的球员的属性。

```
nebula> FETCH PROP ON player "player100";
+-------------------------------------------------+
| vertices_                                       |
+-------------------------------------------------+
| ("player100" :player{age: 42, name: "Tim Duncan"}) |
+-------------------------------------------------+
Got 1 rows (time spent 2006/2406 us)
```

> **说明**： `LOOKUP` 和 `MATCH` 的示例在下文的索引 [#2.quick-start/4.nebula-graph-crud/:_14]部分查看。

2.6.7 修改点和边

您可以使用 `UPDATE` 语句或 `UPSERT` 语句修改现有数据。

UPSERT 是 UPDATE 和 INSERT 的结合体。当您使用 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。

> **说明**： UPSERT 操作是基于Nebula Graph的分区进行串行操作，所以执行速度比单独执行 INSERT 或 UPDATE 慢。

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。

  ```ngql nebula> UPDATE VERTEX "player100" SET player.name = "Tim"; Execution succeeded (time spent 3483/3914 us)

  Wed, 21 Oct 2020 10:53:14 UTC
  ```

nebula> FETCH PROP ON player "player100"; +----------+ | vertices_ | +----------+ | ("player100" :player{age: 42, name: "Tim"}) | +----------+ Got 1 rows (time spent 2463/3042 us) ```

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

```
nebula> UPDATE EDGE "player100" -> "player101" OF follow SET degree = 96;
Execution succeeded (time spent 3932/4432 us)

nebula> FETCH PROP ON follow "player100" -> "player101";
+-------------------------------------------------+
| edges_                                          |
+-------------------------------------------------+
| [:follow "player100"->"player101" @0 {degree: 96}] |
+-------------------------------------------------+
Got 1 rows (time spent 2205/2800 us)
```

- 用 `UPSERT` 插入一个VID为 `player111` 的点。

```
nebula> INSERT VERTEX player(name, age) VALUES "player111":("Ben Simmons", 22);
Execution succeeded (time spent 2115/2900 us)

Wed, 21 Oct 2020 11:11:50 UTC

nebula> UPSERT VERTEX "player111" SET player.name = "Dwight Howard", player.age = $^.player.age + 11 \
WHEN $^.player.name == "Ben Simmons" AND $^.player.age > 20 \
YIELD $^.player.name AS Name, $^.player.age AS Age;
+---------------+-----+
| Name          | Age |
+---------------+-----+
| Dwight Howard | 33  |
+---------------+-----+
Got 1 rows (time spent 1815/2329 us)
```

2.6.8 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 "team1", "team2";
Execution succeeded (time spent 4337/4782 us)
```

- 删除边

```
nebula> DELETE EDGE follow "team1" -> "team2";
Execution succeeded (time spent 3700/4101 us)
```

2.6.9 索引

您可以通过CREATE INDEX [#3.ngql-guide/14.native-index-statements/1.create-native-index/:]语句为标签（tag）和边类型（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，除非您很清楚使用索引对业务的影响。
- 您**必须**为已存在的数据重建索引，否则不能索引已存在的数据，导致无法在 `MATCH` 和 `LOOKUP` 语句中返回这些数据。更多信息，请参见重建索引 [#3.ngql-guide/14.native-index-statements/1.create-native-index/:]。

nGQL语法

- 创建索引

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

- 重建索引

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

示例

为标签 `player` 的属性 `name` 创建索引，并且重建索引。

```
nebula> CREATE TAG INDEX player_index_0 on player(name(20));
nebula> REBUILD TAG INDEX player_index_0;
```

> **说明**：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 [#3.ngql-guide/14.native-index-statements/1.create-native-index/:_6]。

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。

找到标签为 `player` 的点的信息，它的 `name` 属性值为 `Tony Parker` 。

```
// 为name属性创建索引player_name_0。
nebula> CREATE TAG INDEX player_name_0 on player(name(10));
Execution succeeded (time spent 3465/4150 us)

// 重建索引确保能对已存在数据生效。
nebula> REBUILD TAG INDEX player_name_0
+------------+
| New Job Id |
+------------+
| 31         |
+------------+
Got 1 rows (time spent 2379/3033 us)

// 使用LOOKUP语句检索点的属性。
nebula> LOOKUP ON player WHERE player.name == "Tony Parker" \
YIELD player.name, player.age;
+-------------+---------------+------------+
| VertexID    | player.name   | player.age |
+-------------+---------------+------------+
| "player101" | "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"}) |
+-------------------------------------------------------+
Got 1 rows (time spent 5132/6246 us)
```

2.7 常用链接

2.7.1 API

链接	commit id
C++ [https://github.com/vesoft-inc/nebula-cpp/tree/v2.0.0]	7305c72
Go [https://github.com/vesoft-inc/nebula-go/tree/release-v2.0.0-ga]	542ed24
Python [https://github.com/vesoft-inc/nebula-python/releases/tag/v2.0.0]	cb48e8a

以下项目 v2.0.0 还未发布. (TODO Mar. 23 2021):

- Java Client [https://github.com/vesoft-inc/nebula-java]
- Rust Client [https://github.com/vesoft-inc/nebula-rust]
- Node.js Client [https://github.com/vesoft-inc/nebula-node]
- HTTP Client [https://github.com/vesoft-inc/nebula-http-gateway]

2.7.2 工具

链接	commit id
命令行 [https://github.com/vesoft-inc/nebula-console/tree/v2.0.0-ga]	1f32236

以下项目 v2.0.0 还未发布. (TODO Mar. 23 2021):

- Studio [https://github.com/vesoft-inc/nebula-web-docker]
- Dashboard [https://github.com/vesoft-inc/nebula-stats-exporter]
- [Graph Computing TODO]

2.7.3 大数据对接

链接	commit id
csv (importer) [https://github.com/vesoft-inc/nebula-importer/tree/release-v2.0.0-ga]	1d87c7b
Spark [https://github.com/vesoft-inc/nebula-spark-utils/tree/v2.0.0]	af3fdf4
nebula-docker-compose [https://github.com/vesoft-inc/nebula-docker-compose/tree/v2.0.0]	2c2549a

以下项目 v2.0.0 还未发布. (TODO Mar. 23 2021):

- Flink [https://github.com/vesoft-inc/nebula-flink-connector]
- Promethus [https://github.com/vesoft-inc/nebula-stats-exporter]
- ElasticSearch [#4.deployment-and-installation/6.deploy-text-based-index/2.deploy-es/:]

2.7.4 性能、测试、备份与恢复

以下项目 v2.0.0 还未发布. (TODO Mar. 23 2021):

- Benchmark [https://github.com/vesoft-inc/nebula-bench]
- Chaos Test [https://github.com/vesoft-inc/nebula-chaos]
- Backup&Restore [https://github.com/vesoft-inc/nebula-br]

2.7.5 其他

链接	commit id
Nebula Graph 1.2 (已过期) [https://github.com/vesoft-inc/nebula]	53f56b6

- 社区 [https://github.com/vesoft-inc/nebula-community]

3. nGQL指南

3.1 nGQL概述

3.1.1 Nebula Graph查询语言（nGQL）

本文介绍Nebula Graph使用的查询语言nGQL（Nebula Graph Query Language）。

什么是nGQL

nGQL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 [#3.ngql-guide/1.nGQL-overview/3.graph-patterns/:]，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 [https://github.com/vesoft-inc/nebula-graph/issues]通知Nebula Graph团队。Nebula Graph 2.0及更新版本将支持openCypher 9 [https://www.opencypher.org/resources]。

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

您可以下载Nebula Graph示例数据脚本 [https://docs.nebula-graph.io/2.0/nba-2.X.ngql]，然后使用Nebula Graph Console [#2.quick-start/3.connect-to-nebula-graph/:]，使用选项 `-f` 执行脚本。

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：

- ISO/IEC 10646
- ISO/IEC 39075
- ISO/IEC NP 39075 (Draft)
- 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。

nGQL的符号说明如下。

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

例如创建点或边的nGQL语法：

```
CREATE {TAG | EDGE} {<tag_name> | <edge_type>}(<property_name> <data_type>
[, <property_name> <data_type> ...]);
```

示例语句：

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

3.1.2 图建模

本文介绍在Nebula Graph项目中成功应用的一些图建模通用建议。

> **说明**：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。

以性能为目标进行建模

目前Nebula Graph没有完美的建模方法，如何建模取决于您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，您可以多次更改或重新设计模型。

合理设置边属性

- 深度图遍历的性能较低，为了减少遍历深度，请使用点属性代替边。例如，模型a包括姓名、年龄、眼睛颜色三种属性，建议您创建一个标签`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标签和一个边类型`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。
- 为边创建属性时请勿使用长字符串，Nebula Graph支持在边上存储长字符串属性，但是这些属性会同时保存在出边和入边，请小心写入放大（write amplification）。

合理设置标签属性

在图建模中，请将一组类似的平级属性放入同一个标签，即按不同概念进行分组。

正确使用索引

正确使用索引可以加速查询，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。

> **注意**：请**不要随意**在生产环境中使用索引，除非您很清楚使用索引对业务的影响。

3.1.3 模式

模式（pattern）和图模式匹配，是图查询语言的核心，本文介绍Nebula Graph的各种模式。

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)
```

标签模式

> **说明**：nGQL中的 `tag` 概念与openCypher中的 `label` 有一些不同。例如，您必须创建一个 `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，还可以描述点的标签。例如：

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

模式也可以描述有多个标签的点，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)
```

边模式

描述一条边最简单的方法是使用箭头连接两个点。

您可以用以下方式描述边以及它的方向性。如果您不关心边的方向，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)
```

和点上的标签一样，边也可以有类型。描述边的类型，例如：

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

和点上的标签不同，一条边只能有一种边类型。但是如果我们想描述多个可选边类型，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。

您也可以忽略最小长度，只指定最大长度，例如：

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

> **说明**：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)
```

您可以在MATCH语句中使用路径变量。

3.2 数据类型

3.2.1 数值类型

int

Nebula Graph使用关键字 `int` 声明64位带符号整数，支持的范围是[-9223372036854775808, 9223372036854775807]。整数支持多种进制表示：

- 十进制，例如 `123456` 。
- 十六进制，例如 `0x1e240` 。
- 八进制，例如 `0361100` 。

double

Nebula Graph使用关键字 `double` 声明双精度浮点数，例如 `1.2` 、 `-3.0000001` 。同时支持科学计数法，例如 `1e2` 、 `1.1e2` 、 `.3e4` 、 `1.e4` 、 `-1234E-10` 。

3.2.2 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` 。

3.2.3 字符串

Nebula Graph使用关键字 `string` （变长）或 `fixed_string(<length>)` （定长）声明字符串类型数据，格式为双引号或单引号包裹的任意长度的字符串，例如 `"Shaquille O'Neal"` 或 `'This is a double-quoted literal string'` 。

字符串类型

Nebula Graph支持定长字符串和变长字符串。示例如下：

- 定长字符串

```
nebula> CREATE TAG t1 (p1 fixed_string(10));
```

- 变长字符串

```
nebula> CREATE TAG t2 (p2 string);
```

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。

```ngql # 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
```

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

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

3.2.4 日期和时间类型

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

Nebula Graph将当前时区转换为协调世界时间（UTC）并存储，查询时会再次转换回当前时区（ `TIMESTAMP` 例外）。

结合 `RETURN` 子句，函数 `date()` 、 `time()` 和 `datetime()` 可以用空值返回当前的日期或时间。

OpenCypher兼容性

- 支持年、月、日、时、分、秒，不支持毫秒。
- 不支持函数 `localdatetime()` 和 `duration()` 。
- 不支持大部分字符串时间格式，仅支持 `YYYY-MM-DDThh:mm:ss` 。

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` 。

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.ususus` 。支持的范围是 `00:00:00.000` 到 `23:59:59.999` 。

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.ususus` 。支持的范围是 `-32768-01-01T00:00:00.000` 到 `32767-12-31T23:59:59.999` 。

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。
- 底层存储的数据格式为**64位int**。

示例

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

```
nebula> CREATE TAG 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("2021-03-17T17:53:59"));
```

3. 创建标签，名称为 `school` ， 包含 `TIMESTAMP` 类型。

```
nebula> CREATE TAG school(name string , found_time timestamp);
```

4. 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"));
```

5. 插入点，名称为 `dut` ， 用 `now()` 函数存储时间。

```
nebula> INSERT VERTEX school(name, found_time) VALUES "dut":("dut", now());
```

您还可以使用 `WITH` 语句设置具体日期和时间，例如：

```
nebula> WITH time({hour: 12, minute: 31, second: 14}) AS d RETURN d;
+--------------+
| d            |
+--------------+
| 12:31:14.000 |
+--------------+

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

3.2.5 NULL

默认情况下，插入点或边时，属性值可以为 `NULL`，您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	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

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

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

使用 `SHOW` 命令查看创建标签语句，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);
```

使用NOT NULL并设置默认值

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

```
nebula> CREATE TAG 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"
+-----------------------------------------+
| vertices_                               |
+-----------------------------------------+
| ("Kobe" :player{age: 18, name: "Kobe"}) |
+-----------------------------------------+
```

3.2.6 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。

示例

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

# 筛选列表[1,2,3,4,5]中位置下标大于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] |
+--------------+

# 将列表[1,2,3,4,5]中的元素分别做运算，然后返回。
nebula> RETURN [n IN range(1,5) | n + 10] AS a;
+----------------------+
| a                    |
+----------------------+
| [11, 12, 13, 14, 15] |
+----------------------+

# 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。
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([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 follow.degree NOT IN [x IN [92, 90] | x + $$.player.age] \
        YIELD follow._dst AS id, follow.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

- 复合数据类型（例如set、map、list）**不能**存储为点或边的属性。
- 不支持使用 `range()` 函数返回一个列表的部分元素。

```
nebula> RETURN range(0,5)[0..3];
[ERROR (-7)]: SyntaxError: syntax error near `3]'
```

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

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

3.2.7 集合

集合（Set）是复合数据类型。

OpenCypher兼容性

在OpenCypher中，集合不是一个数据类型，而在nGQL中，集合仍在设计阶段。

3.2.8 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。您可以通过 `map['<key>']` 的方法获取映射中的元素。

字面值映射

```
nebula> YIELD {key: 'Value', listKey: [{inner: 'Map1'}, {inner: 'Map2'}]}
+-----------------------------------------------------------+
| {key:Value,listKey:[{inner:Map1},{inner:Map2}]}           |
+-----------------------------------------------------------+
| {key: "Value", listKey: [{inner: "Map1"}, {inner: "Map2"}]} |
+-----------------------------------------------------------+
```

OpenCypher兼容性

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

3.2.9 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。

遗留兼容问题

nGQL 1.0使用C语言风格的类型转换（显示或隐式）：`(type_name)expression`。例如`YIELD (int)(TRUE)`，结果为`1`。但是对于不熟悉C语言的用户来说，很容易出错。

nGQL 2.0使用openCypher的方式进行类型强制转换。

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。
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(id(a)) == 100 RETURN a;
+--------------------------------------------+
| a                                          |
+--------------------------------------------+
| ("100" :player{age: 42, name: "Tim Duncan"}) |
+--------------------------------------------+

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

3.3 变量和复合查询

3.3.1 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语句）连接各个子句，让它们在彼此之间提供中间结果集。
- （nGQL扩展）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。
- （nGQL扩展）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语句和nGQL扩展语句，例如 `MATCH ... | GO ... | YIELD ...`，混用两种语句，行为是未定义的。

- 如果您使用OpenCypher语句（ `MATCH` 、 `RETURN` 、 `WITH` 等），请不要使用管道符或分号组合子句。
- 如果您使用nGQL扩展语句（ `FETCH` 、 `GO` 、 `LOOKUP` 等），必须使用管道符或分号组合子句。

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。

> **说明**：openCypher没有事务要求。

示例

- OpenCypher语句

```
# 子句连接多个查询。
nebula> MATCH p=(v:player{name:"Tim Duncan"})--() \
        WITH nodes(p) AS n \
```

```
        UNWIND n AS n1 \
        RETURN DISTINCT n1;
```

- nGQL扩展（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);
```

- nGQL扩展（管道符）

```
# 管道符连接多个查询。
nebula> GO FROM "player100" OVER follow YIELD follow._dst AS id | \
    GO FROM $-.id OVER serve YIELD $$.team.name AS Team, \
    $^.player.name AS Player;
    +---------+-------------+
    | Team    | Player      |
    +---------+-------------+
    | Nuggets | Tony Parker |
    +---------+-------------+
```

3.3.2 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

当您引用一个变量的点、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。

nGQL扩展

nGQL扩展的自定义变量可以表示为 `$<var_name>`， `var_name` 由字母或数字构成，不允许使用其他字符。

自定义变量仅在当前执行有效，执行结束后变量也会释放，**不能**在其他客户端或执行中使用之前的自定义变量。

您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 [#3.ngql-guide/4.variable-and-composite-queries/1.composite-queries/:]。

> **说明**：自定义变量区分大小写。

示例

```
nebula> $var = GO FROM "player100" OVER follow YIELD follow._dst AS id; \
GO FROM $var.id OVER serve YIELD $$.team.name AS Team, \
$^.player.name AS Player;
+---------+-------------+
| Team    | Player      |
+---------+-------------+
| Nuggets | Tony Parker |
+---------+-------------+
```

3.3.3 引用属性

您可以在 `WHERE` 和 `YIELD` 子句中引用点或边的属性。

> **说明**：本功能仅适用于nGQL扩展。

引用点的属性

起始点

```
$^.<tag_name>.<prop_name>
```

参数	说明
`$^`	表示起始点。
`tag_name`	点的标签名称。
`prop_name`	标签内的属性名称。

目的点

```
$$.tag_name.prop_name
```

参数	说明
`$$`	表示目的点。
`tag_name`	点的标签名称。
`prop_name`	标签内的属性名称。

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`	边类型。
`prop_name`	边类型的属性名称。

引用内置的边属性

除了自定义的边属性，每条边还有如下三种内置属性：

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。
`_rank`	边的rank值。

示例

```
# 返回起始点的标签player的name属性值和目的点的标签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" | 33     |
+--------------+--------+

# 返回边类型follow的degree属性值。
nebula> GO FROM "player100" OVER follow YIELD follow.degree;
+---------------+
| follow.degree |
+---------------+
| 95            |
+---------------+
| 90            |
+---------------+

# 返回边类型follow的起始点、目的点、边类型编码和边rank值。
```ngql
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" | 136          | 0            |
+-------------+-------------+--------------+--------------+
| "player100" | "player102" | 136          | 0            |
+-------------+-------------+--------------+--------------+
```
```

3.4 运算符

3.4.1 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
<	小于
<=	小于等于
%	取模
-	负数符号
IS NULL	为NULL
IS NOT NULL	不为NULL

比较操作的结果是 `true` 或者 `false` 。

> **说明**：比较不同类型的值通常没有定义，结果可能是NULL或其它。

OpenCypher兼容性

Nebula Graph中，NULL的比较操作和openCypher不同，行为也可能会改变。在openCypher中， `IS [NOT] NULL` 通常与 `OPTIONAL MATCH` 一起使用，但是nGQL不支持 `OPTIONAL MATCH` 。

示例

==

字符串比较时，会区分大小写。不同类型的值不相等。

> **说明**：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__ |
+--------+----------+----------+

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 |
+--------------+------------+-------------+--------------+------------------+-----------+-------------+------------------+
| BAD_TYPE     | __NULL__   | 0           | true         | false            | BAD_TYPE  | __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.age IS NULL, n.name IS NOT NULL, n.empty IS NULL;
+---------------+--------------------+-----------------+
| n.age IS NULL | n.name IS NOT NULL | n.empty IS NULL |
+---------------+--------------------+-----------------+
| false         | true               | true            |
+---------------+--------------------+-----------------+
| false         | true               | true            |
+---------------+--------------------+-----------------+
| false         | true               | true            |
+---------------+--------------------+-----------------+
...
```

3.4.2 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 [#3.ngql-guide/5.operators/9.precedence/:]。

对于带有 `NULL` 的逻辑运算，请参见NULL [#3.ngql-guide/3.data-types/5.null/:]。

历史兼容问题

在Nebula Graph 2.0中，非0数字不能转换为布尔值。

3.4.3 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nGQL扩展。

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 follow._dst AS dstid, $$.player.name AS Name | \
GO FROM $-.dstid OVER follow;

+-------------+
| follow._dst |
+-------------+
| "player101" |
+-------------+
```

您可以使用 `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。

您必须在 `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`。

3.4.4 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nGQL扩展。

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 [#3.ngql-guide/4.variable-and-composite-queries/3.property-reference/:]。
`$$`	引用目的点。更多信息请参见引用属性 [#3.ngql-guide/4.variable-and-composite-queries/3.property-reference/:]。
`$-`	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 [#3.ngql-guide/5.operators/4.pipe/:]。

示例

```
# 返回起始点和目的点的年龄。
nebula> GO FROM "player100" OVER follow YIELD $^.player.name AS SrcAge, $$.player.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 follow._dst AS id | \
        GO FROM $-.id OVER serve \
        YIELD $^.player.name AS Player, $$.team.name AS Team;
+-----------------+-----------+
| Player          | Team      |
+-----------------+-----------+
| "Tony Parker"   | "Spurs"   |
+-----------------+-----------+
| "Tony Parker"   | "Hornets" |
+-----------------+-----------+
| "Manu Ginobili" | "Spurs"   |
+-----------------+-----------+
```

3.4.5 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nGQL扩展。

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 [#3.ngql-guide/3.data-types/9.type-conversion/:]。

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 \
    UNION \
    GO FROM "player100" OVER follow;
+-------------+
| follow._dst |
+-------------+
| "player101" |
+-------------+
| "player102" |
+-------------+

# 返回两个查询结果的并集，包含重复的元素。
nebula> GO FROM "player102" OVER follow \
    UNION ALL \
    GO FROM "player100" OVER follow;
+-------------+
| follow._dst |
+-------------+
| "player101" |
+-------------+
| "player101" |
+-------------+
| "player102" |
+-------------+

# 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 follow._dst AS id, follow.degree AS Degree, $$.player.age AS Age \
        UNION /* DISTINCT */ \
        GO FROM "player100" OVER follow \
        YIELD follow._dst AS id, follow.degree AS Degree, $$.player.age AS Age;
```

```
+-------------+--------+-----+
| id          | Degree | Age |
+-------------+--------+-----+
| "player101" | 75     | 36  |
+-------------+--------+-----+
| "player101" | 96     | 36  |
+-------------+--------+-----+
| "player102" | 90     | 33  |
+-------------+--------+-----+
```

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 [#3.ngql-guide/3.data-types/9.type-conversion/:]。

示例

```
nebula> GO FROM "player102" OVER follow \
        YIELD follow._dst AS id, follow.degree AS Degree, $$.player.age AS Age \
        INTERSECT \
        GO FROM "player100" OVER follow \
        YIELD follow._dst AS id, follow.degree AS Degree, $$.player.age AS Age;
Empty set (time spent 2990/3511 us)
```

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 \
        MINUS \
        GO FROM "player102" OVER follow;
+-------------+
| follow._dst |
+-------------+
| "player102" |
+-------------+

nebula> GO FROM "player102" OVER follow \
        MINUS \
        GO FROM "player100" OVER follow;
Empty set (time spent 2243/3259 us)
```

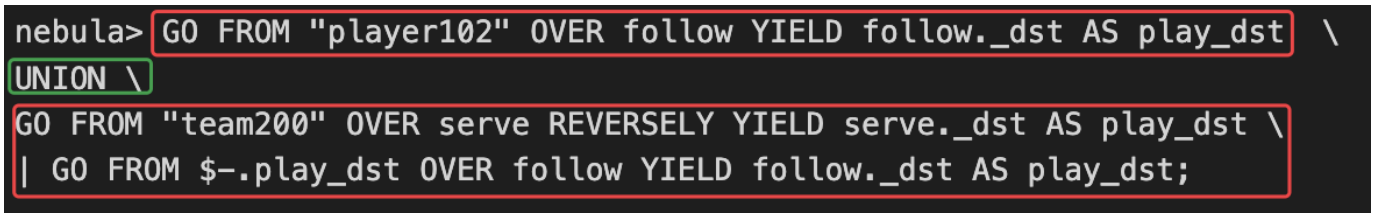
集合运算符和管道符的优先级

当查询包含集合运算符和管道符（|）时，管道符 [#3.ngql-guide/5.operators/4.pipe/:]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 follow._dst AS play_dst  \
        UNION \
        GO FROM "team200" OVER serve REVERSELY \
        YIELD serve._dst AS play_dst \
        | GO FROM $-.play_dst OVER follow YIELD follow._dst AS play_dst;

+-------------+
| play_dst    |
+-------------+
| "player101" |
+-------------+
| "player102" |
+-------------+
```



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 follow._dst AS play_dst  \
        UNION \
        GO FROM "team200" OVER serve REVERSELY \
        YIELD serve._dst AS play_dst) \
        | GO FROM $-.play_dst OVER follow YIELD follow._dst AS play_dst;
```

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。

3.4.6 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。

> **说明**：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.name CONTAINS "ets" RETURN s.name, e.start_year, e.end_year, t.name;
+---------------+--------------+------------+-----------+
| s.name        | e.start_year | e.end_year | t.name    |
+---------------+--------------+------------+-----------+
| "Tony Parker" | 2018         | 2019       | "Hornets" |
+---------------+--------------+------------+-----------+

nebula> GO FROM "player101" OVER serve WHERE (STRING)serve.start_year CONTAINS "19" AND \
    $^.player.name CONTAINS "ny" \
    YIELD $^.player.name, serve.start_year, serve.end_year, $$.team.name;
+---------------+-----------------+---------------+--------------+
| $^.player.name | serve.start_year | serve.end_year | $$.team.name |
+---------------+-----------------+---------------+--------------+
| "Tony Parker"  | 1999             | 2018           | "Spurs"      |
+---------------+-----------------+---------------+--------------+
```

```
nebula> GO FROM "player101" OVER serve WHERE !($$.team.name CONTAINS "ets") \
    YIELD $^.player.name, serve.start_year, serve.end_year, $$.team.name;
+---------------+-----------------+---------------+-------------+
| $^.player.name | serve.start_year | serve.end_year | $$.team.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"]) |
+---------------+-----------------------------------+-------------------------------+
| true           | true                              | false                         |
+---------------+-----------------------------------+-------------------------------+
```

(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

> **说明**：当前仅openCypher语句（`MATCH`、`WITH`等）支持正则表达式，nGQL扩展语句（`FETCH`、`GO`、`LOOKUP`等）暂不支持正则表达式。

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`，您可以使用语法`=~ '<regexp>'`进行正则表达式匹配配。例如：

```
nebula> RETURN "384748.39" =~ "\\d+(\\.\\d{2})?";
+----------------------------+
| (384748.39=~\d+(\.\d{2})?) |
+----------------------------+
| true                        |
+----------------------------+

nebula> MATCH (v:player) WHERE v.name =~ 'Tony.*' RETURN v.name;
+--------------+
```

```
| v.name        |
+---------------+
| "Tony Parker" |
+---------------+
```

3.4.7 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]) |
+--------------------+
| true               |
+--------------------+

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

3.4.8 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`。

3.5 函数和表达式

3.5.1 内置数学函数

Nebula Graph支持以下内置数学函数。

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。 您可以只传入一个参数，该参数会判定为 `max`，此时 `min` 默认为 `0`。 如果您不传入参数，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。 您可以只传入一个参数，该参数会判定为 `max`，此时 `min` 默认为 `0`。 如果您不传入参数，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。
map(fun, iter)	将给定函数应用于给定可迭代对象的每一项后，返回一个映射对象。
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。

> **说明**：如果参数为 `NULL`，则输出结果是未定义的。

3.5.2 内置字符串函数

Nebula Graph支持以下内置字符串函数。

> **说明**：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。

> **说明**：如果参数为`NULL`，则输出结果是未定义的。

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 [https://github.com/vesoft-inc/nebula-graph/issues/878]。

> **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，In openCypher, if pos is 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。

3.5.3 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。
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）。

`date()` 、 `time()` 和 `datetime()` 函数除了传入空值获取当前时间或日期，还接受string和map类型的参数。

openCypher兼容性

- 在openCypher中，时间精确到毫秒。
- 在nGQL中，时间精确到秒。毫秒数显示为 `000` 。

示例

```
> RETURN now(), date(), time(), datetime();
+------------+------------+--------------+-------------------------+
| now()      | date()     | time()       | datetime()              |
+------------+------------+--------------+-------------------------+
| 1611907165 | 2021-01-29 | 07:59:22.000 | 2021-01-29T07:59:22.000 |
+------------+------------+--------------+-------------------------+
```

3.5.4 Schema函数

Nebula Graph支持以下Schema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。
list tags(vertex)	返回点的标签。
list labels(vertex)	返回点的标签。
map properties(vertex_or_edge)	接收点或边并返回其属性。
string type(edge)	返回边的边类型。
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。

示例

```
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       |
+---------+---------+

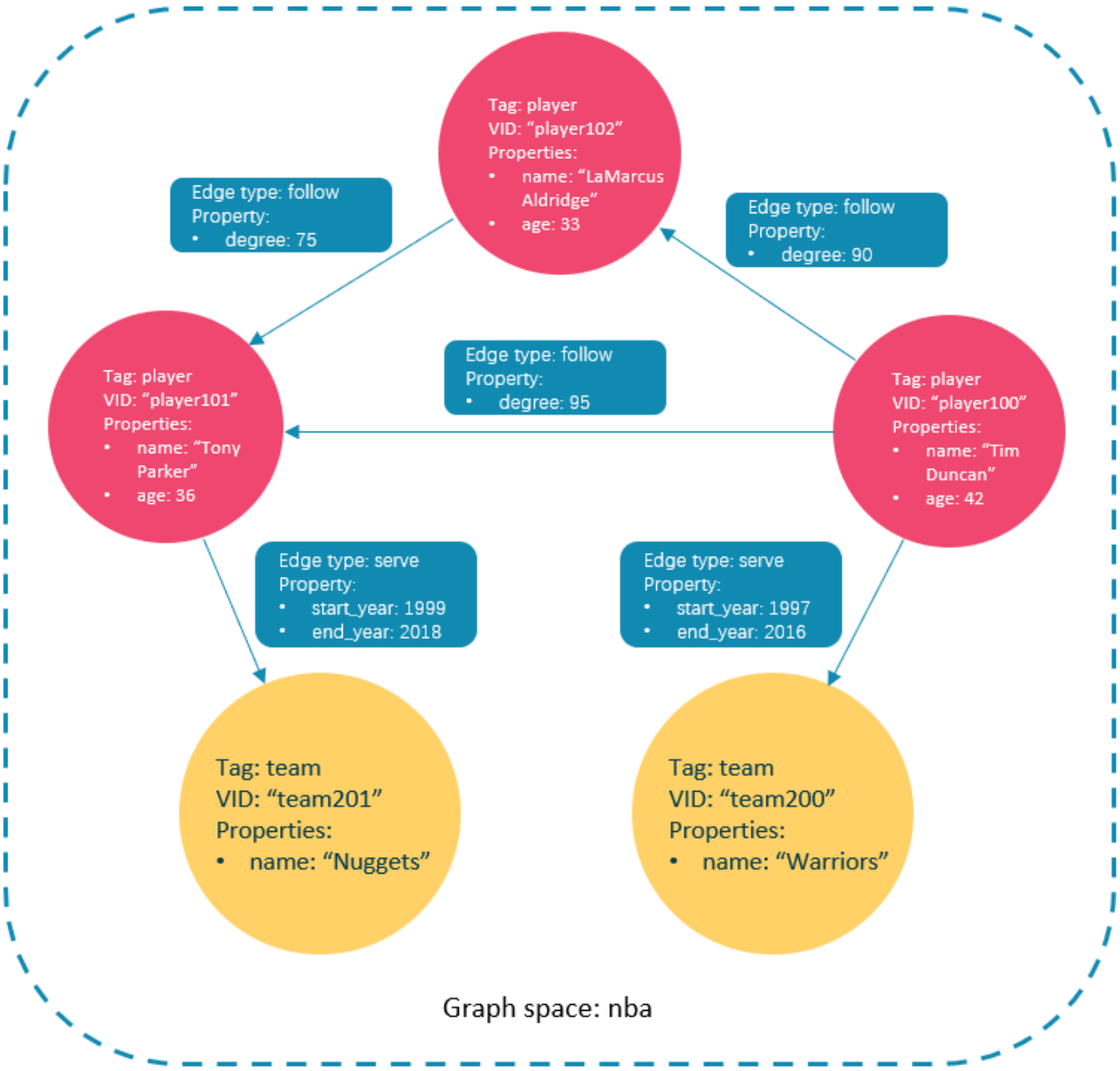
nebula> MATCH p = (a :player {name : "Tim Duncan"})-[r:serve]-(t) \
        RETURN startNode(p), endNode(p);
+--------------------------------------------------+----------------------------------+
| startNode(p)                                     | endNode(p)                       |
+--------------------------------------------------+----------------------------------+
| ("player100" :player{age: 42, name: "Tim Duncan"}) | ("team204" :team{name: "Spurs"}) |
+--------------------------------------------------+----------------------------------+
```

3.5.5 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` 。

下图为本文的示例图。



简单形式

语法

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

> 警告： `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 $$.player.name AS Name, \
        CASE $$.player.age > 35 \
        WHEN true THEN "Yes" \
        WHEN false THEN "No" \
        ELSE "Nah" \
        END \
        AS Age_above_35;
+---------------------+--------------+
| Name                | Age_above_35 |
+---------------------+--------------+
| "Tony Parker"       | "Yes"        |
+---------------------+--------------+
| "LaMarcus Aldridge" | "No"         |
+---------------------+--------------+
```

通用形式

语法

```
```

```
CASE
WHEN <condition> THEN <result>
[WHEN ...]
[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.age > 30 \
        RETURN v.name AS Name,  \
        CASE \
        WHEN v.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，您需要了解它们的差异。请参见如下示例：

```
nebula> GO FROM "player100" OVER follow \
        YIELD $$.player.name AS Name, $$.player.age AS Age, \
        CASE $$.player.age \
        WHEN $$.player.age > 35 THEN "Yes" \
        ELSE "No" \
        END \
        AS Age_above_35;
+---------------------+-----+--------------+
| Name                | Age | Age_above_35 |
+---------------------+-----+--------------+
```

```
| "Tony Parker"       | 36  | "No"         |
+---------------------+-----+--------------+
| "LaMarcus Aldridge" | 33  | "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` 。

3.5.6 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)	返回点的标签列表。
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 [#3.ngql-guide/6.functions-and-expressions/11.reduce/:]。

> **说明**：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);
+----------------------------------+-------------------------+-----------+-----------+---------------+
| reverse(ids)                     | tail(ids)               | head(ids) | last(ids) | coalesce(ids) |
+----------------------------------+-------------------------+-----------+-----------+---------------+
| [487, 521, "abc", 4923, __NULL__] | [4923, "abc", 521, 487] | __NULL__  | 487       | 4923          |
+----------------------------------+-------------------------+-----------+-----------+---------------+

nebula> MATCH (a:player)-[r]->() \
        WHERE id(a) == "player100" \
        RETURN labels(a),  keys(r);
+------------+---------------------------+
| labels(a)  | keys(r)                   |
+------------+---------------------------+
| ["player"] | ["degree"]                |
+------------+---------------------------+
| ["player"] | ["degree"]                |
+------------+---------------------------+
| ["player"] | ["end_year", "start_year"] |
+------------+---------------------------+

nebula> MATCH p = (a:player)-[]->(b)-[]->(c:team) \
        WHERE a.name == "Tim Duncan" AND c.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.name == "Tim Duncan" AND c.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}]] |
+-------------------------------------------------------------------------------------------------------------------------------------------------+
```

3.5.7 count函数

`count()` 函数可以计数指定的值或行数。

- （nGQL扩展）您可以同时使用 `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）您可以使用 `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 $$.player.name AS Name \
        | GROUP BY $-.Name YIELD $-.Name, count(*);
+---------------------+----------+
| $-.Name             | COUNT(*) |
+---------------------+----------+
| "Dejounte Murray"   | 1        |
+---------------------+----------+
| "LaMarcus Aldridge" | 2        |
+---------------------+----------+
| "Tim Duncan"        | 2        |
+---------------------+----------+
| "Marco Belinelli"   | 1        |
+---------------------+----------+
| "Manu Ginobili"     | 1        |
+---------------------+----------+
| "Boris Diaw"        | 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。

因为测试数据库 `nba` 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.age as age, count(*) as number \
        ORDER BY number DESC, age DESC;
+-----+--------+
| age | number |
+-----+--------+
| 34  | 4      |
+-----+--------+
| 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                   |
+------------+---------------------+
```

3.5.8 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.age);
+-------------------------------------------------------------+
| COLLECT(n.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.age AS age, collect(n.name);
+-----+-------------------------------------------------------------------------+
| age | collect(n.name)                                                         |
+-----+-------------------------------------------------------------------------+
| 24  | ["Giannis Antetokounmpo"]                                               |
+-----+-------------------------------------------------------------------------+
| 20  | ["Luka Doncic"]                                                         |
+-----+-------------------------------------------------------------------------+
| 25  | ["Joel Embiid", "Kyle Anderson"]                                        |
+-----+-------------------------------------------------------------------------+
...
```

3.5.9 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` 。

> **说明**：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" \
        |  GO FROM $-.VertexID over follow \
        WHERE follow.degree != reduce(totalNum = 5, n IN range(1, 3) | $$.player.age + totalNum + n) \
        YIELD $$.player.name AS id, $$.player.age AS age, follow.degree AS degree;
+---------------------+-----+--------+
| id                  | age | degree |
+---------------------+-----+--------+
| "Tim Duncan"        | 42  | 95     |
+---------------------+-----+--------+
| "LaMarcus Aldridge" | 33  | 90     |
+---------------------+-----+--------+
| "Manu Ginobili"     | 41  | 95     |
+---------------------+-----+--------+
```

3.5.10 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` 。您可以在`MurmurHahs2.h` [https://github.com/vesoft-inc/nebula-common/blob/master/src/common/base/MurmurHash2.h]中查看其源代码。

> **说明：** 在有10亿个点的情况下，发生碰撞的概率大约是1/10。边的数量与碰撞的概率无关。

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nGQL 2.0同时支持了字符串和整数类型的VID，所以您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)   |
+----------------------+
| -1098335330293915 40 |
+----------------------+
```

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         |
+------------+
```

计算表达式的哈希值

```
nebula> YIELD hash(toLower("HELLO NEBULA"));
+-------------------------------+
| hash(toLower("HELLO NEBULA")) |
+-------------------------------+
| -8481157362655072082          |
+-------------------------------+
```

3.5.11 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`。

> **说明**：如果列表为空，或者列表中的所有元素都为空，则返回NULL。

openCypher兼容性

在openCypher中，只定义了函数 `exists()`，其他函数依赖工具实现。

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 |
+----------------+------------------+-------+
| "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.id), n IS NOT NULL;
+--------------+---------------+
| exists(n.id) | n IS NOT NULL |
+--------------+---------------+
| false        | true          |
+--------------+---------------+
...
```

3.5.12 自定义函数

openCypher兼容性

Nebula Graph 2.x暂不支持自定义函数，也没有相应计划。

3.6 通用查询语句

3.6.1 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 [#3.ngql-guide/1.nGQL-overview/3.graph-patterns/:]，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库 `nba` 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` 语法可以概括如下：

```
MATCH <pattern> [<WHERE clause>] RETURN <output>
```

MATCH工作流程

1. `MATCH` 语句使用原生索引查找起始点，起始点可以在模式的任何位置。即一个有效的 `MATCH` 语句，**必须有一个属性、标签或者点已经创建索引**。如何创建索引，请参见创建原生索引 [#3.ngql-guide/14.native-index-statements/1.create-native-index/:]。
2. `MATCH` 语句在模式中搜索，寻找匹配的边或点。
3. `MATCH` 语句根据 `RETURN` 子句检索数据。

openCypher兼容性

nGQL暂不支持遍历所有点和边，例如 `MATCH (v) RETURN v` 。

使用模式（pattern）

前提条件

请确保 `MATCH` 语句有至少一个索引可用。如果您需要创建索引，但是已经有相关的点、边或属性，您必须在创建索引后重建索引，索引才能生效。

> **警告**：正确使用索引可以加快查询速度，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，除非您很清楚使用索引对业务的影响。

```
# 在属性name上创建索引。
nebula> CREATE TAG INDEX name ON player(name(20));

# 重建索引使其生效。
nebula> REBUILD TAG INDEX name;
+------------+
| New Job Id |
+------------+
| 121        |
```

```
+------------+

# 确认重建索引成功。
nebula> SHOW JOB 121;
+----------------+---------------------+------------+------------+------------+
| Job Id(TaskId) | Command(Dest)       | Status     | Start Time | Stop Time  |
+----------------+---------------------+------------+------------+------------+
| 121            | "REBUILD_TAG_INDEX" | "FINISHED" | 1607073046 | 1607073046 |
+----------------+---------------------+------------+------------+------------+
| 0              | "storaged2"         | "FINISHED" | 1607073046 | 1607073046 |
+----------------+---------------------+------------+------------+------------+
| 1              | "storaged0"         | "FINISHED" | 1607073046 | 1607073046 |
+----------------+---------------------+------------+------------+------------+
| 2              | "storaged1"         | "FINISHED" | 1607073046 | 1607073046 |
+----------------+---------------------+------------+------------+------------+
```

匹配点

您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)` 。

匹配标签

> **说明**：标签的索引和属性的索引不同。如果标签的某个属性有索引，但是标签本身没有索引，您无法基于该标签执行 `MATCH` 语句。

您可以在点的右侧用 `:<tag_name>` 表示模式中的标签。

```
nebula> MATCH (v:player) RETURN v;
+------------------------------------------------------------+
| v                                                          |
+------------------------------------------------------------+
| ("player102" :player{age: 33, name: "LaMarcus Aldridge"})  |
+------------------------------------------------------------+
| ("player106" :player{age: 25, name: "Kyle Anderson"})      |
+------------------------------------------------------------+
| ("player115" :player{age: 40, name: "Kobe Bryant"})        |
+------------------------------------------------------------+
...
```

匹配点的属性

您可以在标签的右侧用 `{<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{name: "Tim Duncan", age: 42}) |
+-------------------------------------------------+
```

使用 `WHERE` 子句也可以实现相同的操作：

```
nebula> MATCH (v:player) WHERE v.name == "Tim Duncan" RETURN v;
+-------------------------------------------------+
| v                                               |
+-------------------------------------------------+
```

```
| ("player100" :player{name: "Tim Duncan", age: 42}) |
+----------------------------------------------------+
```

> **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

您可以使用点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.name:Tony Parker,player.age:36 |
+-------------------------------------------------+
```

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{name: "Tony Parker", age: 36})       |
+-----------------------------------------------------------+
| ("player102" :player{name: "LaMarcus Aldridge", age: 33}) |
+-----------------------------------------------------------+
| ("player101" :player{name: "Tony Parker", age: 36})       |
+-----------------------------------------------------------+
```

匹配连接的点

您可以使用 `--` 符号表示两个方向的边，并匹配这些边连接的点。

> **说明**：在nGQL 1.x中， `--` 符号用于行内注释，在nGQL 2.x中， `--` 符号表示出边或入边。

```
nebula> MATCH (v:player{name:"Tim Duncan"})--(v2) \
        RETURN v2.name AS Name;
+---------------------+
| Name                |
+---------------------+
| "Tony Parker"       |
+---------------------+
| "LaMarcus Aldridge" |
+---------------------+
| "Marco Belinelli"   |
+---------------------+
| "Danny Green"       |
+---------------------+
| "Aron Baynes"       |
+---------------------+
...
```

您可以在 `--` 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) \
        RETURN v2.name AS Name;
+-----------------+
```

```
| Name            |
+-----------------+
| "Spurs"         |
+-----------------+
| "Tony Parker"   |
+-----------------+
| "Manu Ginobili" |
+-----------------+
```

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.name AS Name;
+---------------------+
| Name                |
+---------------------+
| "Tony Parker"       |
+---------------------+
| "Tiago Splitter"    |
+---------------------+
| "Dejounte Murray"   |
+---------------------+
| "Tony Parker"       |
+---------------------+
| "LaMarcus Aldridge" |
+---------------------+
...
```

如果您不需要引用点，可以省略括号中表示点的变量。

```
nebula> MATCH (v:player{name:"Tim Duncan"})-->()<--(v3) \
        RETURN v3.name AS Name;
+---------------------+
| Name                |
+---------------------+
| "Tony Parker"       |
+---------------------+
| "LaMarcus Aldridge" |
+---------------------+
| "Rudy Gay"          |
+---------------------+
| "Danny Green"       |
+---------------------+
| "Kyle Anderson"     |
+---------------------+
...
```

匹配路径

连接起来的点和边构成了路径。您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"})-[:follow@0 {degree: 95}]->("player101" :player{age: 36, name: "Tony
Parker"})>   |
+-------------------------------------------+
| <("player100" :player{age: 42, name: "Tim Duncan"})-[:follow@0 {degree: 95}]->("player125" :player{age: 41, name: "Manu
```

```
Ginobili"})> |
+-----------------------------------------+
| <("player100" :player{age: 42, name: "Tim Duncan"})-[:serve@0 {end_year: 2016, start_year: 1997}]->("team204" :team{name:
"Spurs"})> |
+-----------------------------------------+
```

> **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

除了用 `--` 、 `-->` 、 `<--` 表示未命名的边之外，您还可以在方括号中使用自定义变量命名边。例如 `-[e]-` 。

```
nebula> MATCH (v:player{name:"Tim Duncan"})-[e]-(v2) \
        RETURN e;
+------------------------------------------------------------------------------+
| e                                                                            |
+------------------------------------------------------------------------------+
| [:follow "player101"->"player100" @0 {degree: 95}]                           |
+------------------------------------------------------------------------------+
| [:follow "player102"->"player100" @0 {degree: 75}]                           |
+------------------------------------------------------------------------------+
| [:serve "player100"->"team204" @0 {end_year: 2016, start_year: 1997}]        |
+------------------------------------------------------------------------------+
...
```

匹配边类型和属性

和点一样，您可以用 `:<edge_type>` 表示模式中的边类型。例如 `-[e:serve]-` 。

```
nebula> MATCH (v:player{name:"Tim Duncan"})-[e:serve]-(v2) \
        RETURN e;
+------------------------------------------------------------------------------+
| e                                                                            |
+------------------------------------------------------------------------------+
| [:serve "player100"->"team204" @0 {end_year: 2016, start_year: 1997}]        |
+------------------------------------------------------------------------------+
```

您可以用 `{<prop_name>: <prop_value>}` 表示模式中边类型的属性。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}]    |
+-------------------------------------------------------+
```

匹配多个边类型

使用 `|` 可以匹配多个边类型。例如 `[e:follow|:serve]` 。

```
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

您可以扩展模式，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                                                        |
+----------------------------------+-----------------------------------------------------------+
| ("player204" :team{name: "Spurs"}) | ("player101" :player{name: "Tony Parker", age: 36})       |
+----------------------------------+-----------------------------------------------------------+
| ("player204" :team{name: "Spurs"}) | ("player102" :player{name: "LaMarcus Aldridge", age: 33}) |
+----------------------------------+-----------------------------------------------------------+
| ("player204" :team{name: "Spurs"}) | ("player103" :player{age: 32, name: "Rudy Gay"})          |
+----------------------------------+-----------------------------------------------------------+
...
```

匹配定长路径

您可以在模式中使用 `:<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{name: "Tim Duncan", age: 42})        |
+-----------------------------------------------------------+
| ("player102" :player{name: "LaMarcus Aldridge", age: 33}) |
+-----------------------------------------------------------+
| ("player125" :player{name: "Manu Ginobili", age: 41})     |
+-----------------------------------------------------------+
```

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"}) |
+--------------------------------------------------+
```

匹配变长路径

您可以在模式中使用 `:<edge_type>*[minHop]..<maxHop>` 匹配变长路径。

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	必选项。表示路径的最大长度。 maxHop 必须是一个非负整数，没有默认值。

> **openCypher兼容性**：在openCypher中， maxHop 是可选项，默认为无穷大。当没有设置时， .. 可以省略。在nGQL中， maxHop 是必选项，而且 .. 不可以省略。

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[e:follow*1..3]->(v2) \
        RETURN v2 AS Friends;
+-------------------------------------------------------+
| Friends                                               |
+-------------------------------------------------------+
| ("player100" :player{age: 42, name: "Tim Duncan"})    |
+-------------------------------------------------------+
| ("player101" :player{age: 36, name: "Tony Parker"})   |
+-------------------------------------------------------+
| ("player125" :player{age: 41, name: "Manu Ginobili"}) |
+-------------------------------------------------------+
| ("player102" :player{age: 33, name: "LaMarcus Aldridge"}) |
+-------------------------------------------------------+
```

您可以使用 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) |
+-------------------------------------------------------+-----------+
| ("player125" :player{age: 41, name: "Manu Ginobili"}) | 3         |
+-------------------------------------------------------+-----------+
| ("player102" :player{age: 33, name: "LaMarcus Aldridge"}) | 1         |
+-------------------------------------------------------+-----------+
| ("player100" :player{age: 42, name: "Tim Duncan"})    | 4         |
+-------------------------------------------------------+-----------+
| ("player101" :player{age: 36, name: "Tony Parker"})   | 3         |
+-------------------------------------------------------+-----------+
```

如果 minHop 为 0 ，模式会匹配路径上的起始点。与上个示例相比，下面的示例设置 minHop 为 0 ，因为它是起始点，所以结果集中 "Tim Duncan" 比上个示例多计算一次。

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[e:follow*0..3]->(v2:player) \
        RETURN DISTINCT v2 AS Friends, count(v2);
+-------------------------------------------------------+-----------+
| Friends                                               | COUNT(v2) |
+-------------------------------------------------------+-----------+
| ("player125" :player{age: 41, name: "Manu Ginobili"}) | 3         |
+-------------------------------------------------------+-----------+
| ("player101" :player{age: 36, name: "Tony Parker"})   | 3         |
+-------------------------------------------------------+-----------+
| ("player102" :player{age: 33, name: "LaMarcus Aldridge"}) | 1         |
+-------------------------------------------------------+-----------+
| ("player100" :player{age: 42, name: "Tim Duncan"})    | 5         |
+-------------------------------------------------------+-----------+
```

匹配多个边类型的变长路径

您可以在变长或定长模式中指定多个边类型。 `hop` 、 `minHop` 和 `maxHop` 对所有边类型都生效。

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[e:follow|serve*2]->(v2) \
        RETURN DISTINCT v2;
+-------------------------------------------------------------+
| v2                                                          |
+-------------------------------------------------------------+
| ("player100" :player{name: "Tim Duncan", age: 42})          |
+-------------------------------------------------------------+
| ("player102" :player{name: "LaMarcus Aldridge", age: 33})   |
+-------------------------------------------------------------+
| ("player125" :player{name: "Manu Ginobili", age: 41})       |
+-------------------------------------------------------------+
| ("player204" :team{name: "Spurs"})                          |
+-------------------------------------------------------------+
| ("player215" :team{name: "Hornets"})                        |
+-------------------------------------------------------------+
```

常用检索操作

检索点或边的信息

使用 `RETURN {<vertex_name> | <edge_name>}` 检索点或边的所有信息。

```
nebula> MATCH (v:player{name:"Tim Duncan"}) \
        RETURN v;
+-------------------------------------------------+
| v                                               |
+-------------------------------------------------+
| ("player100" :player{name: "Tim Duncan", age: 42}) |
+-------------------------------------------------+

nebula> MATCH (v:player{name:"Tim Duncan"})-[e]->(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}]         |
+-------------------------------------------------------------------------------+
```

检索点ID

使用 `id()` 函数检索点ID。

```
nebula> MATCH (v:player{name:"Tim Duncan"}) \
        RETURN id(v);
+-------------+
| id(v)       |
+-------------+
| "player100" |
+-------------+
```

检索标签

使用 `labels()` 函数检索点上的标签列表。

```
nebula> MATCH (v:player{name:"Tim Duncan"}) \
        RETURN labels(v);
+------------+
| labels(v)  |
+------------+
| ["player"] |
+------------+
```

检索列表 `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"     |
+--------------+
```

检索点或边的单个属性

使用 `RETURN {<vertex_name> | <edge_name>}.<property>` 检索单个属性。

```
nebula> MATCH (v:player{name:"Tim Duncan"}) \
        RETURN v.age;
+-------+
| v.age |
+-------+
| 42    |
+-------+
```

使用 `AS` 设置属性的别名。

```
nebula> MATCH (v:player{name:"Tim Duncan"}) \
        RETURN v.age AS Age;
+-----+
| Age |
+-----+
| 42  |
+-----+
```

检索点或边的所有属性

使用 `properties()` 函数检索点或边的所有属性。

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[]->(v2) \
        RETURN properties(v2);
+-----------------------------------+
| properties(v2)                    |
+-----------------------------------+
| {"name":"Spurs"}                  |
+-----------------------------------+
| {"name":"Tony Parker", "age":36}  |
+-----------------------------------+
```

```
| {"age":41, "name":"Manu Ginobili"} |
+------------------------------------+
```

检索边类型

使用 `type()` 函数检索匹配的边类型。

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[e]->() \
        RETURN DISTINCT type(e);
+----------+
| type(e)  |
+----------+
| "follow" |
+----------+
| "serve"  |
+----------+
```

检索路径

使用 `RETURN <path_name>` 检索匹配路径的所有信息。

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[*3]->() \
        RETURN p;
+-------------------------------------------------------------------------------------------------------+
| p                                                                                                     |
+-------------------------------------------------------------------------------------------------------+
| <("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"})-[:serve@0 {end_year: 2019,
start_year: 2015}]->("team204" :team{name: "Spurs"})>           |
+-------------------------------------------------------------------------------------------------------+
| <("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"})-[:serve@0 {end_year: 2015,
start_year: 2006}]->("team203" :team{name: "Trail Blazers"})>   |
+-------------------------------------------------------------------------------------------------------+
| <("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"})-[:follow@0 {degree: 75}]-
>("player101" :player{age: 36, name: "Tony Parker"})>             |
+-------------------------------------------------------------------------------------------------------+
...
```

检索路径中的点

使用 `nodes()` 函数检索路径中的所有点。

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[]->(v2) \
        RETURN nodes(p);
+-----------------------------------------------------------------------------------------------------------------------+
| nodes(p)                                                                                                              |
+-----------------------------------------------------------------------------------------------------------------------+
| [("player100" :star{} :player{age: 42, name: "Tim Duncan"}), ("player204" :team{name: "Spurs"})]                      |
+-----------------------------------------------------------------------------------------------------------------------+
| [("player100" :star{} :player{age: 42, name: "Tim Duncan"}), ("player101" :player{name: "Tony Parker", age: 36})]   |
+-----------------------------------------------------------------------------------------------------------------------+
| [("player100" :star{} :player{age: 42, name: "Tim Duncan"}), ("player125" :player{name: "Manu Ginobili", age: 41})] |
+-----------------------------------------------------------------------------------------------------------------------+
```

检索路径中的边

使用 `relationships()` 函数检索路径中的所有边。

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[]->(v2) \
        RETURN relationships(p);
+------------------------------------------------------------------------------+
| relationships(p)                                                             |
+------------------------------------------------------------------------------+
| [[:follow "player100"->"player101" @0 {degree: 95}]]                         |
+------------------------------------------------------------------------------+
| [[:follow "player100"->"player125" @0 {degree: 95}]]                         |
+------------------------------------------------------------------------------+
| [[:serve "player100"->"team204" @0 {end_year: 2016, start_year: 1997}]]      |
+------------------------------------------------------------------------------+
```

检索路径长度

使用 `length()` 函数检索路径的长度。

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[*..2]->(v2) \
        RETURN p AS Paths, length(p) AS Length;
+-------------------------------------------------------------------------+--------+
| Paths                                                                   | Length |
+-------------------------------------------------------------------------+--------+
| <("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      |
+-------------------------------------------------------------------------+--------+
|  <("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"})-[: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"})-[:follow@0 {degree: 95}]->("player125" :player{age: 41, name: "Manu Ginobili"})>     | 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}]->("player100" :player{age: 42, name: "Tim Duncan"})>        | 2      |
+-------------------------------------------------------------------------+--------+
| <("player100" :player{age: 42, name: "Tim Duncan"})-[:serve@0 {end_year: 2016, start_year: 1997}]->("team204" :team{name:
"Spurs"})>                                                                | 1      |
+-------------------------------------------------------------------------+--------+
| <("player100" :player{age: 42, name: "Tim Duncan"})-[:follow@0 {degree: 95}]->("player125" :player{age: 41, name: "Manu
Ginobili"})>                                                              | 1      |
+-------------------------------------------------------------------------+--------+
```

3.6.2 LOOKUP

`LOOKUP` 根据索引遍历数据。您可以使用 `LOOKUP` 实现如下功能：

- 根据 `WHERE` 子句搜索特定数据。
- 通过标签列出点: 检索指定标签的所有点ID。
- 通过边类型列出边：检索指定边类型的所有边的起始点、目的点和rank。
- 计数指定标签的点或指定边类型的边。

openCypher兼容性

本文操作仅适用于nGQL扩展。

前提条件

请确保 `LOOKUP` 语句有至少一个索引可用。如果您需要创建索引，但是已经有相关的点、边或属性，您必须在创建索引后重建索引，索引才能生效。

> **警告**：正确使用索引可以加快查询速度，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，除非您很清楚使用索引对业务的影响。

语法

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

<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AND、OR一起使用。详情请参见WHERE [#3.ngql-guide/8.clauses-and-options/where/:]。
- `YIELD <return_list>` ：指定要返回的结果和格式.
- 如果只有 `WHERE` 子句，没有 `YIELD` 子句:
  - `LOOKUP` 标签时，返回点ID。
  - `LOOKUP` 边类型时，返回起始点ID、目的点ID和rank。

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。
- 字符串类型索引不支持范围扫描。
- 不支持 OR 和 XOR 运算符。

检索点

返回标签为 player 且 name 为 Tony Parker 的点。

```
nebula> CREATE TAG INDEX 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";
============
| VertexID |
============
| 101      |
------------

nebula> LOOKUP ON player WHERE player.name == "Tony Parker" \
        YIELD player.name, player.age;
====================================
| VertexID | player.name | player.age |
====================================
| 101      | Tony Parker | 36         |
------------------------------------

nebula> LOOKUP ON player WHERE player.name == "Kobe Bryant" YIELD player.name AS name \
        | GO FROM $-.VertexID OVER serve \
        YIELD $-.name, serve.start_year, serve.end_year, $$.team.name;
=================================================================
| $-.name     | serve.start_year | serve.end_year | $$.team.name |
=================================================================
| Kobe Bryant | 1996             | 2016           | Lakers       |
-----------------------------------------------------------------
```

检索边

返回边类型为 follow 且 degree 为 90 的边。

```
nebula> CREATE EDGE INDEX index_follow ON follow(degree);

nebula> REBUILD EDGE INDEX index_follow;
+------------+
```

```
| New Job Id |
+------------+
| 62         |
+------------+

nebula> LOOKUP ON follow WHERE follow.degree == 90;
============================
| SrcVID | DstVID | Ranking |
============================
| 100    | 106    | 0       |
----------------------------

nebula> LOOKUP ON follow WHERE follow.degree == 90 YIELD follow.degree;
==========================================
| SrcVID | DstVID | Ranking | follow.degree |
==========================================
| 100    | 106    | 0       | 90            |
------------------------------------------

nebula> LOOKUP ON follow WHERE follow.degree == 60 YIELD follow.degree AS Degree \
        | GO FROM $-.DstVID OVER serve \
        YIELD $-.DstVID, serve.start_year, serve.end_year, $$.team.name;
============================================================
| $-.DstVID | serve.start_year | serve.end_year | $$.team.name |
============================================================
| 105       | 2010             | 2018           | Spurs        |
------------------------------------------------------------
| 105       | 2009             | 2010           | Cavaliers    |
------------------------------------------------------------
| 105       | 2018             | 2019           | Raptors      |
------------------------------------------------------------
```

通过标签列出点/通过边类型列出边

如果需要通过标签列出点，或通过边类型列出边，则标签、边类型或属性上必须有至少一个索引。

例如一个标签 `player` 有两个属性 `name` 和 `age`，为了遍历所有包含标签 `player` 的点ID，标签 `player` 、属性 `name` 或属性 `age` 中必须有一个已经创建索引。

- 检索所有标记为 `player` 的点ID。

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

nebula> CREATE TAG INDEX 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);

nebula> LOOKUP ON player;
+-------------+
| _vid        |
+-------------+
| "player100" |
+-------------+
| "player101" |
+-------------+
```

- 检索边类型为 `like` 的所有边的信息。

```
nebula)> CREATE EDGE like(likeness int);

nebula)> CREATE EDGE INDEX like_index on like();

nebula> REBUILD EDGE INDEX like_index;
+------------+
| New Job Id |
+------------+
| 88         |
+------------+

nebula)> INSERT EDGE like(likeness) values "player100"->"player101":(95);

nebula)> LOOKUP ON like;
+-------------+----------+-------------+
| _src        | _ranking | _dst        |
+-------------+----------+-------------+
| "player100" | 0        | "player101" |
+-------------+----------+-------------+
```

计数点或边

计数标签为 `player` 的点和边类型为 `like` 的边。

```
nebula> LOOKUP ON player | YIELD COUNT(*) AS Player_Number;
+---------------+
| Player_Number |
+---------------+
| 2             |
+---------------+
```

```
nebula> LOOKUP ON like | YIELD COUNT(*) AS Like_Number;
+-------------+
| Like_Number |
+-------------+
| 1           |
+-------------+
```

FAQ

错误码411

```
[ERROR (-8)]: Unknown error(411):
```

错误码 `411` 表示 `WHERE` 过滤时没有有效的索引。Nebula Graph的索引使用的是最左匹配原则，即从最左边的为起点任何连续的索引都能匹配上。例如：

```
# 为标签t的前三个属性创建索引。
nebula> CREATE TAG INDEX example_index ON TAG t(p1, p2, p3);

# 无法匹配索引，因为不是从p1开始。
nebula> LOOKUP ON t WHERE p2 == 1 and p3 == 1;

# 可以匹配索引。
nebula> LOOKUP ON t WHERE p1 == 1;

# 可以匹配索引，因为p1和p2是连续的。
nebula> LOOKUP ON t WHERE p1 == 1 and p2 == 1;

# 可以匹配索引，因为p1、p2、p3是连续的。
nebula> LOOKUP ON t WHERE p1 == 1 and p2 == 1 and p3 == 1;
```

没有找到有效索引

```
No valid index found
```

如果您的查询过滤器包含字符串类型的字段，Nebula Graph会选择匹配所有字段的索引。例如：

```
nebula> CREATE TAG t1 (c1 string, c2 int);

nebula> CREATE TAG INDEX i1 ON t1 (c1, c2);

# 索引i1无效。
nebula> LOOKUP ON t1 WHERE t1.c1 == "a";

# 索引i1有效。
nebula> LOOKUP ON t1 WHERE t1.c1 == "a" and t1.c2 == 1;
```

3.6.3 GO

`GO` 用指定的过滤条件遍历图，并返回结果。

openCypher兼容性

本文操作仅适用于nGQL扩展。

语法

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

GO [[<M> TO] <N> STEPS ] FROM <vertex_list>
OVER <edge_type_list> [{REVERSELY | BIDIRECT}]
[ WHERE <conditions> ]
[| GROUP BY {col_name | expr | position} YIELD <col_name>]

<vertex_list> ::=
    <vid> [, <vid> ...]

<edge_type_list> ::=
   edge_type [, edge_type ...]
   | *
```

```
<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` 。详情请参见管道符 [#3.ngql-guide/5.operators/4.pipe/:]。
- `<edge_type_list>` ：遍历的边类型列表。
- `REVERSELY | BIDIRECT` ：默认情况下检索的是 `<vertex_list>` 的出边， `REVERSELY` 表示反向，即检索入边， `BIDIRECT` 表示双向，即检索出边和入边。
- `WHERE <conditions>` ：指定遍历的过滤条件。您可以在起始点、目的点和边使用 `WHERE` 子句，还可以结合 `AND` 、 `OR` 、 `NOT` 、 `XOR` 一起使用。详情请参见WHERE [#3.ngql-guide/8.clauses-and-options/where/:]。

  > **说明**：遍历多个边类型时， `WHERE` 子句有一些限制。例如不支持 `WHERE edge1.prop1 > edge2.prop2` 。

- `YIELD [DISTINCT] <return_list>` ：指定输出结果。详情请参见YIELD [#3.ngql-guide/8.clauses-and-options/yield/:]。如果没有指定，默认返回目的点ID。
- `ORDER BY` ：指定输出结果的排序规则。详情请参见ORDER BY [#3.ngql-guide/8.clauses-and-options/order-by/:]。

  > **说明**：没有指定排序规则时，输出结果的顺序不是固定的。

- `LIMIT` ：限制输出结果的行数。详情请参见LIMIT [#3.ngql-guide/8.clauses-and-options/limit/:]。
- `GROUP BY` ：根据指定属性的值将输出分组。详情请参见GROUP BY [#3.ngql-guide/8.clauses-and-options/group-by/:]。

示例

```
# 返回player102所属队伍。
nebula> GO FROM "player102" OVER serve;
+------------+
| serve._dst |
+------------+
| "team203"  |
+------------+
| "team204"  |
+------------+
```

```
# 返回距离player102两跳的朋友。
nebula> GO 2 STEPS FROM "player102" OVER follow;
+-------------+
| follow._dst |
+-------------+
| "player101" |
+-------------+
| "player125" |
+-------------+
...
```

```
# 添加过滤条件。
nebula> GO FROM "player100", "player102" OVER serve \
        WHERE serve.start_year > 1995 \
        YIELD DISTINCT $$.team.name AS team_name, serve.start_year AS start_year, $^.player.name AS player_name;
+-----------------+------------+--------------------+
| team_name       | start_year | player_name        |
+-----------------+------------+--------------------+
| "Spurs"         | 1997       | "Tim Duncan"       |
+-----------------+------------+--------------------+
| "Trail Blazers" | 2006       | "LaMarcus Aldridge" |
+-----------------+------------+--------------------+
| "Spurs"         | 2015       | "LaMarcus Aldridge" |
+-----------------+------------+--------------------+
```

```
# 遍历多个边类型。属性没有值时，会显示__EMPTY__。
nebula> GO FROM "player100" OVER follow, serve \
        YIELD follow.degree, serve.start_year;
+---------------+-----------------+
| follow.degree | serve.start_year |
+---------------+-----------------+
| 95            | __EMPTY__       |
+---------------+-----------------+
| 95            | __EMPTY__       |
+---------------+-----------------+
| __EMPTY__     | 1997            |
+---------------+-----------------+
```

```
# 返回player100的入边。
nebula> GO FROM "player100" OVER follow REVERSELY \
        YIELD follow._dst 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 follow._dst AS id | \
        GO FROM $-.id OVER serve \
        WHERE $^.player.age > 20 \
        YIELD $^.player.name AS FriendOf, $$.team.name AS Team;
+--------------------+-----------------+
| FriendOf           | Team            |
+--------------------+-----------------+
| "Tony Parker"      | "Spurs"         |
```

```
+--------------------+-----------------+
| "Tony Parker"      | "Hornets"       |
+--------------------+-----------------+
...

# 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.name AS FriendOf, v3.name AS Team;
+--------------------+-----------------+
| FriendOf           | Team            |
+--------------------+-----------------+
| "Tony Parker"      | "Spurs"         |
+--------------------+-----------------+
| "Tony Parker"      | "Hornets"       |
+--------------------+-----------------+
...
```

```
# 返回player102的出边和入边。
nebula> GO FROM "player102" OVER follow BIDIRECT \
        YIELD follow._dst AS both;
+-------------+
| both        |
+-------------+
| "player100" |
+-------------+
| "player101" |
+-------------+
...

# 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)== "player102" \
        RETURN id(v2) AS both;
+-------------+
| both        |
+-------------+
| "player101" |
+-------------+
| "player103" |
+-------------+
...
```

```
# 查询player100 1~2跳内的朋友。
nebula> GO 1 TO 2 STEPS FROM "player100" OVER follow \
        YIELD follow._dst 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 follow._src AS src, follow._dst AS dst, $$.player.age AS age \
        | GROUP BY $-.dst \
        YIELD $-.dst AS dst, collect_set($-.src) AS src, collect($-.age) AS age
+-------------+---------------------------+----------+
| 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 follow._src AS src, follow._dst AS dst; \
        GO 2 STEPS FROM $a.dst OVER follow \
        YIELD $a.src AS src, $a.dst, follow._src, follow._dst \
        | ORDER BY $-.src | OFFSET 1 LIMIT 2;
+-------------+-------------+-------------+-------------+
| src         | $a.dst      | follow._src | follow._dst |
+-------------+-------------+-------------+-------------+
| "player100" | "player125" | "player100" | "player101" |
+-------------+-------------+-------------+-------------+
| "player100" | "player101" | "player100" | "player125" |
+-------------+-------------+-------------+-------------+
```

3.6.4 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nGQL扩展。

获取点的属性值

语法

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

参数	说明
`tag_name`	标签名称。
`*`	表示当前图空间中的所有标签。
`vid`	点ID。
`output`	指定要返回的信息。详情请参见`YIELD` [#3.ngql-guide/8.clauses-and-options/yield/:]。如果没有 `YIELD` 子句，将返回所有匹配的信息。

基于标签获取点的属性值

在 `FETCH` 语句中指定标签获取对应点的属性值。

```
nebula> FETCH PROP ON player "player100";
+-------------------------------------------------+
| vertices_                                       |
+-------------------------------------------------+
| ("player100" :player{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 player.name;
+-------------+--------------+
| VertexID    | player.name  |
+-------------+--------------+
| "player100" | "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";
+-------------------------------------------------------+
| vertices_                                             |
+-------------------------------------------------------+
| ("player101" :player{age: 36, name: "Tony Parker"})   |
+-------------------------------------------------------+
| ("player102" :player{age: 33, name: "LaMarcus Aldridge"}) |
+-------------------------------------------------------+
| ("player103" :player{age: 32, name: "Rudy Gay"})      |
+-------------------------------------------------------+
```

基于多个标签获取点的属性值

在 `FETCH` 语句中指定多个标签获取属性值。标签之间用英文逗号（,）分隔。

```
# 创建新标签t1。
nebula> CREATE TAG t1(a string, b int);

# 为点player100添加标签t1。
nebula> INSERT VERTEX t1(a, b) VALUE "player100":("Hello", 100);

# 基于标签player和t1获取点player100上的属性值。
nebula> FETCH PROP ON player, t1 "player100";
+-------------------------------------------------------------------------------+
| vertices_                                                                     |
+-------------------------------------------------------------------------------+
| ("player100" :t1{a: "Hello", b: 100} :player{age: 42, name: "Tim Duncan"}) |
+-------------------------------------------------------------------------------+
```

您可以在 `FETCH` 语句中组合多个标签和多个点。

```ngql
nebula> FETCH PROP ON player, t1 "player100", "player103";
+-------------------------------------------------------------------------------+
| vertices_                                                                     |
+-------------------------------------------------------------------------------+
| ("player100" :t1{a: "Hello", b: 100} :player{age: 42, name: "Tim Duncan"}) |
+-------------------------------------------------------------------------------+
| ("player103" :player{age: 32, name: "Rudy Gay"})                              |
+-------------------------------------------------------------------------------+
```

获取点上所有属性值

在`FETCH`语句中使用`*`获取点上所有属性值。

```ngql
nebula> FETCH PROP ON * "player100", "player106", "team200";
+-------------------------------------------------------------------------------+
| vertices_                                                                     |
+-------------------------------------------------------------------------------+
| ("player106" :player{age: 25, name: "Kyle Anderson"})                         |
+-------------------------------------------------------------------------------+
| ("team200" :team{name: "Warriors"})                                           |
+-------------------------------------------------------------------------------+
| ("player100" :t1{a: "Hello", b: 100} :player{age: 42, name: "Tim Duncan"}) |
+-------------------------------------------------------------------------------+
```

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`	边类型名称。
`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、边类型和rank可以唯一确定一条边。
`output`	指定要返回的信息。详情请参见`YIELD` [#3.ngql-guide/8.clauses-and-options/yield/:]。如果没有 `YIELD` 子句，将返回所有匹配的信息。

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";
+------------------------------------------------------------------------+
| edges_                                                                 |
+------------------------------------------------------------------------+
| [:serve "player100"->"team204" @0 {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 serve.start_year;
+-------------+------------+-------------+------------------+
| serve._src  | serve._dst | serve._rank | serve.start_year |
+-------------+------------+-------------+------------------+
| "player100" | "team204"  | 0           | 1997             |
+-------------+------------+-------------+------------------+
```

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";
+------------------------------------------------------------------------+
| edges_                                                                 |
+------------------------------------------------------------------------+
| [: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、目的点和边类型都相同，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";
+------------------------------------------------------------------------+
| edges_                                                                 |
+------------------------------------------------------------------------+
| [: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;
+------------------------------------------------------------------------+
| edges_                                                                 |
+------------------------------------------------------------------------+
| [:serve "player100"->"team204" @1 {end_year: 2017, start_year: 1998}] |
+------------------------------------------------------------------------+
```

复合语句中使用FETCH

将 `FETCH` 与nGQL扩展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 follow._src AS s, follow._dst AS d \
        | FETCH PROP ON follow $-.s -> $-.d \
        YIELD follow.degree;
+-------------+-------------+--------------+---------------+
| follow._src | follow._dst | follow._rank | follow.degree |
+-------------+-------------+--------------+---------------+
| "player101" | "player100" | 0            | 95            |
+-------------+-------------+--------------+---------------+
| "player101" | "player102" | 0            | 90            |
+-------------+-------------+--------------+---------------+
| "player101" | "player125" | 0            | 95            |
+-------------+-------------+--------------+---------------+
```

您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 follow._src AS s, follow._dst AS d; \
        FETCH PROP ON follow $var.s -> $var.d \
        YIELD follow.degree;
+-------------+-------------+--------------+---------------+
| follow._src | follow._dst | follow._rank | follow.degree |
+-------------+-------------+--------------+---------------+
| "player101" | "player100" | 0            | 95            |
+-------------+-------------+--------------+---------------+
| "player101" | "player102" | 0            | 90            |
+-------------+-------------+--------------+---------------+
| "player101" | "player125" | 0            | 95            |
+-------------+-------------+--------------+---------------+
```

更多复合语句的详情，请参见复合查询（子句结构） [#3.ngql-guide/4.variable-and-composite-queries/1.composite-queries/:]。

3.6.5 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) [nba]> 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"})] |
+-------------------------------------------------------------------------------------------------------------------------------------------------------+
```

3.6.6 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。

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` 。
- `utf8mb4_bin` 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。

SHOW CREATE SPACE

`SHOW CREATE SPACE` 语句显示指定图空间的基本信息，例如创建图空间的nGQL、分片数量、副本数量等。

图空间的更多详细信息，请参见CREATE SPACE [#3.ngql-guide/9.space-statements/1.create-space/:]。

语法

```
SHOW CREATE SPACE <space_name>;
```

示例

```
nebula> SHOW CREATE SPACE nba;
+-------
+-----------------------------------------------------------------------------------------------------------------------------------------------
+
| Space | Create
Space                                                                                                                                          |
+-------
+-----------------------------------------------------------------------------------------------------------------------------------------------
+
| "nba" | "CREATE SPACE `nba` (partition_num = 10, replica_factor = 1, charset = utf8, collate = utf8_bin, vid_type =
FIXED_STRING(32))" |
+-------
+-----------------------------------------------------------------------------------------------------------------------------------------------
+
```

SHOW CREATE TAG/EDGE

`SHOW CREATE TAG` 语句显示指定标签的基本信息。标签的更多详细信息，请参见CREATE TAG [#3.ngql-guide/10.tag-statements/1.create-tag/:]。

`SHOW CREATE EDGE` 语句显示指定边类型的基本信息。边类型的更多详细信息，请参见CREATE EDGE [#3.ngql-guide/11.edge-type-statements/1.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 = """ |
+----------+----------------------------------+
```

SHOW HOSTS

`SHOW HOSTS` 语句显示由Meta服务注册的Graph、Storage、Meta主机。

语法

```
SHOW HOSTS [GRAPH/STORAGE/META];
```

示例

```
nebula> SHOW HOSTS;
+-------------+-------+----------+--------------+---------------------+------------------------+
| Host        | Port  | Status   | Leader count | Leader distribution | Partition distribution |
+-------------+-------+----------+--------------+---------------------+------------------------+
| "storaged0" | 9779  | "ONLINE" | 8            | "docs:5, nba:3"     | "docs:5, nba:3"        |
+-------------+-------+----------+--------------+---------------------+------------------------+
| "storaged1" | 9779  | "ONLINE" | 9            | "nba:4, docs:5"     | "docs:5, nba:4"        |
+-------------+-------+----------+--------------+---------------------+------------------------+
| "storaged2" | 9779  | "ONLINE" | 8            | "nba:3, docs:5"     | "docs:5, nba:3"        |
+-------------+-------+----------+--------------+---------------------+------------------------+
Got 3 rows (time spent 866/1411 us)

nebula> SHOW HOSTS GRAPH;
+-------------+------+----------+---------+--------------+
| Host        | Port | Status   | Role    | Git Info Sha |
+-------------+------+----------+---------+--------------+
| "12.16.2.3" | 9669 | "ONLINE" | "GRAPH" | "761f22b"    |

nebula> SHOW HOSTS STORAGE;
+-------------+------+----------+-----------+--------------+
| Host        | Port | Status   | Role      | Git Info Sha |
+-------------+------+----------+-----------+--------------+
| "12.16.2.3" | 9779 | "ONLINE" | "STORAGE" | "761f22b"    |

nebula> SHOW HOSTS META;
+-------------+------+----------+--------+--------------+
| Host        | Port | Status   | Role   | Git Info Sha |
+-------------+------+----------+--------+--------------+
| "12.16.2.3" | 9559 | "ONLINE" | "META" | "761f22b"    |
```

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 |
+----------------+--------------+
| "like_index_0" | "FINISHED"   |
+----------------+--------------+
| "like1"        | "FINISHED"   |
+----------------+--------------+

nebula> SHOW EDGE INDEX STATUS;
+----------------+--------------+
| Name           | Index Status |
+----------------+--------------+
| "index_follow" | "FINISHED"   |
+----------------+--------------+
```

相关文档

- 管理作业 [#3.ngql-guide/18.operation-and-maintenance-statements/4.job-statements/:]
- REBUILD NATIVE INDEX [#3.ngql-guide/14.native-index-statements/4.rebuild-native-index/:]

SHOW INDEXES

`SHOW INDEXES` 语句显示现有的原生索引。

语法

```
SHOW {TAG | EDGE} INDEXES;
```

示例

```
nebula> SHOW TAG INDEXES;
+------------------+
| Names            |
+------------------+
| "play_age_0"     |
+------------------+
| "player_index_0" |
+------------------+

nebula> SHOW EDGE INDEXES;
+----------------+
| Names          |
+----------------+
| "index_follow" |
+----------------+
```

SHOW PARTS

`SHOW PARTS` 语句显示图空间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            | "storaged1:44500" | "storaged1:44500" | ""    |
+--------------+------------------+------------------+-------+
| 2            | "storaged2:44500" | "storaged2:44500" | ""    |
+--------------+------------------+------------------+-------+
| 3            | "storaged0:44500" | "storaged0:44500" | ""    |
+--------------+------------------+------------------+-------+
| 4            | "storaged1:44500" | "storaged1:44500" | ""    |
+--------------+------------------+------------------+-------+
| 5            | "storaged2:44500" | "storaged2:44500" | ""    |
+--------------+------------------+------------------+-------+
| 6            | "storaged0:44500" | "storaged0:44500" | ""    |
+--------------+------------------+------------------+-------+
| 7            | "storaged1:44500" | "storaged1:44500" | ""    |
+--------------+------------------+------------------+-------+
| 8            | "storaged2:44500" | "storaged2:44500" | ""    |
+--------------+------------------+------------------+-------+
| 9            | "storaged0:44500" | "storaged0:44500" | ""    |
+--------------+------------------+------------------+-------+
| 10           | "storaged1:44500" | "storaged1:44500" | ""    |
+--------------+------------------+------------------+-------+

nebula> SHOW PARTS 1;
+--------------+------------------+------------------+-------+
| Partition ID | Leader           | Peers            | Losts |
+--------------+------------------+------------------+-------+
| 1            | "storaged1:44500" | "storaged1:44500" | ""    |
+--------------+------------------+------------------+-------+
```

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 [#7.data-security/1.authentication/3.role-list/:]。

语法

```
SHOW ROLES IN <space_name>;
```

示例

```
nebula> SHOW ROLES in nba;
+---------+-----------+
| Account | Role Type |
+---------+-----------+
| "user1" | "ADMIN"   |
+---------+-----------+
```

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 [#7.data-security/3.manage-snapshot/:]。

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

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 [#3.ngql-guide/9.space-statements/1.create-space/:]。

语法

```
SHOW SPACES;
```

示例

```
nebula> SHOW SPACES;
+--------+
| Name   |
+--------+
| "docs" |
+--------+
| "nba"  |
+--------+
```

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
- 每个标签关联的点的总数
- 每个边类型关联的边的总数

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 [#3.ngql-guide/18.operation-and-maintenance-statements/4.job-statements/:]。

> **说明**： `SHOW STATS` 的结果取决于最近一次执行的 `SUBMIT JOB STATS` ，如果您需要更新结果，请再次执行 `SUBMIT JOB STATS` 。

语法

```
SHOW STATS;
```

示例

```
# 选择图空间。
nebula> USE nba;

# 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  |
+----------------+---------------+------------+------------+------------+
| 98             | "STATS"       | "FINISHED" | 1606552675 | 1606552675 |
+----------------+---------------+------------+------------+------------+
| 0              | "storaged2"   | "FINISHED" | 1606552675 | 1606552675 |
+----------------+---------------+------------+------------+------------+
| 1              | "storaged0"   | "FINISHED" | 1606552675 | 1606552675 |
+----------------+---------------+------------+------------+------------+
| 2              | "storaged1"   | "FINISHED" | 1606552675 | 1606552675 |
+----------------+---------------+------------+------------+------------+

# 显示图空间统计信息。
nebula> SHOW STATS;
+---------+------------+-------+
| Type    | Name       | Count |
+---------+------------+-------+
| "Tag"   | "player"   | 51    |
+---------+------------+-------+
| "Tag"   | "team"     | 30    |
```

```
+---------+------------+-------+
| "Edge"  | "like"     | 81    |
+---------+------------+-------+
| "Edge"  | "serve"    | 152   |
+---------+------------+-------+
| "Space" | "vertices" | 81    |
+---------+------------+-------+
| "Space" | "edges"    | 233   |
+---------+------------+-------+
```

SHOW TAGS/EDGES

`SHOW TAGS` 语句显示当前图空间内的所有标签。

`SHOW EDGES` 语句显示当前图空间内的所有边类型。

语法

```
SHOW {TAGS | EDGES};
```

示例

```
nebula> SHOW TAGS;
+----------+
| Name     |
+----------+
| "player" |
+----------+
| "star"   |
+----------+
| "team"   |
+----------+

nebula> SHOW EDGES;
+---------+
| Name    |
+---------+
| "like"  |
+---------+
| "serve" |
+---------+
```

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 |
+---------+
| "root"  |
+---------+
| "user1" |
+---------+
```

3.7 子句和选项

3.7.1 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nGQL扩展。

您也可以使用openCypher方式的count() [#3.ngql-guide/6.functions-and-expressions/7.count/:]函数聚合数据。

```
nebula>  MATCH (v:player)<-[:follow]-(:player) RETURN v.name AS Name, count(*) as cnt ORDER BY cnt DESC
+----------------------+--------------+
| Name                 | Follower_Num |
+----------------------+--------------+
| "Tim Duncan"         | 10           |
+----------------------+--------------+
| "LeBron James"       | 6            |
+----------------------+--------------+
| "Tony Parker"        | 5            |
+----------------------+--------------+
| "Manu Ginobili"      | 4            |
+----------------------+--------------+
| "Chris Paul"         | 4            |
+----------------------+--------------+
| "Tracy McGrady"      | 3            |
+----------------------+--------------+
| "Dwyane Wade"        | 3            |
+----------------------+--------------+
...
```

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 $$.player.name as Name \
        | GROUP BY $-.Name \
        YIELD $-.Name as Player, count(*) AS Name_Count;
+--------------------+------------+
| Player             | Name_Count |
+--------------------+------------+
| "Tiago Splitter"   | 1          |
```

```
+--------------------+------------+
| "Aron Baynes"      | 1          |
+--------------------+------------+
| "Boris Diaw"       | 1          |
+--------------------+------------+
| "Manu Ginobili"    | 2          |
+--------------------+------------+
| "Dejounte Murray"  | 1          |
+--------------------+------------+
| "Danny Green"      | 1          |
+--------------------+------------+
| "Tony Parker"      | 2          |
+--------------------+------------+
| "Shaquille O'Neal"  | 1         |
+--------------------+------------+
| "LaMarcus Aldridge" | 1         |
+--------------------+------------+
| "Marco Belinelli"  | 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 follow._src AS player, follow.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 [#3.ngql-guide/6.functions-and-expressions/1.math/:]。

3.7.2 LIMIT

`LIMIT` 子句限制输出结果的行数。

- 在nGQL扩展中，必须使用管道符（|），可以忽略偏移量。
- 在openCypher方式中，不允许使用管道符，可以使用 `SKIP` 指明偏移量。

> **说明**：在nGQL扩展或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。

nGQL扩展语法

在nGQL扩展中， `LIMIT` 的工作原理与 `SQL` 相同，必须和管道符一起使用。 `LIMIT` 子句接收一个或两个参数。参数的值必须是非负整数。

```
YIELD <var>
[| LIMIT [<offset_value>,] <number_rows>];
```

参数	说明
`var`	排序的列或计算结果。
`offset_value`	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

```
# 从排序结果中返回第2行开始的3行数据。
nebula> GO FROM "player100" OVER follow REVERSELY \
        YIELD $$.player.name AS Friend, $$.player.age AS Age \
        | ORDER BY Age,Friend \
        | LIMIT 1, 3;
+------------------+-----+
| Friend           | Age |
+------------------+-----+
| "Danny Green"    | 31  |
+------------------+-----+
| "Aron Baynes"    | 32  |
+------------------+-----+
| "Marco Belinelli" | 32  |
+------------------+-----+
```

openCypher方式语法

```
RETURN <var>
[SKIP <offset>]
[LIMIT <number_rows>];
```

参数	说明
`var`	排序的列或计算结果。
`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。

> **说明**：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。

示例

```
nebula> MATCH (v:player) RETURN v.name AS Name, v.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.name AS Name, v.age AS Age \
        ORDER BY Age LIMIT rand32(5);
+-------------------------+-----+
| Name                    | Age |
+-------------------------+-----+
| "Luka Doncic"           | 20  |
+-------------------------+-----+
| "Ben Simmons"           | 22  |
+-------------------------+-----+
| "Kristaps Porzingis"    | 23  |
+-------------------------+-----+
| "Giannis Antetokounmpo" | 24  |
+-------------------------+-----+
```

SKIP示例

您可以单独使用 `SKIP <offset>` 设置偏移量，后面不需要添加 `LIMIT <number_rows>`。

```
nebula> MATCH (v:player{name:"Tim Duncan"}) --> (v2) \
        RETURN v2.name AS Name, v2.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.name AS Name, v2.age AS Age \
```

```
        ORDER BY Age DESC SKIP 1+1;
+---------------+-----+
| Name          | Age |
+---------------+-----+
| "Tony Parker" | 36  |
+---------------+-----+
```

您也可以同时使用 `SKIP <offset>` 和 `LIMIT <number_rows>`，返回中间的部分数据。

```
nebula> MATCH (v:player{name:"Tim Duncan"}) --> (v2) \
        RETURN v2.name AS Name, v2.age AS Age \
        ORDER BY Age DESC SKIP 1 LIMIT 1;
+-----------------+-----+
| Name            | Age |
+-----------------+-----+
| "Manu Ginobili" | 41  |
+-----------------+-----+
```

3.7.3 ORDER BY

`ORDER BY` 子句指定输出结果的排序规则。

- 在nGQL扩展中，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。

nGQL扩展语法

```
<YIELD clause>
ORDER BY <expression> [ASC | DESC] [, <expression> [ASC | DESC] ...];
```

示例

```
nebula> FETCH PROP ON player "player100", "player101", "player102", "player103" \
        YIELD player.age AS age, player.name AS name \
        | ORDER BY age ASC, name DESC;
+-------------+-----+--------------------+
| VertexID    | age | name               |
+-------------+-----+--------------------+
| "player103" | 32  | "Rudy Gay"         |
+-------------+-----+--------------------+
| "player102" | 33  | "LaMarcus Aldridge" |
+-------------+-----+--------------------+
| "player101" | 36  | "Tony Parker"      |
+-------------+-----+--------------------+
| "player100" | 42  | "Tim Duncan"       |
+-------------+-----+--------------------+
```

OpenCypher方式语法

```
<RETURN clause>
ORDER BY <expression> [ASC | DESC] [, <expression> [ASC | DESC] ...];
```

示例

```
nebula> MATCH (v:player) RETURN v.name AS Name, v.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.age AS Age, v.name AS Name  \
        ORDER BY Age DESC, Name ASC;
+-----+-------------------+
| Age | Name              |
+-----+-------------------+
| 47  | "Shaquille O'Neal" |
+-----+-------------------+
| 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.name AS Name, v2.age AS Age  \
        ORDER BY Age;
+-----------------+----------+
| Name            | Age      |
+-----------------+----------+
| "Tony Parker"   | 36       |
+-----------------+----------+
| "Manu Ginobili" | 41       |
+-----------------+----------+
| "Spurs"         | __NULL__ |
+-----------------+----------+

nebula> MATCH (v:player{name:"Tim Duncan"}) --> (v2) \
        RETURN v2.name AS Name, v2.age AS Age  \
        ORDER BY Age DESC;
+-----------------+----------+
| Name            | Age      |
+-----------------+----------+
| "Spurs"         | __NULL__ |
+-----------------+----------+
| "Manu Ginobili" | 41       |
+-----------------+----------+
| "Tony Parker"   | 36       |
+-----------------+----------+
```

3.7.4 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nGQL扩展如何定义输出结果，请参见`YIELD` [#3.ngql-guide/8.clauses-and-options/yield/:]。

`RETURN` 暂不支持如下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nGQL扩展，语法为 `RETURN <var_ref> IF <var_ref> IS NOT NULL` 。
- 在nGQL 2.0中， `RETURN` 不适用于nGQL扩展。

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"})      |
+------------------------------------------------------------+
...
```

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.name, v.age \
        LIMIT 3;
+------------------+-------+
| v.name           | v.age |
+------------------+-------+
| "Rajon Rondo"    | 33    |
+------------------+-------+
| "Rudy Gay"       | 32    |
+------------------+-------+
| "Dejounte Murray" | 29    |
+------------------+-------+
```

返回所有元素

使用星号（*）返回匹配模式中的所有元素。

```
nebula> MATCH (v:player{name:"Tim Duncan"}) \
        RETURN *;
+-------------------------------------------------+
| v                                               |
+-------------------------------------------------+
| ("player100" :player{age: 42, name: "Tim Duncan"}) |
+-------------------------------------------------+

nebula> MATCH (v:player{name:"Tim Duncan"})-[e]->(v2) \
        RETURN *;
+-------------------------------------------------+------------------------------------------------------------------------------
+-------------------------------------------------+
| v                                               | e                                                                                |
v2                                               |
+-------------------------------------------------+------------------------------------------------------------------------------
+-------------------------------------------------+
| ("player100" :player{age: 42, name: "Tim Duncan"}) | [:follow "player100"->"player101" @0 {degree: 95}]                               |
("player101" :player{age: 36, name: "Tony Parker"})   |
```

```
+-------------------------------------------------+-------------------------------------------------------------------
+-------------------------------------------------+
| ("player100" :player{age: 42, name: "Tim Duncan"}) | [:follow "player100"->"player125" @0 {degree: 95}]                  |
("player125" :player{age: 41, name: "Manu Ginobili"}) |
+-------------------------------------------------+-------------------------------------------------------------------
+---------------------------------------------------+
| ("player100" :player{age: 42, name: "Tim Duncan"}) | [:serve "player100"->"team204" @0 {end_year: 2016, start_year: 1997}] |
("team204" :team{name: "Spurs"})                     |
+-------------------------------------------------+-------------------------------------------------------------------
+---------------------------------------------------+
```

重命名字段

使用语法 `AS <alias>` 重命名输出结果中的字段。

```
nebula> MATCH (v:player{name:"Tim Duncan"})-[:serve]->(v2) \
        RETURN v2.name AS Team;
+---------+
| Team    |
+---------+
| "Spurs" |
+---------+

nebula> RETURN "Amber" AS Name;
+---------+
| Name    |
+---------+
| "Amber" |
+---------+
```

返回不存在的属性

如果匹配的结果中，某个属性不存在，会返回 `NULL` 。

```
nebula> MATCH (v:player{name:"Tim Duncan"})-[e]->(v2) \
        RETURN v2.name, type(e), v2.age;
+-----------------+----------+----------+
| v2.name         | type(e)  | v2.age   |
+-----------------+----------+----------+
| "Tony Parker"   | "follow" | 36       |
+-----------------+----------+----------+
| "Manu Ginobili" | "follow" | 41       |
+-----------------+----------+----------+
| "Spurs"         | "serve"  | __NULL__ |
+-----------------+----------+----------+
```

返回表达式结果

`RETURN` 语句可以返回文字、函数或谓词等表达式的结果。

```
nebula> MATCH (v:player{name:"Tony Parker"})-->(v2:player) \
        RETURN DISTINCT v2.name, "Hello"+" graphs!", v2.age > 35;
+--------------------+------------------+-------------+
| v2.name            | (Hello+ graphs!) | (v2.age>35) |
+--------------------+------------------+-------------+
| "Tim Duncan"       | "Hello graphs!"  | true        |
+--------------------+------------------+-------------+
```

```
| "LaMarcus Aldridge" | "Hello graphs!"  | false       |
+---------------------+------------------+-------------+
| "Manu Ginobili"     | "Hello graphs!"  | true        |
+---------------------+------------------+-------------+

nebula> RETURN 1+1;
+-------+
| (1+1) |
+-------+
| 2     |
+-------+

nebula> RETURN 3 > 1;
+-------+
| (3>1) |
+-------+
| true  |
+-------+

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.name, v2.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     |
+---------------------+--------+
Got 8 rows (time spent 3273/3893 us)

# 使用DISTINCT。
nebula> MATCH (v:player{name:"Tony Parker"})--(v2:player) \
        RETURN DISTINCT v2.name, v2.age;
+---------------------+--------+
| v2.name             | v2.age |
+---------------------+--------+
| "Tim Duncan"        | 42     |
+---------------------+--------+
```

```
| "LaMarcus Aldridge" | 33     |
+--------------------+--------+
| "Marco Belinelli"   | 32     |
+--------------------+--------+
| "Boris Diaw"        | 36     |
+--------------------+--------+
| "Dejounte Murray"   | 29     |
+--------------------+--------+
| "Manu Ginobili"     | 41     |
+--------------------+--------+
```

3.7.5 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nGQL扩展。

注意事项

- 不能修改带有TTL选项的属性。
- 标签或边类型上不能同时存在TTL选项和索引，即使在不同属性上分别设置也不行。

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标签，点上的一个属性过期，点也会过期。
- 如果一个点有多个标签，点上的一个属性过期，和该属性相同标签的其他属性也会过期，但是点不会过期，点上其他标签的属性保持不变。

边属性过期

因为一条边仅有一个边类型，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 [#8.service-tuning/compaction/:]过程中回收硬盘空间。

> **说明**：如果关闭TTL选项 [#3.ngql-guide/8.clauses-and-options/ttl-options/:ttl_1]，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

标签或边类型已存在

如果标签和边类型已经创建，请使用 `ALTER` 语句更新标签或边类型。

```
# 创建标签。
nebula> CREATE TAG t1 (a timestamp);

# ALTER修改标签，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());
```

标签或边类型不存在

创建标签或边类型时可以同时设置TTL选项。详情请参见CREATE TAG [#3.ngql-guide/10.tag-statements/1.create-tag/:]和CREATE EDGE [#3.ngql-guide/11.edge-type-statements/1.create-edge/:]。

```
# 创建标签并设置TTL选项。
nebula> CREATE TAG t2(a int, b int, c string) ttl_duration= 100, ttl_col = "a";

# 插入点。过期时间戳为1612778164774（1612778164674 + 100）。
nebula> INSERT VERTEX t2(a, b, c) values "102":(1612778164674, 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，同时防止属性过期。

  ```
  nebula> ALTER TAG t1 ttl_duration = 0;
  ```

  > **警告**：即使 `ttl_duration` 为 `0`，您也不能修改对应的属性。

3.7.6 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：

- nGQL扩展，例如 `GO` 和 `LOOKUP` 语句。
- openCypher方式，例如 `MATCH` 和 `WITH` 语句。

openCypher兼容性

- 不支持在模式中使用 `WHERE` 子句（TODO: planning），例如 `WHERE (v)-->(v2)` 。
- 过滤rank [#3.ngql-guide/8.clauses-and-options/where/:rank]是原生nGQL功能。只支持在nGQL扩展的语句（例如 `GO` 和 `LOOKUP` ）中使用，因为openCypher中没有rank的概念。

基础用法

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 [#3.ngql-guide/5.operators/9.precedence/:]。

```
nebula> MATCH (v:player) \
        WHERE v.name == "Tim Duncan" \
        XOR (v.age < 30 AND v.name == "Yao Ming") \
        OR NOT (v.name == "Yao Ming" OR v.name == "Tim Duncan") \
        RETURN v.name, v.age;
+-------------------------+-------+
| v.name                  | v.age |
+-------------------------+-------+
| "Marco Belinelli"       | 32    |
+-------------------------+-------+
| "Aron Baynes"           | 32    |
+-------------------------+-------+
| "LeBron James"          | 34    |
+-------------------------+-------+
| "James Harden"          | 29    |
+-------------------------+-------+
| "Manu Ginobili"         | 41    |
+-------------------------+-------+
...
```

```
nebula> GO FROM "player100" \
        OVER follow \
        WHERE follow.degree > 90 \
        OR $$.player.age != 33 \
        AND $$.player.name != "Tony Parker";
+-------------+
| follow._dst |
+-------------+
| "player101" |
+-------------+
| "player125" |
+-------------+
```

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.age < 25 \
        RETURN v2.name, v2.age;
+----------------------+--------+
| v2.name              | v2.age |
+----------------------+--------+
| "Luka Doncic"        | 20     |
+----------------------+--------+
| "Kristaps Porzingis" | 23     |
+----------------------+--------+
| "Ben Simmons"        | 22     |
+----------------------+--------+
```

```
nebula> GO FROM "player100" \
        OVER follow \
        WHERE $^.player.age >= 42;
+-------------+
| follow._dst |
+-------------+
| "player101" |
+-------------+
| "player125" |
+-------------+
```

- 过滤边属性：

```
nebula> MATCH (v:player)-[e]->() \
        WHERE e.start_year < 2000 \
        RETURN DISTINCT v.name, v.age;
+--------------------+-------+
| v.name             | v.age |
+--------------------+-------+
| "Shaquille O'Neal" | 47    |
+--------------------+-------+
| "Steve Nash"       | 45    |
+--------------------+-------+
| "Ray Allen"        | 43    |
+--------------------+-------+
| "Grant Hill"       | 46    |
+--------------------+-------+
| "Tony Parker"      | 36    |
+--------------------+-------+
...
```

```
nebula> GO FROM "player100" \
        OVER follow \
        WHERE follow.degree > 90;
+-------------+
| follow._dst |
+-------------+
| "player101" |
+-------------+
| "player125" |
+-------------+
```

过滤动态计算属性

```
nebula> MATCH (v:player) \
        WHERE v[toLower("AGE")] < 21 \
        RETURN v.name, v.age;
+---------------+-------+
| v.name        | v.age |
+---------------+-------+
| "Luka Doncic" | 20    |
+---------------+-------+
```

过滤现存属性

```
nebula> MATCH (v:player) \
        WHERE exists(v.age) \
        RETURN v.name, v.age;
+-------------------------+-------+
| v.name                  | v.age |
+-------------------------+-------+
| "Boris Diaw"            | 36    |
+-------------------------+-------+
| "DeAndre Jordan"        | 30    |
+-------------------------+-------+
```

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 test;
nebula> USE test;
nebula> CREATE EDGE e1(p1 int);
nebula> CREATE TAG 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 e1._rank>2 \
        YIELD e1._src, e1._dst, e1._rank AS Rank, e1.p1 | \
        ORDER BY Rank DESC;
==================================
| e1._src | e1._dst | Rank | e1.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.name STARTS WITH "T" \
        RETURN v.name, v.age;
+-----------------+-------+
| v.name          | v.age |
+-----------------+-------+
| "Tracy McGrady" | 39    |
+-----------------+-------+
| "Tony Parker"   | 36    |
+-----------------+-------+
| "Tim Duncan"    | 42    |
+-----------------+-------+
| "Tiago Splitter" | 34    |
+-----------------+-------+
```

如果您使用小写 `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.name STARTS WITH "t" \
        RETURN v.name, v.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.name ENDS WITH "r" \
        RETURN v.name, v.age;
+-----------------+-------+
| v.name          | v.age |
+-----------------+-------+
| "Vince Carter"  | 42    |
+-----------------+-------+
| "Tony Parker"   | 36    |
+-----------------+-------+
| "Tiago Splitter" | 34    |
+-----------------+-------+
```

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.name CONTAINS "Pa" \
        RETURN v.name, v.age;
+--------------+-------+
| v.name       | v.age |
+--------------+-------+
| "Paul George" | 28    |
+--------------+-------+
| "Tony Parker" | 36    |
+--------------+-------+
| "Paul Gasol"  | 38    |
+--------------+-------+
| "Chris Paul"  | 33    |
+--------------+-------+
```

结合NOT使用

您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.name ENDS WITH "R" \
        RETURN v.name, v.age;
+-------------------------+-------+
| v.name                  | v.age |
+-------------------------+-------+
| "Rajon Rondo"           | 33    |
+-------------------------+-------+
| "Rudy Gay"              | 32    |
+-------------------------+-------+
| "Dejounte Murray"       | 29    |
+-------------------------+-------+
| "Chris Paul"            | 33    |
+-------------------------+-------+
| "Carmelo Anthony"       | 34    |
```

```
+-------------------------+-------+
...
```

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.age IN range(20,25) \
        RETURN v.name, v.age;
+-------------------------+-------+
| v.name                  | v.age |
+-------------------------+-------+
| "Ben Simmons"           | 22    |
+-------------------------+-------+
| "Kristaps Porzingis"    | 23    |
+-------------------------+-------+
| "Luka Doncic"           | 20    |
+-------------------------+-------+
| "Kyle Anderson"         | 25    |
+-------------------------+-------+
| "Giannis Antetokounmpo" | 24    |
+-------------------------+-------+
| "Joel Embiid"           | 25    |
+-------------------------+-------+
```

结合NOT使用

```
nebula> MATCH (v:player) \
        WHERE v.age NOT IN range(20,25) \
        RETURN v.name AS Name, v.age AS Age \
        ORDER BY Age;
+--------------------+-----+
| Name               | Age |
+--------------------+-----+
| "Kyrie Irving"     | 26  |
+--------------------+-----+
| "Cory Joseph"      | 27  |
+--------------------+-----+
| "Damian Lillard"   | 28  |
+--------------------+-----+
| "Paul George"      | 28  |
+--------------------+-----+
| "Ricky Rubio"      | 28  |
+--------------------+-----+
...
```

3.7.7 YIELD

`YIELD` 定义nGQL查询的输出结果。

`YIELD` 可以引导子句或语句：

- `YIELD` 子句可以用于nGQL扩展语句中，例如 `GO` 、 `FETCH` 或 `LOOKUP` 。
- `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nGQL扩展。关于openCypher方式如何定义输出结果，请参见`RETURN` [#3.ngql-guide/8.clauses-and-options/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。

  > **说明**：nGQL暂不支持 `CALL[...YIELD]` 。

- 在nGQL中， `YIELD` 和openCypher中的 `RETURN` 类似。

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 $$.player.name AS Friend, $$.player.age AS Age;
+-----------------+-----+
| Friend          | Age |
+-----------------+-----+
| "Tony Parker"   | 36  |
+-----------------+-----+
```

```
| "Manu Ginobili" | 41  |
+-----------------+-----+
```

- FETCH 语句中使用 YIELD ：

```
nebula> FETCH PROP ON player "player100" \
        YIELD player.name;
+-------------+--------------+
| VertexID    | player.name  |
+-------------+--------------+
| "player100" | "Tim Duncan" |
+-------------+--------------+
```

- LOOKUP 语句中使用 YIELD ：

```
nebula> LOOKUP ON player WHERE player.name == "Tony Parker" \
        YIELD player.name, player.age;
=====================================
| VertexID | player.name | player.age |
=====================================
| 101      | 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 [#3.ngql-guide/8.clauses-and-options/where/:]。

复合查询中使用YIELD语句

在复合查询 [#3.ngql-guide/4.variable-and-composite-queries/1.composite-queries/:]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 follow._dst AS ID \
        | FETCH PROP ON player $-.ID \
        YIELD player.age AS Age \
        | YIELD AVG($-.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 follow.degree AS Degree, follow._dst 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) |
+-------------+
| 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      |
+--------+
```

3.7.8 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。

> **说明**：在nGQL扩展中，有与 `WITH` 类似的函数管道符 [#3.ngql-guide/5.operators/4.pipe/:]，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nGQL扩展中使用 `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" :star{} :person{} :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"})                   |
+------------------------------------------------------------------------+
```

示例2

1. 匹配点ID为 `player100` 的点。
2. 通过 `labels()` 函数将点的所有标签输出到一个列表。
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   |
+----------+
| "star"   |
+----------+
| "player" |
+----------+
| "person" |
+----------+
```

过滤聚合查询

`WITH` 可以在聚合查询中作为过滤器使用。

```
nebula> MATCH (v:player)-->(v2:player) \
        WITH DISTINCT v2 AS v2, v2.age AS Age \
        ORDER BY Age \
        WHERE Age<25 \
        RETURN v2.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.name AS Name \
        ORDER BY Name DESC \
        LIMIT 3 \
        RETURN collect(Name);
+-----------------------------------------------+
| COLLECT(Name)                                 |
+-----------------------------------------------+
| ["Yao Ming", "Vince Carter", "Tracy McGrady"] |
+-----------------------------------------------+
```

3.8 图空间语句

3.8.1 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` 语句可以通过指定名称创建一个新的图空间。

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 [#7.data-security/1.authentication/1.authentication/:]。

语法

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

IF NOT EXISTS

`IF NOT EXISTS` 关键字可以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。

> **说明**：仅检测图空间的名称，不会检测具体属性。

图空间名称

`graph_space_name` 在Nebula Graph实例中唯一标识一个图空间。

自定义图空间选项

您可以为新的图空间设置如下选项：

- `partition_num`

  指定图空间的分片数量。建议设置为5倍的集群硬盘数量。例如集群中有3个硬盘，建议设置15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`

  指定点ID的数据类型。可选值为 `FIXED_STRING(<N>)` 和 `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。默认值为 `FIXED_STRING(8)` 。

如果没有指定选项，Nebula Graph会使用默认值创建图空间。

示例

```
# 使用默认值。
nebula> CREATE SPACE my_space_1;

# 指定分片数量。
nebula> CREATE SPACE my_space_2(partition_num=10);

# 指定副本数量。
nebula> CREATE SPACE my_space_3(replica_factor=1);

# 指定点ID最大长度。
nebula> CREATE SPACE my_space_4(vid_type = FIXED_STRING(30));
```

创建图空间说明

尝试使用新创建的图空间可能会失败，因为创建是异步实现的。

Nebula Graph将在下一个心跳周期内完成图空间的创建，为了确保创建成功，可以使用如下方法之一：

- 在SHOW SPACES [#3.ngql-guide/9.space-statements/3.show-spaces/:]或DESCRIBE SPACE [#3.ngql-guide/9.space-statements/4.describe-space/:]语句的结果中查找新的图空间，如果找不到，请等待几秒重试。
- 等待两个心跳周期，例如20秒。

如果需要修改心跳间隔，请为所有配置文件 [#5.configurations-and-logs/1.configurations/1.configurations/:]修改参数`heartbeat_interval_secs`。

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。您可以执行如下命令检查分片的分布情况：

```
nebula> SHOW HOSTS;
+-----------+-------+--------+--------------+---------------------+------------------------+
| Host      | Port  | Status | Leader count | Leader distribution | Partition distribution |
+-----------+-------+--------+--------------+---------------------+------------------------+
| storaged0 | 9779  | ONLINE | 1            | nba:5               | nba:5                  |
+-----------+-------+--------+--------------+---------------------+------------------------+
| storaged1 | 9779  | ONLINE | 2            | test:1, nba:5       | nba:5, test:1          |
+-----------+-------+--------+--------------+---------------------+------------------------+
| storaged2 | 9779  | ONLINE | 1            | nba:5               | nba:5                  |
+-----------+-------+--------+--------------+---------------------+------------------------+
```

如果需要均衡负载，请执行如下命令：

```
nebula> BALANCE LEADER;
```

3.8.2 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 [#7.data-security/1.authentication/1.authentication/:]，否则会报错。

语法

```
USE <graph_space_name>;
```

示例

```
# 指定图空间space1作为工作空间。
nebula> USE space1;

# 检索图空间space1。
nebula> GO FROM 1 OVER edge1;

# 切换到图空间space2。
nebula> USE space2;

# 检索图空间space2。无法从space1读取任何数据，检索的点和边与space1无关。
nebula> GO FROM 2 OVER edge2;
```

> **说明**：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，您无法访问其他空间。检索新图空间的唯一方法是通过 `USE` 语句切换。Fabric Cypher可以在一条语句中使用两个图空间。

3.8.3 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" |
+-------+
| "nba" |
+-------+
```

创建图空间请参见CREATE SPACE [#3.ngql-guide/9.space-statements/1.create-space/:]。

3.8.4 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 nba;
+----+-------+------------------+----------------+---------+------------+--------------------+-------------+-----------+
| ID | Name  | Partition Number | Replica Factor | Charset | Collate    | Vid Type           | Atomic Edge | Group     |
+----+-------+------------------+----------------+---------+------------+--------------------+-------------+-----------+
| 1  | "nba" | 10               | 1              | "utf8"  | "utf8_bin" | "FIXED_STRING(32)" | "false"     | "default" |
+----+-------+------------------+----------------+---------+------------+--------------------+-------------+-----------+
```

3.8.5 DROP SPACE

`DROP SPACE` 语句可以删除指定图空间的所有内容。

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 [#7.data-security/1.authentication/1.authentication/:]。

语法

```
DROP SPACE [IF EXISTS] <graph_space_name>;
```

`IF NOT 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。

`DROP SPACE` 语句不会立刻删除硬盘上对应图空间的目录和文件，请使用 `USE` 语句指定其他任意图空间，然后执行 `SUBMIT JOB COMPACT` 。

> **警告**：请**谨慎**执行删除图空间操作。

3.9 标签语句

3.9.1 CREATE TAG

`CREATE TAG` 语句可以通过指定名称创建一个标签。

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标签权限 [#7.data-security/1.authentication/3.role-list/:]，否则会报错。

语法

创建标签前，需要先用 `USE` 语句指定工作空间。

```
CREATE TAG [IF NOT EXISTS] <tag_name>
    ([<create_definition>, ...])
    [tag_options];

<create_definition> ::=
    <prop_name> <data_type> [NULL | NOT NULL]

<tag_options> ::=
    <option> [, <option> ...]

<option> ::=
    TTL_DURATION [=] <ttl_duration>
    | TTL_COL [=] <prop_name>
    | DEFAULT <default_value>
```

标签名称

- `IF NOT EXISTS` ：您可以使用 `IF NOT EXISTS` 关键字检测待创建的标签是否存在，只有不存在时，才会创建标签。

  > **说明**：仅检测标签的名称，不会检测具体属性。

- `tag_name` ：每个图空间内的标签必须是唯一的。标签名称设置后无法修改。允许的标签名称规则与图空间名称规则相同，详情请参见关键字和保留字 [#3.ngql-guide/20.appendix/keywords-and-reserved-words/:]。

标签属性

- `prop_name`

属性名称。每个标签中的属性名称必须唯一。

- `data_type`

  属性的数据类型。数据类型的完整说明，请参见数值类型 [#3.ngql-guide/3.data-types/1.numeric/:]、布尔 [#3.ngql-guide/3.data-types/2.boolean/:]等文档。

- `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。

TTL（TIME-TO-LIVE）

- `TTL_DURATION`

  指定属性存活时间。超时的属性将会过期。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` 。

一个标签只能指定一个字段为 `TTL_COL` 。更多TTL的信息请参见TTL [#3.ngql-guide/8.clauses-and-options/ttl-options/:]。

示例

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

# 创建没有属性的标签。
nebula> CREATE TAG no_property();

# 创建包含默认值的标签。
nebula> CREATE TAG player_with_default(name string, age int DEFAULT 20);

# 对字段create_time设置TTL为100秒。
nebula> CREATE TAG woman(name string, age int, \
        married bool, salary double, create_time timestamp) \
        TTL_DURATION = 100, TTL_COL = "create_time";
```

创建标签说明

尝试使用新创建的标签可能会失败，因为创建是异步实现的。

Nebula Graph将在下一个心跳周期内完成标签的创建，为了确保创建成功，可以使用如下方法之一：

- 在`SHOW TAGS` [#3.ngql-guide/10.tag-statements/4.show-tags/:]语句的结果中查找新的标签，如果找不到，请等待几秒重试。
- 等待两个心跳周期，例如20秒。

如果需要修改心跳间隔，请为所有配置文件 [#5.configurations-and-logs/1.configurations/1.configurations/:]修改参数 `heartbeat_interval_secs` 。

3.9.2 DROP TAG

`DROP TAG` 语句可以删除当前工作空间内的指定标签。

一个点可以有一个或多个标签。

- 如果点只有一个标签，删除这个标签后，您就**无法访问**这个点，下次Compaction操作时会删除该点。点上的边仍然存在。
- 如果点有多个标签，删除其中一个标签，仍然可以访问这个点，但是**无法访问**已删除标签定义的所有属性。

删除标签操作仅删除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

- 您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 [#7.data-security/1.authentication/3.role-list/:]。
- 确保标签不包含任何索引，否则 `DROP TAG` 时会报冲突错误 `[ERROR (-8)]: Conflict!` 。删除索引请参见drop index [#3.ngql-guide/14.native-index-statements/6.drop-native-index/:]。

语法

```
DROP TAG [IF EXISTS] <tag_name>;
```

- `IF NOT EXISTS` ：检测待删除的标签是否存在，只有存在时，才会删除标签。
- `tag_name` ：指定要删除的标签名称。一次只能删除一个标签。

示例

```
nebula> CREATE TAG test(p1 string, p2 int);
nebula> DROP TAG test;
```

3.9.3 ALTER TAG

`ALTER TAG`语句可以修改标签的结构。例如增删属性、修改数据类型，也可以为属性设置、修改TTL [#3.ngql-guide/8.clauses-and-options/ttl-options/:]（Time-To-Live）。

前提条件

- 您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 [#7.data-security/1.authentication/3.role-list/:]。
- 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 (-8)]: Conflict!`。删除索引请参见drop index [#3.ngql-guide/14.native-index-statements/6.drop-native-index/:]。

语法

```
ALTER TAG <tag_name>
    <alter_definition> [, alter_definition] ...]
    [ttl_definition [, ttl_definition] ... ];

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
```

- `tag_name`：指定要修改的标签名称。一次只能修改一个标签。请确保要修改的标签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 t1 (p1 string, p2 int);
nebula> ALTER TAG t1 ADD (p3 int, p4 string);
nebula> ALTER TAG t1 TTL_DURATION = 2, TTL_COL = "p2";
```

修改标签说明

尝试使用刚修改的标签可能会失败，因为修改是异步实现的。

Nebula Graph将在下一个心跳周期内完成标签的修改，为了确保修改成功，可以使用如下方法之一：

- 在`DESCRIBE TAG` [#3.ngql-guide/10.tag-statements/5.describe-tag/:]语句的结果中查看标签信息，确认修改成功。如果没有修改成功，请等待几秒重试。
- 等待两个心跳周期，例如20秒。

如果需要修改心跳间隔，请为所有配置文件 [#5.configurations-and-logs/1.configurations/1.configurations/:]修改参数`heartbeat_interval_secs`。

3.9.4 SHOW TAGS

`SHOW TAGS` 语句显示当前图空间内的所有标签名称。

执行 `SHOW TAGS` 语句不需要任何权限，但是返回结果由您登录的用户权限 [#7.data-security/1.authentication/3.role-list/:] 决定。

语法

```
SHOW TAGS;
```

示例

```
nebula> SHOW TAGS;
+----------+
| Name     |
+----------+
| "player" |
+----------+
| "team"   |
+----------+
```

3.9.5 DESCRIBE TAG

`DESCRIBE TAG` 显示指定标签的详细信息，例如字段名称、数据类型等。

前提条件

您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 [#7.data-security/1.authentication/3.role-list/:]。

语法

```
DESC[RIBE] TAG <tag_name>;
```

`DESCRIBE` 可以缩写为 `DESC` 。

示例

```
nebula> DESCRIBE TAG player;
+--------+----------+-------+-----------+
| Field  | Type     | Null  | Default   |
+--------+----------+-------+-----------+
| "name" | "string" | "YES" | __EMPTY__ |
+--------+----------+-------+-----------+
| "age"  | "int64"  | "YES" | __EMPTY__ |
+--------+----------+-------+-----------+
```

3.10 边类型语句

3.10.1 CREATE EDGE

`CREATE EDGE` 语句可以通过指定名称创建一个边类型。

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边类型权限 [#7.data-security/1.authentication/3.role-list/:]，否则会报错。

语法

创建边类型前，需要先用 `USE` 语句指定工作空间。

```
CREATE EDGE [IF NOT EXISTS] <edge_type_name>
    ([<create_definition>, ...])
    [edge_type_options];

<create_definition> ::=
    <prop_name> <data_type>

<edge_type_options> ::=
    <option> [, <option> ...]

<option> ::=
    TTL_DURATION [=] <ttl_duration>
    | TTL_COL [=] <prop_name>
    | DEFAULT <default_value>
```

边类型名称

- `IF NOT EXISTS` ：检测待创建的边类型是否存在，只有不存在时，才会创建边类型。

  > **说明**：仅检测边类型的名称，不会检测具体属性。 - `edge_type_name` ：每个图空间内的边类型必须是唯一的。边类型名称设置后无法修改。允许的边类型名称规则与图空间名称规则相同，详情请参见关键字和保留字 [#3.ngql-guide/20.appendix/keywords-and-reserved-words/:]。

边类型属性

- `prop_name`

属性名称。每个边类型中的属性名称必须唯一。

- `data_type`

  属性的数据类型。数据类型的完整说明，请参见数值类型 [#3.ngql-guide/3.data-types/1.numeric/:]、布尔 [#3.ngql-guide/3.data-types/2.boolean/:]等文档。

- `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。

TTL（TIME-TO-LIVE）

- `TTL_DURATION`

  指定属性存活时间。超时的属性将会过期。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` 。

一个边类型只能指定一个字段为 `TTL_COL` 。更多TTL的信息请参见TTL [#3.ngql-guide/8.clauses-and-options/ttl-options/:]。

示例

```
nebula> CREATE EDGE follow(degree int);

# 创建没有属性的边类型。
nebula> CREATE EDGE no_property();

# 创建包含默认值的边类型。
nebula> CREATE EDGE follow_with_default(degree int DEFAULT 20);

# 对字段p2设置TTL为100秒。
nebula> CREATE EDGE e1(p1 string, p2 int, p3 timestamp) \
        TTL_DURATION = 100, TTL_COL = "p2";
```

3.10.2 DROP EDGE

`DROP EDGE` 语句可以删除当前工作空间内的指定边类型。

一个边只能有一个边类型，删除这个边类型后，您就**无法访问**这个边，下次Compaction操作时会删除该边。

删除边类型操作仅删除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

- 您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 [#7.data-security/1.authentication/3.role-list/:]。
- 确保边类型不包含任何索引，否则 `DROP EDGE` 时会报冲突错误 `[ERROR (-8)]: Conflict!` 。删除索引请参见drop index [#3.ngql-guide/14.native-index-statements/6.drop-native-index/:]。

语法

```
DROP EDGE [IF EXISTS] <edge_type_name>
```

- `IF NOT EXISTS` ：检测待删除的边类型是否存在，只有存在时，才会删除边类型。
- `edge_type_name` ：指定要删除的边类型名称。一次只能删除一个边类型。

示例

```
nebula> CREATE EDGE e1(p1 string, p2 int);
nebula> DROP EDGE e1;
```

3.10.3 ALTER EDGE

`ALTER EDGE` 语句可以修改边类型的结构。例如增删属性、修改数据类型，也可以为属性设置、修改TTL [#3.ngql-guide/8.clauses-and-options/ttl-options/:]（Time-To-Live）。

前提条件

- 您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 [#7.data-security/1.authentication/3.role-list/:]。
- 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 (-8)]: Conflict!` 。删除索引请参见drop index [#3.ngql-guide/14.native-index-statements/6.drop-native-index/:]。

语法

```
ALTER EDGE <edge_type_name>
    <alter_definition> [, alter_definition] ...]
    [ttl_definition [, ttl_definition] ... ]

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` ：指定要修改的边类型名称。一次只能修改一个边类型。请确保要修改的边类型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 e1(p1 string, p2 int);
nebula> ALTER EDGE e1 ADD (p3 int, p4 string);
nebula> ALTER EDGE e1 TTL_DURATION = 2, TTL_COL = "p2";
```

修改边类型说明

尝试使用刚修改的边类型可能会失败，因为修改是异步实现的。

Nebula Graph将在下一个心跳周期内完成边类型的修改，为了确保修改成功，可以使用如下方法之一：

- 在`DESCRIBE EDGE` [#3.ngql-guide/11.edge-type-statements/5.describe-edge/:]语句的结果中查看边类型信息，确认修改成功。如果没有修改成功，请等待几秒重试。
- 等待两个心跳周期，例如20秒。

如果需要修改心跳间隔，请为所有配置文件 [#5.configurations-and-logs/1.configurations/1.configurations/:]修改参数 `heartbeat_interval_secs` 。

3.10.4 SHOW EDGES

`SHOW EDGES` 语句显示当前图空间内的所有边类型名称。

执行 `SHOW EDGES` 语句不需要任何权限，但是返回结果由您登录的用户权限 [#7.data-security/1.authentication/3.role-list/:] 决定。

语法

```
SHOW EDGES;
```

示例

```
nebula> SHOW EDGES;
+----------+
| Name     |
+----------+
| "follow" |
+----------+
| "serve"  |
+----------+
```

3.10.5 DESCRIBE EDGE

`DESCRIBE EDGE` 显示指定边类型的详细信息，例如字段名称、数据类型等。

前提条件

您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 [#7.data-security/1.authentication/3.role-list/:]。

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   |
+----------+---------+-------+-----------+
| "degree" | "int64" | "YES" | __EMPTY__ |
+----------+---------+-------+-----------+
```

3.11 点语句

3.11.1 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 [#7.data-security/1.authentication/3.role-list/:]，否则会报错。

语法

```
INSERT VERTEX <tag_name> (<prop_name_list>) [, <tag_name> (<prop_name_list>), ...]
     {VALUES | VALUE} VID: (<prop_value_list>[, <prop_value_list>])

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

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

- `tag_name` ：点关联的标签（点类型）。标签必须提前创建，详情请参见CREATE TAG [#3.ngql-guide/10.tag-statements/1.create-tag/:]。
- `prop_name_list` ：需要设置的属性名称列表。
- `VID` ：点ID。在Nebula Graph 2.0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 [#3.ngql-guide/9.space-statements/1.create-space/:]。
- `prop_value_list` ：根据 `prop_name_list` 填写属性值。如果属性值和标签中的数据类型不匹配，会返回错误。如果没有填写属性值，而标签中对应的属性设置为 `NOT NULL` ，也会返回错误。详情请参见CREATE TAG [#3.ngql-guide/10.tag-statements/1.create-tag/:]。

示例

```
# 插入不包含属性的点。
nebula> CREATE TAG t1();
nebula> INSERT VERTEX t1() VALUE "10":();
```

```
nebula> CREATE TAG 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 t3(p1 int);
nebula> CREATE TAG t4(p2 string);

# 一次插入两个标签的属性到同一个点。
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";
+---------------------------------+
| vertices_                       |
+---------------------------------+
| ("11" :t2{age: 15, name: "n4"}) |
+---------------------------------+
```

```
nebula> CREATE TAG 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 (-8)]: Storage Error: The not null field cannot be null.

# 属性p3为默认值NULL。
nebula> INSERT VERTEX t5(p1, p2) VALUES "003":("cd", 5);
nebula> FETCH PROP ON t5 "003";
+-----------------------------------------+
| vertices_                               |
+-----------------------------------------+
| ("003" :t5{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";
+-----------------------------------------+
| vertices_                               |
+-----------------------------------------+
| ("004" :t5{p1: "shala", p2: 4, p3: __NULL__}) |
+-----------------------------------------+
```

3.11.2 DELETE VERTEX

`DELETE VERTEX` 语句可以删除点，以及点关联的出边和入边。

`DELETE VERTEX` 语句一次可以删除一个或多个点。您可以结合管道符一起使用，详情请参见管道符 [#3.ngql-guide/5.operators/4.pipe/:]。

语法

```
DELETE VERTEX <vid> [, <vid> ...];
```

示例

```
nebula> DELETE VERTEX "team1";
```

```
# 结合管道符，删除符合条件的点。
nebula> GO FROM "player100" OVER serve YIELD serve._dst AS id | DELETE VERTEX $-.id;
```

删除过程

Nebula Graph检索点的出边和入边并删除这些边，然后删除点的信息。

> **说明**：目前还不能保证操作的原子性，如果发生故障请重试。

3.11.3 UPDATE VERTEX

`UPDATE VERTEX` 语句可以修改点上标签的属性。一次只能修改一个标签。

Nebula Graph支持CAS（compare and set）操作。

语法

```
UPDATE VERTEX <vid> SET <update_columns>
[WHEN <condition>] [YIELD <columns>];
```

- `vid` ：需要修改的点ID。
- `update_columns` ：需要修改的属性和值。例如 `tag1.col1 = $^.tag2.col2 + 1` 表示将 `tag1.col1` 修改为 `tag2.col2+1` 。

  > **说明：** `$^` 表示要更新的点.

- `condition` ：结合 `WHEN` 使用的约束条件，满足条件时才修改。 `condition` 支持表达式。
- `columns` ：结合 `YIELD` 指定要返回的输出结果。

示例

```
nebula> UPDATE VERTEX "player100" \
        SET player.age = $^.player.age + 1 \
        WHEN $^.player.name == "Tim Duncan" \
        YIELD $^.player.name AS name, $^.player.age AS age;
+--------------+-----+
| name         | age |
+--------------+-----+
| "Tim Duncan" | 43  |
+--------------+-----+
```

3.11.4 UPSERT VERTEX

`UPSERT VERTEX` 语句结合 `UPDATE` 和 `INSERT`，如果点存在，会更新点的属性；如果点不存在，会插入新的点。 `UPSERT VERTEX` 一次只能修改一个标签。

`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。

> **禁止**：禁止在高并发写操作的情况下使用 `UPSERT` 语句。

语法

```
UPSERT VERTEX <vid> SET <update_columns> [WHEN <condition>] [YIELD <columns>];
```

- `vid` ：需要 `UPSERT` 的点ID。
- `update_columns` ：需要 `UPSERT` 的属性和值。例如 `tag1.col1 = $^.tag2.col2 + 1` 表示将 `tag1.col1` 修改为 `tag2.col2+1` 。

  > **说明**： `$^` 表示要 `UPSERT` 的点.

- `condition` ：结合 `WHEN` 使用的约束条件，满足条件时才执行 `UPSERT` 。 `condition` 支持表达式。
- `columns` ：结合 `YIELD` 指定要返回的输出结果。

注意事项

- 如果点不存在，则不论 `WHEN` 子句中的条件是否满足，都会创建一个新的点。没有被 `SET` 语句指定的属性使用默认值。如果没有默认值，则返回错误。
- 如果点存在，而且满足 `WHEN` 子句的条件，则修改这个点的属性。
- 如果点存在，但是不满足 `WHEN` 子句的条件，则不会执行修改操作。

示例

```
nebula> INSERT VERTEX player(name, age) VALUES "player111":("Ben Simmons", 22);
nebula> UPSERT VERTEX "player111" \
        SET player.name = "Dwight Howard", player.age = $^.player.age + 11 \
        WHEN $^.player.name == "Ben Simmons" \
        AND $^.player.age > 20  \
        YIELD $^.player.name AS Name, $^.player.age AS Age;
+-----------------+-----+
| Name            | Age |
+-----------------+-----+
| "Dwight Howard" | 33  |
+-----------------+-----+
```

```
nebula> FETCH PROP ON * "player123";
Empty set (Time spent: 3.069/4.382 ms)
nebula> UPSERT VERTEX "player123" SET player.age = $^.player.age + 1;
```

在上述示例中，如果点 player123 不存在， age 的默认值为 NULL ，则 player123 的 player.age 为 NULL ；如果 age 有默认值，则 player.age 为 默认值+1 。

```
# age默认值为0。
nebula> CREATE TAG person(followers int, age int DEFAULT 0);

# 顺序执行，即followers为1（0+1），age为8。
nebula> UPSERT VERTEX "300" SET person.followers = $^.person.age + 1,  person.age = 8;

# 顺序执行，即age为8，followers为9（8+1）。
nebula> UPSERT VERTEX "300" SET person.age = 8, person.followers = $^.person.age + 1;
```

3.12 边语句

3.12.1 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）。

语法

```
INSERT EDGE <edge_type> ( <prop_name_list> ) {VALUES | VALUE}
<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> ] ...]
```

- `<edge_type>` ：边关联的边类型，只能指定一个边类型。边类型必须提前创建，详情请参见CREATE EDGE [#3.ngql-guide/11.edge-type-statements/1.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` 。rank值可以用来区分边类型、起始点、目的点都相同的边。

  > **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。如果属性值和边类型中的数据类型不匹配，会返回错误。如果没有填写属性值，而边类型中对应的属性设置为 `NOT NULL` ，也会返回错误。详情请参见CREATE EDGE [#3.ngql-guide/11.edge-type-statements/1.create-edge/:]。

示例

```
# 插入不包含属性的边。
nebula> CREATE EDGE 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 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);
```

```
# 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";
+-----------------------------------------+
| edges_                                  |
+-----------------------------------------+
| [:e2 "11"->"13" @0 {age: 14, name: "n1"}] |
+-----------------------------------------+
```

3.12.2 DELETE EDGE

`DELETE EDGE` 语句可以删除边。一次可以删除一条或多条边。您可以结合管道符一起使用，详情请参见管道符 [#3.ngql-guide/5.operators/4.pipe/:]。

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 [#3.ngql-guide/12.vertex-statements/4.delete-vertex/:]。

> **说明**：目前还不能保证操作的原子性，如果发生故障请重试。

语法

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

示例

```
nebula> DELETE EDGE serve "player100" -> "team200"@0;
```

```
# 结合管道符，删除符合条件的边。
nebula> GO FROM "player100" OVER follow \
        WHERE follow._dst == "team200" \
        YIELD follow._src AS src, follow._dst AS dst, follow._rank AS rank \
        | DELETE EDGE follow $-.src->$-.dst @ $-.rank;
```

3.12.3 UPDATE EDGE

`UPDATE EDGE` 语句可以修改边上边类型的属性。一次只能修改一个边类型。

Nebula Graph支持CAS（compare and set）操作。

语法

```
UPDATE EDGE <src_vid> -> <dest_vid> [@<rank>] OF <edge_type> SET <update_properties> [WHEN <condition>] [YIELD <properties>];
```

- `update_properties` ：需要修改的属性和值。例如 `e1.col1 = $^.e1.col2 + 1` 表示将 `e1.col1` 修改为 `e1.col2+1` 。

  > **说明**： `$^` 表示要更新的边.

- `condition` ：结合 `WHEN` 使用的约束条件，满足条件时才修改。 `condition` 支持表达式。

- `properties` ：结合 `YIELD` 指定要返回的输出结果。

示例

```
nebula> UPDATE EDGE "player100" -> "team204"@0 \
        OF serve \
        SET start_year = start_year + 1 \
        YIELD serve.start_year;
+------------------+
| serve.start_year |
+------------------+
| 1999             |
+------------------+
```

3.12.4 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` 一次只能修改一个边类型。

`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。

> **禁止**：禁止在高并发写操作的情况下使用 `UPSERT` 语句。

语法

```
UPSERT EDGE <src_vid> -> <dst_vid> [@rank] OF <edge_type> SET <update_properties> [WHEN <condition>] [YIELD <properties>]
```

- `update_properties` ：需要 `UPSERT` 的属性和值。例如 `e1.col1 = $^.e1.col2 + 1` 表示将 `e1.col1` 修改为 `e1.col2+1` 。

  > **说明**： `$^` 表示要 `UPSERT` 的边. - `condition` ：结合 `WHEN` 使用的约束条件，满足条件时才执行 `UPSERT` 。 `condition` 支持表达式。

- `columns` ：结合 `YIELD` 指定要返回的输出结果。

注意事项

- 如果边不存在，则不论 `WHEN` 子句中的条件是否满足，都会创建一个新的边。没有被 `SET` 语句指定的属性使用默认值。如果没有默认值，则返回错误。
- 如果边存在，而且满足 `WHEN` 子句的条件，则修改这个边的属性。
- 如果边存在，但是不满足 `WHEN` 子句的条件，则不会执行修改操作。

示例

```
nebula> INSERT EDGE serve(start_year, end_year) VALUES "player100" -> "team200":(1997, 2016);
nebula> UPSERT EDGE "player100" -> "team200" \
        OF serve \
        SET start_year = serve.start_year + 2 \
        WHEN serve.end_year == 2016 \
        YIELD serve.start_year AS Start, serve.end_year AS End;
+-------+------+
| Start | End  |
+-------+------+
| 1999  | 2016 |
+-------+------+

nebula> FETCH PROP ON serve "player100" -> "team200";
+------------------------------------------------------------------------+
| edges_                                                                 |
+------------------------------------------------------------------------+
| [:serve "player100"->"team200" @0 {end_year: 2016, start_year: 1999}] |
+------------------------------------------------------------------------+
```

3.13 原生索引

3.13.1 CREATE INDEX

`CREATE INDEX` 语句可以为现存的标签、边类型或属性创建原生索引。

大多数图查询都是从属性标识的点或边的列表开始检索的。索引可以让大规模图数据库的全局检索更加高效。

> **说明**：如何创建全文索引，请参见部署全文索引 [#4.deployment-and-installation/6.deploy-text-based-index/2.deploy-es/:]。

前提条件

创建索引之前，请确保相关的标签或边类型已经创建。如何创建标签和边类型，请参见CREATE TAG [#3.ngql-guide/10.tag-statements/1.create-tag/:]和CREATE EDGE [#3.ngql-guide/11.edge-type-statements/1.create-edge/:]。

使用索引必读

正确使用索引可以加速查询，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。请**不要随意**在生产环境中使用索引，除非您很清楚使用索引对业务的影响。

如果您必须使用索引，建议您按照如下步骤：

1. 导入数据至Nebula Graph。
2. 创建索引。
3. 重建索引 [#3.ngql-guide/14.native-index-statements/4.rebuild-native-index/:]。
4. 使用LOOKUP [#3.ngql-guide/7.general-query-statements/5.lookup/:]或MATCH [#3.ngql-guide/7.general-query-statements/2.match/:]语句查询数据。
   - 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。
   - 创建索引后，您**必须**为已存在的数据重建索引，否则不能索引已存在的数据，导致无法在 `MATCH` 和 `LOOKUP` 语句中返回这些数据。
   - 您不需要指定使用哪个索引，Nebula Graph会自动计算。

语法

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

- `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。
- `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。

    > **说明**：长索引会降低Storage服务的扫描性能，以及占用更多内存。建议您将索引长度设置为和要索引的最长字符串相同。索引长度最长为255，超过部分会被截断。

  - 为**定长**字符串属性创建索引时，使用 `prop_name` 设置要索引的属性即可，索引长度为字符串长度。
  - 为标签或边类型本身创建索引时，请忽略 `prop_name_list` 。

    > **禁止**：如果您已经为标签或边类型中的任何属性创建了索引，请不要再为标签或边类型本身创建索引。

创建索引说明

尝试使用新创建的索引可能会失败，因为创建是异步实现的。

Nebula Graph将在下一个心跳周期内完成索引的创建，为了确保创建成功，可以使用如下方法之一：

- 在`SHOW TAG/EDGE INDEXES` [#3.ngql-guide/14.native-index-statements/2.show-native-indexes/:]语句的结果中查找新的索引，如果找不到，请等待几秒重试。
- 等待两个心跳周期，例如20秒。

如果需要修改心跳间隔，请为所有配置文件 [#5.configurations-and-logs/1.configurations/1.configurations/:]修改参数 `heartbeat_interval_secs` 。

创建标签/边类型索引

```
nebula> CREATE TAG INDEX player_index on player();
```

```
nebula> CREATE EDGE INDEX like_index on like();
```

为标签或边类型创建索引后，您可以使用 `LOOKUP` 语句检索带有标签的所有点的ID，或者起始点ID、目的点ID，以及带有边类型的所有边的rank。详情请参见LOOKUP [#3.ngql-guide/7.general-query-statements/5.lookup/:]。

创建单个属性索引

```
nebula> CREATE TAG INDEX player_index_0 on player(name(10));
```

上述示例是为所有包含标签 `player` 的点创建属性 `name` 的索引，索引长度为10，即使用属性 `name` 的前10个字符创建索引。

```
# 变长字符串需要指定索引长度。
nebula> CREATE TAG var_string(p1 string);
nebula> CREATE TAG INDEX var ON var_string(p1(10));

# 定长字符串不需要指定索引长度。
```

```
nebula> CREATE TAG fix_string(p1 FIXED_STRING(10));
nebula> CREATE TAG INDEX fix ON fix_string(p1);
```

```
nebula> CREATE EDGE INDEX follow_index_0 on follow(degree);
```

创建复合属性索引

多个属性上的索引称为复合索引。

> **说明**：不支持跨标签或边类型创建复合索引。

```
nebula> CREATE TAG INDEX player_index_1 on player(name(10), age);
```

3.13.2 SHOW INDEXES

`SHOW INDEXES` 语句可以列出当前图空间内的所有标签和边类型（包括属性）的索引。

语法

```
SHOW {TAG | EDGE} INDEXES;
```

示例

```
nebula> SHOW TAG INDEXES;
+------------------+
| Names            |
+------------------+
| "fix"            |
+------------------+
| "player_index_0" |
+------------------+
| "player_index_1" |
+------------------+
| "var"            |
+------------------+

nebula> SHOW EDGE INDEXES;
+------------------+
| Names            |
+------------------+
| "follow_index_0" |
+------------------+
```

3.13.3 SHOW CREATE INDEX

`SHOW CREATE INDEX` 展示创建标签或者边类型时使用的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

您可以先运行 `SHOW TAG INDEXES` 查看有哪些标签索引，然后用 `SHOW CREATE TAG INDEX` 查看指定索引的创建信息。

```
nebula> SHOW TAG INDEXES;
+------------------+
| Names            |
+------------------+
| "player_index_0" |
+------------------+
| "player_index_1" |
+------------------+

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)`                                     |
|                  | )"                                              |
+------------------+-------------------------------------------------+
```

边类型索引可以用类似的方法查询：

```
nebula> SHOW EDGE INDEXES;
+----------------+
| Names          |
+----------------+
| "index_follow" |
+----------------+

nebula> SHOW CREATE EDGE INDEX index index_follow;
+-----------------+-------------------------------------------------+
| Edge Index Name | Create Edge Index                               |
+-----------------+-------------------------------------------------+
| "index_follow"  | "CREATE EDGE INDEX `index_follow` ON `follow` ( |
|                 |  `degree`                                       |
|                 | )"                                              |
+-----------------+-------------------------------------------------+
```

3.13.4 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"            |
+--------+--------------------+
```

3.13.5 REBUILD INDEX

为保障服务性能，索引不会自动对其创建之前已存在的数据生效。请在创建索引后，选择合适的时间为存量数据重建索引。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。使用索引的详情请参见CREATE INDEX [#3.ngql-guide/14.native-index-statements/1.create-native-index/:]。

> **说明**：重建索引期间，所有查询都会跳过索引并执行顺序扫描，返回结果可能不一致。

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，您可以使用命令 `SHOW {TAG | EDGE} INDEX STATUS` 检查索引是否重建完成。详情请参见SHOW INDEX STATUS [#3.ngql-guide/14.native-index-statements/5.show-native-index-status/:]。

示例

```
nebula> CREATE TAG person(name string, age int, gender string, email string);
nebula> CREATE TAG INDEX 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 |
+------------+
| 66         |
+------------+

# 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 66;
+----------------+---------------------+------------+-------------------------+-------------------------+
| Job Id(TaskId) | Command(Dest)       | Status     | Start Time              | Stop Time               |
+----------------+---------------------+------------+-------------------------+-------------------------+
| 66             | "REBUILD_TAG_INDEX" | "FINISHED" | 2021-03-31T03:35:21.000 | 2021-03-31T03:35:21.000 |
+----------------+---------------------+------------+-------------------------+-------------------------+
```

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 [#3.ngql-guide/14.native-index-statements/18.operation-and-maintenance-statements/4.job-statements/:show_job]。

历史版本兼容性

在Nebula Graph 2.0中，不需要也不支持选项 `OFFLINE` 。

3.13.6 SHOW INDEX STATUS

`SHOW INDEX STATUS` 语句可以查看索引名称和对应的状态。

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

> **说明**：如何创建索引请参见CREATE INDEX [#3.ngql-guide/14.native-index-statements/1.create-native-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"   |
+----------------------+--------------+
```

3.13.7 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 [#7.data-security/1.authentication/3.role-list/:]，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;
```

3.14 全文索引

3.14.1 索引介绍

索引可以加速图查询，Nebula Graph支持两种类型索引：原生索引和全文索引。

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：

- 包括标签索引和边类型索引。
- 必须手动重建索引（`REBUILD INDEX`）。
- 支持创建同一个标签或边类型的多个属性的索引（复合索引），但是不能跨标签或边类型。
- 复合索引可以实现部分匹配检索，遵循最左匹配原则。详情请参见LOOKUP FAQ [#3.ngql-guide/7.general-query-statements/5.lookup/:faq]。
- 不支持 `CONTAINS` 和 `STARTS WITH` 等字符串操作符。

原生索引操作

- CREATE INDEX [#3.ngql-guide/14.native-index-statements/1.create-native-index/:]
- SHOW CREATE INDEX [#3.ngql-guide/14.native-index-statements/2.1.show-create-index/:]
- SHOW INDEXES [#3.ngql-guide/14.native-index-statements/2.show-native-indexes/:]
- DESCRIBE INDEX [#3.ngql-guide/14.native-index-statements/3.describe-native-index/:]
- REBUILD INDEX [#3.ngql-guide/14.native-index-statements/4.rebuild-native-index/:]
- SHOW INDEX STATUS [#3.ngql-guide/14.native-index-statements/5.show-native-index-status/:]
- DROP INDEX [#3.ngql-guide/14.native-index-statements/6.drop-native-index/:]
- LOOKUP [#3.ngql-guide/7.general-query-statements/5.lookup/:]
- MATCH [#3.ngql-guide/7.general-query-statements/2.match/:]

全文索引

全文索引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`。

> **说明**：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 [#4.deployment-and-installation/6.deploy-text-based-index/2.deploy-es/:]和部署listener [#4.deployment-and-installation/6.deploy-text-based-index/3.deploy-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 [#3.ngql-guide/15.full-text-index-statements/1.search-with-text-based-index/:]。

NULL值说明

暂不支持对值为NULL的属性创建索引。

范围查询

原生索引除了可以查询单个结果之外，还可以执行范围查询。当前仅支持对数字、日期和时间类型的属性进行范围查询。

3.14.2 全文索引限制

本文介绍全文索引的限制，请在使用全文索引前仔细阅读。

目前为止，全文索引有如下限制：

- 字符串索引的最大长度为256字节，如果需要为超过256字节的数据创建索引，请逆序存储数据。
- 不能同时为多个属性创建全文索引。
- 全文搜索语句 `LOOKUP` / `MATCH` 中的 `WHERE` 子句不支持逻辑运算 `AND` 和 `OR` 。
- 全文索引不支持多个标签的搜索。
- 中文没有单词分隔符，所以内置的全文解析器不能确定中文单词的起始和结束位置。您需要安装elasticsearch-analysis-ik [https://github.com/medcl/elasticsearch-analysis-ik]来解析中文。
- 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。
- 目前不支持重建或修改Elasticsearch索引。
- `LOOKUP` 和 `MATCH` 语句不支持管道符，文档中的示例除外。
- 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。
- Elasticsearch创建索引可能需要一些时间，如果Nebula Graph报警未找到索引，请耐心等待索引生效。

3.14.3 部署全文索引

Nebula Graph的全文索引是基于Elasticsearch [https://en.wikipedia.org/wiki/Elasticsearch]实现，这意味着您可以使用Elasticsearch全文查询语言来检索您想要的内容。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，请确认您已经了解全文索引的使用限制 [#4.deployment-and-installation/6.deploy-text-based-index/1.text-based-index-restrictions/:]。

部署Elasticsearch集群

部署Elasticsearch集群请参见Elasticsearch官方文档 [https://www.elastic.co/guide/en/cloud-on-k8s/current/k8s-deploy-elasticsearch.html]。

当Elasticsearch集群启动时，请添加Nebula Graph全文索引的模板文件。以下面的模板为例：

```
{
 "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"}
        }
  }
}
```

请确保您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"}
```

您可以配置Elasticsearch来满足您的业务需求，如果需要定制Elasticsearch，请参见Elasticsearch官方文档 [https://www.elastic.co/guide/en/elasticsearch/reference/current/settings.html]。

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。您必须使用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> [,<username>, <password>]), (<elastic_ip:port>), ...];
```

示例

```
nebula> SIGN IN TEXT SERVICE (127.0.0.1:9200);
```

> **说明**：Elasticsearch默认没有用户名和密码，如果您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 |
+-------------+------+
| "127.0.0.1" | 9200 |
+-------------+------+
| "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;
```

3.14.4 部署Raft listener

全文索引的数据是异步写入Elasticsearch集群的，通过Raft listener（简称listener）这个独立进程，从Storage服务获取数据，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 [#4.deployment-and-installation/6.deploy-text-based-index/1.text-based-index-restrictions/:]。
- 已经部署Nebula Graph集群 [#4.deployment-and-installation/deploy-nebula-graph-cluster/:]。
- 已经准备至少一台额外的服务器，为Raft listener提供Storage服务。

注意事项

- 为listener提供服务的Storage服务必须与集群中其他所有Nebula Graph服务具有相同或更新的版本。
- 目前您只能为一个图空间一次性的添加listener。向已经存在listener的图空间添加listener会失败。请在一个语句里一次性添加多个listener。

部署流程

第一步：准备LISTENER的配置文件

你必须在需要部署listener的机器上准备listener的配置文件。文件名称必须为 `nebula-storaged-listener.conf` 。您可以参考提供的模板 [https://github.com/vesoft-inc/nebula-storage/blob/master/conf/nebula-storaged-listener.conf.production]。

> **说明**：在配置文件中请使用真实的IP地址替换域名或回送地址（127.0.0.1）。

第二步：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` 是存储listener配置文件的路径。

第三步：添加LISTENER到NEBULA GRAPH

连接Nebula Graph [#2.quick-start/3.connect-to-nebula-graph/:]，然后执行`USE <space>` [#3.ngql-guide/9.space-statements/2.use-space/:]进入需要创建全文索引的图空间。最后执行如下命令添加listener：

> **说明**：listener必须使用真实的IP地址。

```
ADD LISTENER ELASTICSEARCH <listener_ip:port> [,<listener_ip:port>, ...]
```

请在一个语句里一次性添加多个listener。例如：

```

```

```
nebula> ADD LISTENER ELASTICSEARCH 192.168.8.5:46780,192.168.8.6:46780;
```

查看listener

执行 `SHOW LISTENER` 语句可以列出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;
```

下一步

部署全文索引 [#4.deployment-and-installation/6.deploy-text-based-index/2.deploy-es/:]和listener后，全文索引会在Elasticsearch集群中自动创建，您可以进行全文搜索。详情请参见全文搜索 [#3.ngql-guide/15.full-text-index-statements/1.search-with-text-based-index/:]。

3.14.5 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` 或 `MATCH` 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 [#4.deployment-and-installation/6.deploy-text-based-index/2.deploy-es/:]和部署listener [#4.deployment-and-installation/6.deploy-text-based-index/3.deploy-listener/:]。

注意事项

使用全文索引前，请确认您已经了解全文索引的使用限制 [#4.deployment-and-installation/6.deploy-text-based-index/1.text-based-index-restrictions/:]。

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。

语法

```
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 [https://www.elastic.co/guide/en/elasticsearch/reference/6.8/common-options.html#fuzziness]。
  - `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` 。

示例

```
nebula> CREATE SPACE nba (partition_num=3,replica_factor=1, vid_type=fixed_string(30));
nebula> SIGN IN TEXT SERVICE (127.0.0.1:9200);
nebula> USE nba;
nebula> ADD LISTENER ELASTICSEARCH 192.168.8.5:46780;
nebula> CREATE TAG player(name string, age int);
nebula> CREATE TAG INDEX name ON player(name(20));
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);


nebula> LOOKUP ON player WHERE PREFIX(player.name, "B");
+-----------------+
| _vid            |
+-----------------+
| "Boris Diaw"    |
+-----------------+
| "Ben Simmons"   |
+-----------------+
| "Blake Griffin" |
+-----------------+

nebula> LOOKUP ON player WHERE WILDCARD(player.name, "*ri*") YIELD player.name, player.age;
+-----------------+-----------------+-----+
| _vid            | name            | age |
+-----------------+-----------------+-----+
| "Chris Paul"    | "Chris Paul"    | 33  |
+-----------------+-----------------+-----+
| "Boris Diaw"    | "Boris Diaw"    | 36  |
+-----------------+-----------------+-----+
| "Blake Griffin" | "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;
+--------------------+--------------------+-----+
| _vid               | name               | age |
+--------------------+--------------------+-----+
| "Russell Westbrook" | "Russell Westbrook" | 30  |
+--------------------+--------------------+-----+

nebula> LOOKUP ON player WHERE REGEXP(player.name, ".*");
+--------------------+
| _vid               |
+--------------------+
| "Danny Green"      |
+--------------------+
| "David West"       |
```

```
+--------------------+
| "Russell Westbrook" |
+--------------------+
...

nebula> LOOKUP ON player WHERE FUZZY(player.name, "Tim Dunncan", AUTO, OR) YIELD player.name;
+--------------+--------------+
| _vid         | name         |
+--------------+--------------+
| "Tim Duncan" | "Tim Duncan" |
+--------------+--------------+
```

3.15 子图和路径

3.15.1 GET SUBGRAPH

`GET SUBGRAPH` 语句检索指定边类型的起始点可以到达的点和边的信息，返回子图信息。

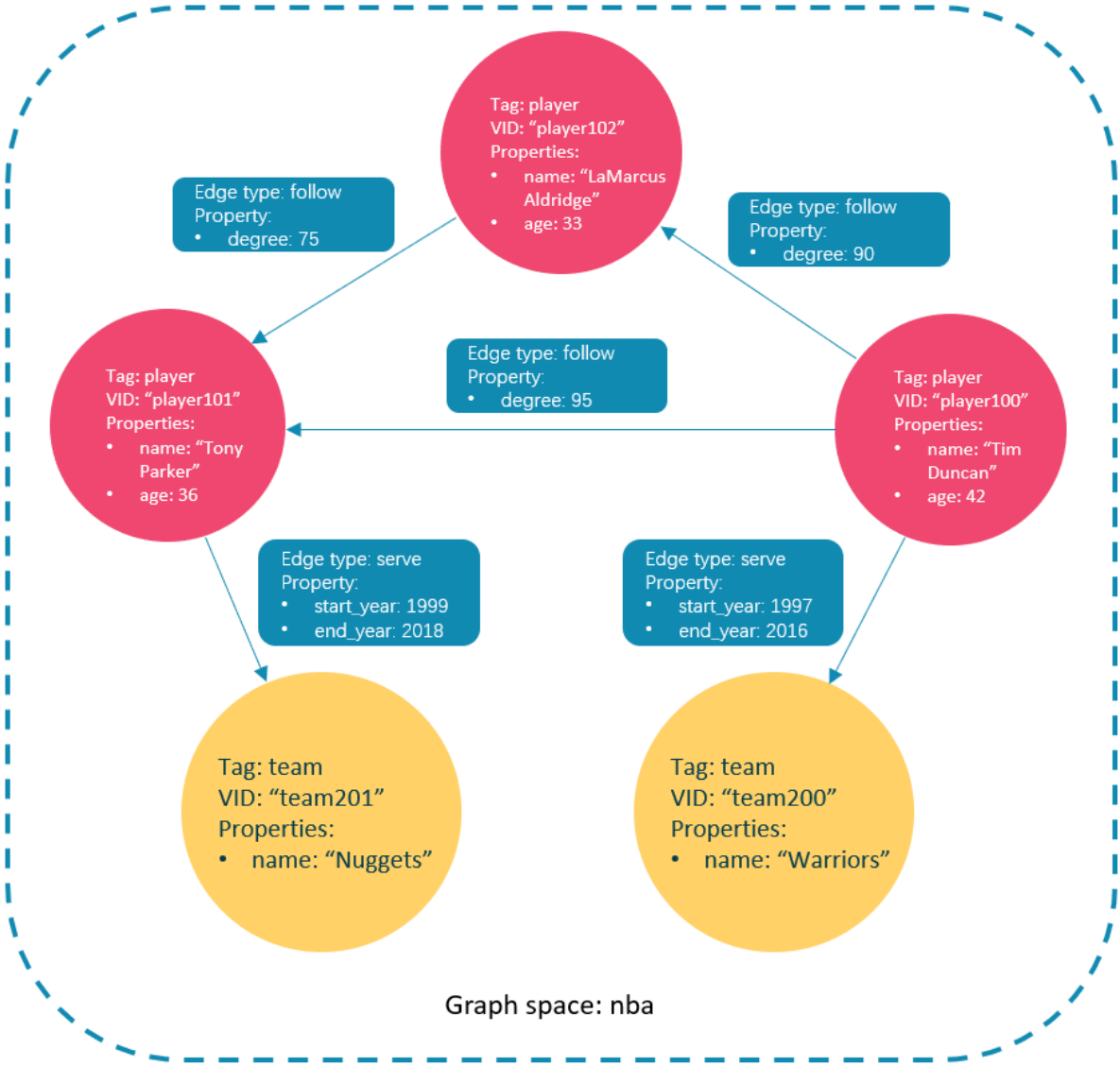
语法

```
GET SUBGRAPH [<step_count> STEPS] FROM {<vid>, <vid>...}
[IN <edge_type>, <edge_type>...]
[OUT <edge_type>, <edge_type>...]
[BOTH <edge_type>, <edge_type>...];
```

- `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` ：指定边类型。可以用 `IN` 、 `OUT` 和 `BOTH` 来指定起始点上该边类型的方向。默认为 `BOTH` 。

示例

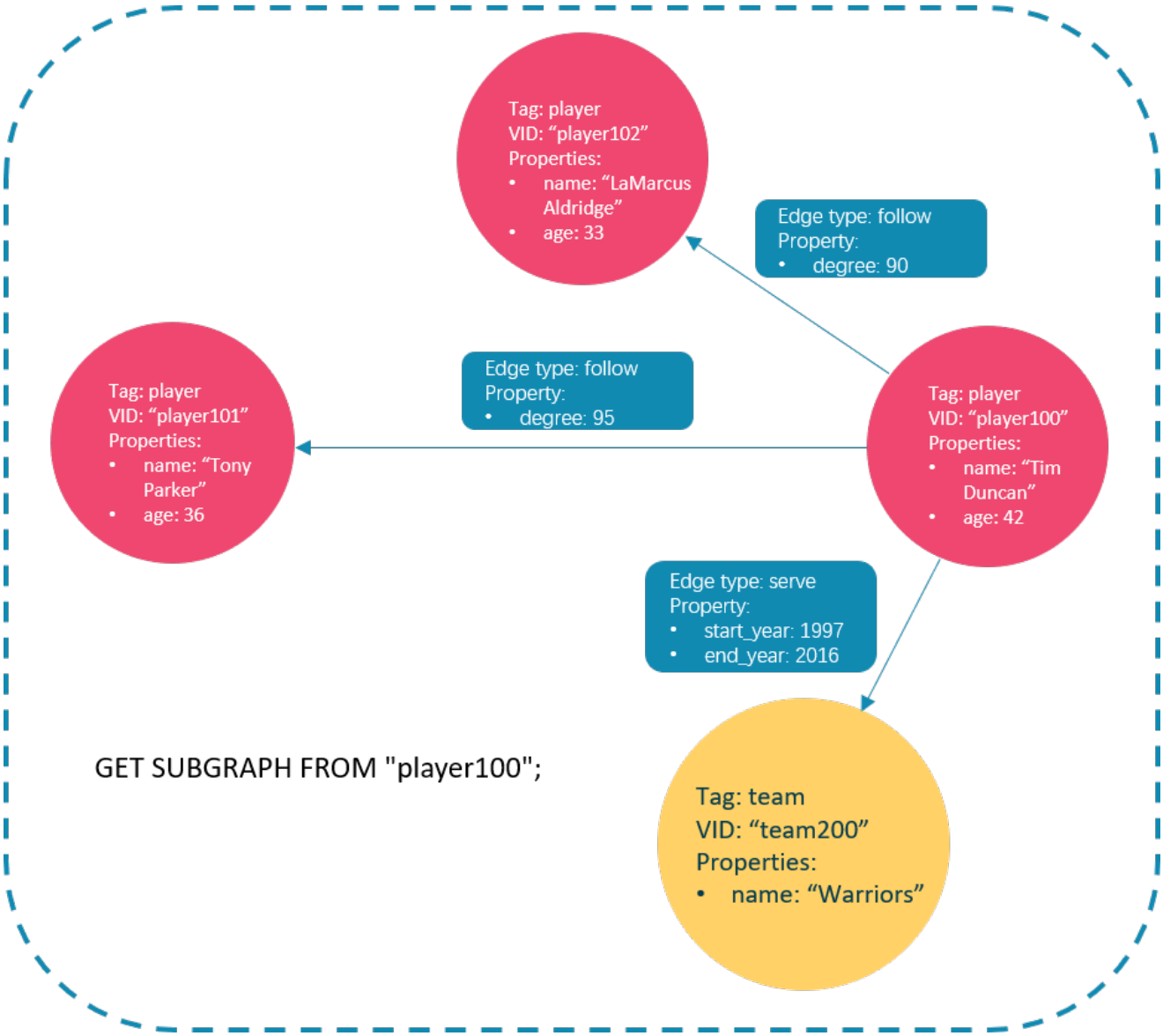
以下面的示例图进行演示。



- 查询从点 player100 开始、0~1跳、所有边类型的子图。

```
nebula> GET SUBGRAPH 1 STEPS FROM "player100";
+-------------------------------------------------------------------------------------------------------------------------------------------
+-------------------------------------------------------------------------------------------------------------------------------------------
+
| _vertices
|
_edges
|
+-------------------------------------------------------------------------------------------------------------------------------------------
+-------------------------------------------------------------------------------------------------------------------------------------------
+
| [(player100) player.name:Tim,player.age:42]
| [player100-[follow]->player101@0 degree:96,player100-[follow]->player102@0 degree:90,player100-[serve]->team200@0
end_year:2016,start_year:1997] |
+-------------------------------------------------------------------------------------------------------------------------------------------
+-------------------------------------------------------------------------------------------------------------------------------------------
+
| [(player101) player.age:36,player.name:Tony Parker,(player102) player.age:33,player.name:LaMarcus Aldridge,(team200)
team.name:Warriors] | [player102-[follow]->player101@0 degree:
75]                                                                                                  |
+-------------------------------------------------------------------------------------------------------------------------------------------
+-------------------------------------------------------------------------------------------------------------------------------------------
+
```

返回的子图如下。



- 查询从点 `player100` 开始、0~1跳、 `follow` 类型的入边的子图。

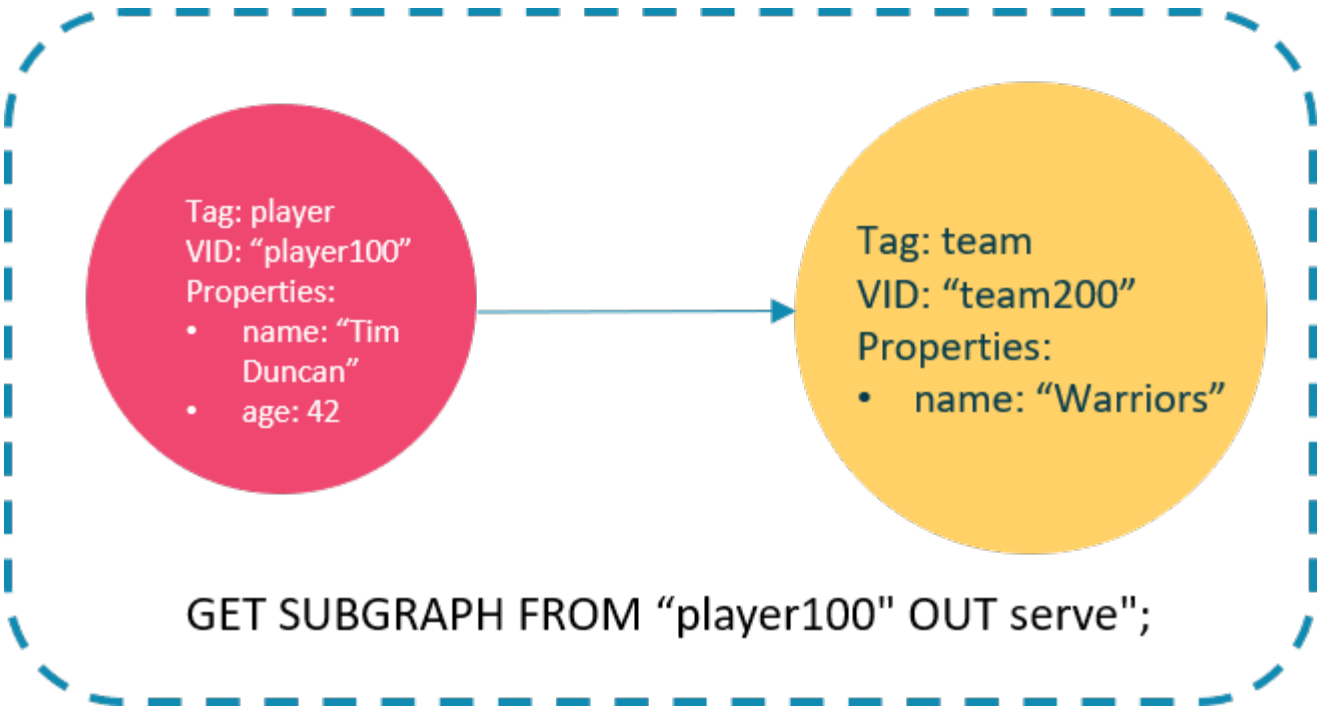
```
nebula> GET SUBGRAPH 1 STEPS FROM "player100" IN follow;
+-----------+--------+
| _vertices | _edges |
+-----------+--------+
| []        | []     |
+-----------+--------+
| []        | []     |
+-----------+--------+
```

因为 `player100` 没有 `follow` 类型的入边。所以返回的子图为空。

- 查询从点 player100 开始、0~1跳、 serve 类型的出边的子图。

```
nebula> GET SUBGRAPH 1 STEPS FROM "player100" OUT serve;
+------------------------------------------+-------------------------------------------------------------------+
| _vertices                                | _edges                                                            |
+------------------------------------------+-------------------------------------------------------------------+
| [(player100) player.age:42,player.name:Tim] | [player100-[serve]->team200@0 start_year:1997,end_year:2016] |
+------------------------------------------+-------------------------------------------------------------------+
| [(team200) team.name:Warriors]           | []                                                                |
+------------------------------------------+-------------------------------------------------------------------+
```

返回的子图如下。



3.15.2 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 FROM <vertex_id_list> TO <vertex_id_list>
OVER <edge_type_list> [REVERSELY | BIDIRECT] [UPTO <N> STEPS] [| 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。
- `<vertex_id_list>` ：点ID列表.多个点用英文逗号（,）分隔。支持 `$-` 和 `$var` 。
- `<edge_type_list>` ：边类型列表.多个边类型用英文逗号（,）分隔。 `*` 表示所有边类型。
- `REVERSELY | BIDIRECT` ： `REVERSELY` 表示反向， `BIDIRECT` 表示双向。
- `<N>` ：路径的最大跳数。默认值为 `5` 。
- `<M>` ：指定返回的最大行数。

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。
- 不支持过滤属性。
- 路径的查找是单线程，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 "team201" OVER *;
+------------------------------------------------------------------+
| path                                                             |
+------------------------------------------------------------------+
| ("player102")-[:follow@0]->("player101")-[:serve@0]->("team201") |
+------------------------------------------------------------------+
```

```
nebula> FIND SHORTEST PATH FROM "team200" TO "player100" OVER * REVERSELY;
+--------------------------------------+
| path                                 |
+--------------------------------------+
```

```
| ("team200")<-[:serve@0]-("player100") |
+---------------------------------------+
```

```
nebula> FIND ALL PATH FROM "player100" TO "team200" OVER *;
+---------------------------------------+
| path                                  |
+---------------------------------------+
| ("player100")-[:serve@0]->("team200") |
+---------------------------------------+
```

```
nebula> FIND NOLOOP PATH FROM "player100" TO "team200" OVER *;
+---------------------------------------+
| path                                  |
+---------------------------------------+
| ("player100")-[:serve@0]->("team200") |
+---------------------------------------+
```

FAQ

是否支持WHERE子句，以实现图遍历过程中的条件过滤？

不支持WHERE子句，因此不能在遍历过程中进行条件过滤。

3.16 查询调优

3.16.1 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。您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` 。您可以使用 `format` 选项修改输出格式。

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。

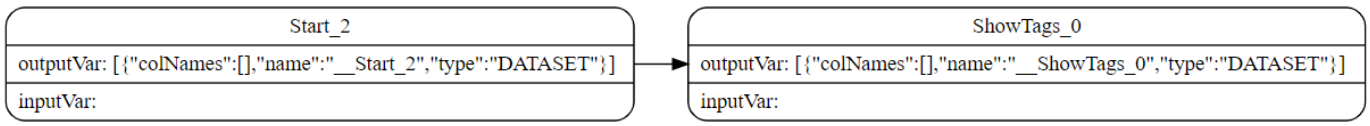
dot格式

dot 格式将返回DOT语言的信息，然后您可以使用Graphviz生成计划图。

> **说明**：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 [https://dreampuf.github.io/GraphvizOnline/]。

```
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];
  }
-------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------
```

将上述示例的DOT语言转换为Graphviz图，如下所示。



3.17 运维

3.17.1 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 [#8.service-tuning/load-balance/:]。

`BALANCE` 语法说明如下。

语法	说明
`BALANCE DATA`	启动任务均衡分布Nebula Graph集群里的所有分片。该命令会返回任务ID（ `balance_id` ）。
`BALANCE DATA <balance_id>`	显示 `BALANCE DATA` 任务的状态。
`BALANCE DATA STOP`	停止 `BALANCE DATA` 任务。
`BALANCE DATA REMOVE <host_list>`	在Nebula Graph集群中扫描并解绑指定的Storage主机。
`BALANCE LEADER`	在Nebula Graph集群中均衡分布leader。

3.17.2 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。作业管理可以帮助您执行、查看、停止和恢复作业。

SUBMIT JOB COMPACT

`SUBMIT JOB COMPACT`语句 会触发RocksDB的长耗时 `compact` 操作。

`compact` 配置详情请参见Storage服务配置 [#5.configurations-and-logs/1.configurations/4.storage-config/:]。

示例

```
nebula> SUBMIT JOB COMPACT;
+------------+
| New Job Id |
+------------+
| 40         |
+------------+
```

SUBMIT JOB FLUSH

`SUBMIT JOB FLUSH` 语句将内存中的RocksDB memfile写入硬盘。

示例

```
nebula> SUBMIT JOB FLUSH;
+------------+
| New Job Id |
+------------+
| 96         |
+------------+
```

SUBMIT JOB STATS

`SUBMIT JOB STATS` 语句启动一个作业，该作业对当前图空间进行统计。作业完成后，您可以使用 `SHOW STATS` 语句列出统计结果。详情请参见SHOW STATS [#3.ngql-guide/7.general-query-statements/6.show/14.show-stats/:]。

> **说明**：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 |
+------------+
| 97         |
+------------+
```

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 96;
+----------------+---------------+------------+-------------------------+-------------------------+
| Job Id(TaskId) | Command(Dest) | Status     | Start Time              | Stop Time               |
+----------------+---------------+------------+-------------------------+-------------------------+
| 96             | "FLUSH"       | "FINISHED" | 2020-11-28T14:14:29.000 | 2020-11-28T14:14:29.000 |
+----------------+---------------+------------+-------------------------+-------------------------+
| 0              | "storaged2"   | "FINISHED" | 2020-11-28T14:14:29.000 | 2020-11-28T14:14:29.000 |
+----------------+---------------+------------+-------------------------+-------------------------+
| 1              | "storaged0"   | "FINISHED" | 2020-11-28T14:14:29.000 | 2020-11-28T14:14:29.000 |
+----------------+---------------+------------+-------------------------+-------------------------+
| 2              | "storaged1"   | "FINISHED" | 2020-11-28T14:14:29.000 | 2020-11-28T14:14:29.000 |
+----------------+---------------+------------+-------------------------+-------------------------+
```

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 [#3.ngql-guide/18.operation-and-maintenance-statements/4.job-statements/:_6]。
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` 。

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 -/
```

SHOW JOBS

`SHOW JOBS` 语句列出所有未过期的作业。

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[#5.configurations-and-logs/1.configurations/2.meta-config/:]。

示例

```
nebula> SHOW JOBS;
+--------+----------------------+------------+-------------------------+-------------------------+
| Job Id | Command              | Status     | Start Time              | Stop Time               |
+--------+----------------------+------------+-------------------------+-------------------------+
| 97     | "STATS"              | "FINISHED" | 2020-11-28T14:48:52.000 | 2020-11-28T14:48:52.000 |
+--------+----------------------+------------+-------------------------+-------------------------+
| 96     | "FLUSH"              | "FINISHED" | 2020-11-28T14:14:29.000 | 2020-11-28T14:14:29.000 |
+--------+----------------------+------------+-------------------------+-------------------------+
| 95     | "STATS"              | "FINISHED" | 2020-11-28T13:02:11.000 | 2020-11-28T13:02:11.000 |
+--------+----------------------+------------+-------------------------+-------------------------+
| 86     | "REBUILD_EDGE_INDEX" | "FINISHED" | 2020-11-26T13:38:24.000 | 2020-11-26T13:38:24.000 |
+--------+----------------------+------------+-------------------------+-------------------------+
```

STOP JOB

`STOP JOB` 语句可以停止未完成的作业。

示例

```
nebula> STOP JOB 22;
+---------------+
| Result        |
+---------------+
| "Job stopped" |
+---------------+
```

RECOVER JOB

`RECOVER JOB` 语句会重新执行失败的作业，并返回已恢复的作业数量。

示例

```
nebula> RECOVER JOB;
+-------------------+
| Recovered job num |
+-------------------+
| 5 job recovered   |
+-------------------+
```

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。您可以执行如下命令调试：

```
curl "http://{storaged-ip}:19779/admin?space={space_name}&op=compact"
```

3.18 附录

3.18.1 注释

本文介绍nGQL中的注释方式。

历史版本兼容性

- Nebula Graph 1.0支持四种注释方式：`#`、`--`、`//`、`/* */`。
- Nebula Graph 2.0中，`--`不再是注释符，而是代表边模式 [#3.ngql-guide/1.nGQL-overview/3.graph-patterns/:]。

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，您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;
```

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

3.18.2 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 my_space;
nebula> use MY_SPACE;
[ERROR (-8)]: SpaceNotFound:
```

关键字和保留字不区分大小写

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;
```

3.18.3 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。

非保留关键字作为标识符时可以不使用引号。保留关键字作为标识符时，需要用反引号（`）将它们括起来，例如`AND`。

> **说明**：关键字不区分大小写。

```
nebula> CREATE TAG TAG(name string);
[ERROR (-7)]: SyntaxError: syntax error near `TAG'

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

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

- `TAG` 是保留关键字，要将 `TAG` 作为标识符，您必须使用反引号（`）括起来。
- `SPACE` 是非保留关键字，可以直接作为标识符使用。

保留关键字

```
ADD
ALTER
AND
AS
ASC
BALANCE
BOOL
BY
CASE
CHANGE
COMPACT
CREATE
DATE
DATETIME
DELETE
DESC
DESCRIBE
DISTINCT
DOUBLE
DOWNLOAD
DROP
EDGE
EDGES
EXISTS
EXPLAIN
FETCH
FIND
FIXED_STRING
FLOAT
FLUSH
FORMAT
FROM
GET
GO
GRANT
```

```
IF
IN
INDEX
INDEXES
INGEST
INSERT
INT
INT16
INT32
INT64
INT8
INTERSECT
IS
LIMIT
LOOKUP
MATCH
MINUS
NO
NOT
NULL
OF
OFFSET
ON
OR
ORDER
OVER
OVERWRITE
PROFILE
PROP
REBUILD
RECOVER
REMOVE
RETURN
REVERSELY
REVOKE
SET
SHOW
STEP
STEPS
STOP
STRING
SUBMIT
TAG
TAGS
TIME
TIMESTAMP
TO
UNION
UPDATE
UPSERT
UPTO
USE
VERTEX
WHEN
WHERE
WITH
XOR
YIELD
```

非保留关键字

```
```

```
ACCOUNT
ADMIN
ALL
ANY
ATOMIC_EDGE
AUTO
AVG
BIDIRECT
BIT_AND
BIT_OR
BIT_XOR
BOTH
CHARSET
CLIENTS
COLLATE
COLLATION
COLLECT
COLLECT_SET
CONFIGS
CONTAINS
COUNT
COUNT_DISTINCT
DATA
DBA
DEFAULT
ELASTICSEARCH
ELSE
END
ENDS
FALSE
FORCE
FUZZY
GOD
GRAPH
GROUP
GROUPS
GUEST
HDFS
HOST
HOSTS
INTO
JOB
JOBS
LEADER
LISTENER
MAX
META
MIN
NOLOOP
NONE
OPTIONAL
OUT
PART
PARTITION_NUM
PARTS
PASSWORD
PATH
PLAN
PREFIX
REGEXP
REPLICA_FACTOR
RESET
ROLE
```

```
ROLES
SEARCH
SERVICE
SHORTEST
SIGN
SINGLE
SKIP
SNAPSHOT
SNAPSHOTS
SPACE
SPACES
STARTS
STATS
STATUS
STD
STORAGE
SUBGRAPH
SUM
TEXT
TEXT_SEARCH
THEN
TRUE
TTL_COL
TTL_DURATION
UNWIND
USER
USERS
UUID
VALUE
VALUES
VID_TYPE
WILDCARD
ZONE
ZONES
```

3.18.4 点ID和分片ID

点ID（VID）

在Nebula Graph中，插入点时必须用点ID（`VID`）标识这个点。`VID`有如下特点：

- `VID`数据类型可以为定长字符串或64位整数。
- `VID`在图空间中必须唯一。
- 一个`VID`可以有多个标签（`TAG`）。例如一个人（`VID`）可以有两个角色（`TAG`）。
- 不同图空间中的`VID`是完全独立的。

插入点请参见INSERT VERTEX [#3.ngql-guide/12.vertex-statements/1.insert-vertex/:]。

分片ID

点和边分布在不同的分片，分片分布在不同的机器。如果您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 [https://github.com/vesoft-inc/nebula-common/blob/master/src/common/clients/meta/MetaClient.cpp]。

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 [#3.ngql-guide/9.space-statements/1.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，所以您不能假定任何两个分片位于同一台机器上。

4. 安装部署

4.1 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。

4.1.1 阅读指南

如果您是带着如下问题阅读本文，可以直接单击问题跳转查看对应的说明。

- 编译Nebula Graph源码的要求是什么？[#4.deployment-and-installation/1.resource-preparations/:nebula_graph_1]
- 测试环境中运行Nebula Graph的要求是什么？[#4.deployment-and-installation/1.resource-preparations/:nebula_graph_2]
- 生产环境中运行Nebula Graph的要求是什么？[#4.deployment-and-installation/1.resource-preparations/:nebula_graph_3]
- 需要预留多少内存和硬盘空间给Nebula Graph集群？[#4.deployment-and-installation/1.resource-preparations/:nebula_graph_4]

4.1.2 编译Nebula Graph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，建议您使用内核版本为 `2.6.32` 及以上版本的Linux系统。

软件要求

您的软件版本需要如下表所示，如果它们不符合要求，或者您也不确定它们的版本，请按照安装编译所需软件[#4.deployment-and-installation/1.resource-preparations/:_5]中的步骤进行操作。

软件名称	版本	备注
glibc	2.12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	-
zlid-devel	任意稳定版本	-
gcc	7.1.0及以上	执行命令 `gcc -v` 检查版本。
gcc-c++	任意稳定版本	-
cmake	3.5.0及以上	执行命令 `cmake --version` 检查版本。
gettext	任意稳定版本	-
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。

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。

安装编译所需软件

本小节指导您下载和安装Nebula Graph编译时需要的软件。

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 \
                    gettext \
                    curl \
                    redhat-lsb-core
   // 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 \
                        gettext
   ```

2. 检查主机上的GCC和CMake版本是否正确。版本信息请参见软件要求 [#4.deployment-and-installation/1.resource-preparations/:_4]。

   ```
   $ g++ --version
   $ cmake --version
   ```

   如果版本正确，您可以跳过本小节。如果不正确，请根据如下步骤安装：

   1. 克隆仓库 `nebula-common` 到主机。

      ```
      ```bash
      $ git clone https://github.com/vesoft-inc/nebula-common.git
      ```

      类似2.0.0版本的Nebula Graph源码存储在特定的分支，您可以使用`--branch`或`-b`选项指定克隆的分支。例如指定2.0.0，命令如下：

      ```bash
      $ git clone --branch v2.0.0 https://github.com/vesoft-inc/nebula-common.git
      ```
      ```

2. 进入目录 `nebula-common` 。

```
```bash
$ cd nebula-common
```
```

3. 执行如下命令安装和启用GCC和CMake。

```
```bash
// 安装CMake。
$ ./third-party/install-cmake.sh cmake-install

// 启用CMake。
$ source cmake-install/bin/enable-cmake.sh

// 安装GCC。安装到opt目录需要root权限，您也可以修改为其他目录。
$ sudo ./third-party/install-gcc.sh --prefix=/opt

// 启用GCC。
$ source /opt/vesoft/toolset/gcc/7.5.0/enable
```

4.1.3 测试环境运行Nebula Graph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，建议您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，您可以在机器上部署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，您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

4.1.4 生产环境运行Nebula Graph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，建议您在生产环境中使用内核版本为`3.9`及以上版本的Linux系统。

您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 [#5.configurations-and-logs/1.configurations/6.kernel-config/:]。

服务架构建议

进程	建议数量
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

通常您只需要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。

您可以在一台机器上部署多个进程，例如五台机器构成的集群，您可以按照如下方案部署Nebula Graph。

> **警告**：请**不要**跨机房部署集群。

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

4.1.5 Nebula Graph资源要求

您可以预估一个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%	-
内存	Bytes	[`点和边的总数` * 15 + `RocksDB实例数量` * (`write_buffer_size` * `max_write_buffer_number` + `块缓存大小`] * 120%	`write_buffer_size` 和 `max_write_buffer_number` 是RocksDB内存相关参数，详情请参见MemTable [https://github.com/facebook/rocksdb/wiki/MemTable]。块缓存大小请参见Memory usage in RocksDB [https://github.com/facebook/rocksdb/wiki/Memory-usage-in-RocksDB#block-cache]。
分区数量	-	`集群硬盘数量` * `disk_partition_num_multiplier`	`disk_partition_num_multiplier` 是取值为2~10的一个整数，用于衡量硬盘性能。HDD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。

  > **说明**：您可以在配置文件 `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。

4.1.6 关于存储设备

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。

不推荐使用HDD，因为HDD的IOPS性能差，随机寻道延迟高，可能还会遇到许多其他问题。也不建议使用远端存储设备，例如NAS或SAN。

请使用本地SSD设备。

4.2 编译安装Nebula Graph

4.2.1 使用源码安装Nebula Graph

使用源码安装Nebula Graph允许您自定义编译和安装设置，并测试最新特性。

前提条件

- 已准备正确的编译环境。详情请参见准备编译、安装和运行Nebula Graph的环境 [#4.deployment-and-installation/1.resource-preparations/:]。
- 待安装Nebula Graph的主机可以访问互联网。

安装步骤

1. 克隆Nebula Graph源码到主机。

   - 如果需要安装最新版本的开发代码，请执行如下命令下载 `master` 分支的源码：

   ```
   ```bash
   $ git clone https://github.com/vesoft-inc/nebula-graph.git
   ```
   ```

   - 如果需要安装指定版本的Nebula Graph 2.x，请使用选项 `--branch` 指定分支。例如安装2.0.0 发布版本，请执行如下命令：

   ```
   ```bash
   $ git clone --branch v2.0.0 https://github.com/vesoft-inc/nebula-graph.git
   ```
   ```

2. 进入目录 `nebula-graph` 。

   ```
   $ cd nebula-graph
   ```

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。

   > **说明**：
   >
   > - 默认安装路径为 `/user/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 [#4.deployment-and-installation/2.compile-and-install-nebula-graph/1.install-nebula-graph-by-compiling-the-source-code/:cmake]。

   - 如果您在第1步下载的是最新版本的开发代码，请执行如下命令：

   ```
   ```

```bash
$ cmake -DENABLE_BUILD_STORAGE=on -DENABLE_TESTING=OFF -DCMAKE_BUILD_TYPE=Release ..
```

- 如果您在第1步下载的是指定版本的Nebula Graph 2.x，请使用选项 `-DNEBULA_COMMON_REPO_TAG` 和 `-DNEBULA_STORAGE_REPO_TAG` 指定相同的nebula-common [https://github.com/vesoft-inc/nebula-common]和nebula-storage [https://github.com/vesoft-inc/nebula-storage]分支。例如安装2.0.0 GA版本，请执行如下命令：

```bash
$ cmake -DENABLE_BUILD_STORAGE=on -DENABLE_TESTING=OFF -DCMAKE_BUILD_TYPE=Release \
-DNEBULA_COMMON_REPO_TAG=v2.0.0 -DNEBULA_STORAGE_REPO_TAG=v2.0.0 ..
```

5. 编译Nebula Graph。

   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$min(CPU核数,\frac{内存(GB)}{2})$。

   ```
   $ make -j{N}
   ```

6. 安装Nebula Graph。

   ```
   $ sudo make install-all
   ```

6. （可选）更新 `master` 分支的源码，因为它经常更新。

   1. 在目录 `nebula-graph/` 中，执行命令 `git pull upstream master` 更新源码。
   2. 在目录 `nebula-graph/modules/common/` 和 `nebula-graph/modules/storage/` 中，分别执行命令 `git pull upstream master` 。
   3. 在目录 `nebula-graph/build/` 中，重新执行 `make -j{N}` 和 `make install-all` 。

下一步

- 管理Nebula Graph服务 [#2.quick-start/5.start-stop-service/:]
- 连接Nebula Graph [#2.quick-start/3.connect-to-nebula-graph/:]
- 基础操作语法 [#2.quick-start/4.nebula-graph-crud/:]

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。

ENABLE_BUILD_STORAGE

从2.0.0版本开始，Nebula Graph的graph和storage代码仓库分离，可以各自单独编译。 `ENABLE_BUILD_STORAGE` 默认值为 `OFF` ，表示Storage服务不会和Graph服务一起安装。

如果您单机部署Nebula Graph进行测试，可以设置 `ENABLE_BUILD_STORAGE=ON` ，安装时会自动下载和安装Storage服务。

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

`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，您可以设置为 `OFF` 。

ENABLE_TESTING

`ENABLE_TESTING` 默认值为 `ON` ，表示单元测试服务由Nebula Graph服务构建。如果您只需要服务模块，可以设置为 `OFF` 。

ENABLE_ASAN

`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` 。

MAKE_BUILD_TYPE

`MAKE_BUILD_TYPE` 控制Nebula Graph的build方法，取值说明如下：

- `Debug`

  `MAKE_BUILD_TYPE` 的默认值，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。

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，或者您想使用其他编译器，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，因此您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 [https://ccache.dev/manual/3.7.6.html]。

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` 。

4.2.2 使用RPM或DEB安装包安装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安装包快速安装Nebula Graph。

前提条件

准备合适资源 [#4.deployment-and-installation/1.resource-preparations/:]

下载安装包

阿里云OSS下载

- 下载release版本

  URL格式如下：

```
//Centos 6
https://oss-cdn.nebula-graph.com.cn/package/<release_version>/nebula-graph-<release_version>.el6.x86_64.rpm

//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 `2.0.0` 安装包：

```
wget https://oss-cdn.nebula-graph.com.cn/package/2.0.0/nebula-graph-2.0.0.el7.x86_64.rpm
wget https://oss-cdn.nebula-graph.com.cn/package/2.0.0/nebula-graph-2.0.0.el7.x86_64.rpm.sha256sum.txt
```

  下载适用于 `ubuntu 1804` 的 `2.0.0` 安装包：

```
wget https://oss-cdn.nebula-graph.com.cn/package/2.0.0/nebula-graph-2.0.0.ubuntu1804.amd64.deb
wget https://oss-cdn.nebula-graph.com.cn/package/2.0.0/nebula-graph-2.0.0.ubuntu1804.amd64.deb.sha256sum.txt
```

- 下载日常开发版本(nightly)

  > **禁止**：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。

  URL格式如下：

```
//Centos 6
https://oss-cdn.nebula-graph.com.cn/package/v2-nightly/<yyyy.mm.dd>/nebula-graph-<yyyy.mm.dd>-nightly.el6.x86_64.rpm

//Centos 7
https://oss-cdn.nebula-graph.com.cn/package/v2-nightly/<yyyy.mm.dd>/nebula-graph-<yyyy.mm.dd>-nightly.el7.x86_64.rpm

//Centos 8
https://oss-cdn.nebula-graph.com.cn/package/v2-nightly/<yyyy.mm.dd>/nebula-graph-<yyyy.mm.dd>-nightly.el8.x86_64.rpm

//Ubuntu 1604
https://oss-cdn.nebula-graph.com.cn/package/v2-nightly/<yyyy.mm.dd>/nebula-graph-<yyyy.mm.dd>-nightly.ubuntu1604.amd64.deb

//Ubuntu 1804
https://oss-cdn.nebula-graph.com.cn/package/v2-nightly/<yyyy.mm.dd>/nebula-graph-<yyyy.mm.dd>-nightly.ubuntu1804.amd64.deb

//Ubuntu 2004
https://oss-cdn.nebula-graph.com.cn/package/v2-nightly/<yyyy.mm.dd>/nebula-graph-<yyyy.mm.dd>-nightly.ubuntu1804.amd64.deb
```

  例如要下载 `2021.03.28` 适用于 `Centos 7.5` 的 `2.x` 安装包：

```
wget https://oss-cdn.nebula-graph.com.cn/package/v2-nightly/2021.03.28/nebula-graph-2021.03.28-nightly.el7.x86_64.rpm
wget https://oss-cdn.nebula-graph.com.cn/package/v2-nightly/2021.03.28/nebula-graph-2021.03.28-
nightly.el7.x86_64.rpm.sha256sum.txt
```

  要下载 `2021.03.28` 适用于 `Ubuntu 1804` 的 `2.x` 安装包：

```
wget https://oss-cdn.nebula-graph.com.cn/package/v2-nightly/2021.03.28/nebula-graph-2021.03.28-nightly.ubuntu1804.amd64.deb
wget https://oss-cdn.nebula-graph.com.cn/package/v2-nightly/2021.03.28/nebula-graph-2021.03.28-
nightly.ubuntu1804.amd64.deb.sha256sum.txt
```

GITHUB下载

- 下载release版本

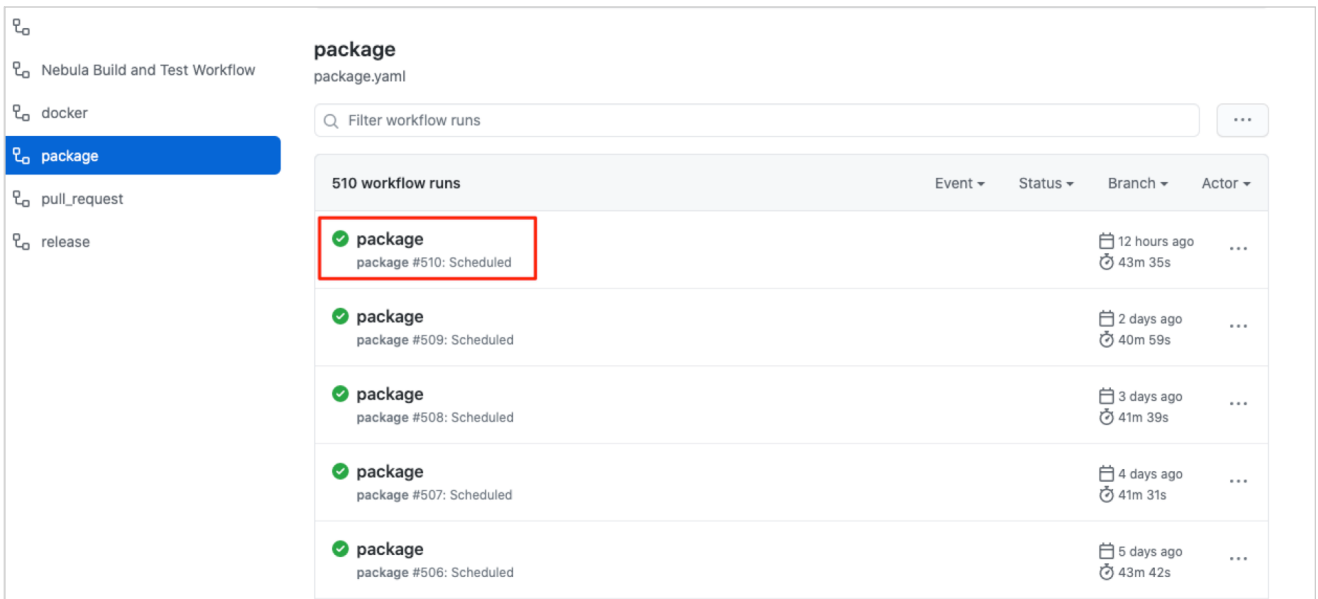
  + 登录Nebula Graph Releases [https://github.com/vesoft-inc/nebula-graph/releases]页面，确认需要的版本，单击**Assets**。

+ 在**Assets**区域找到机器运行所需的安装包，下载文件到机器上。

- 下载nightly版本

> 禁止：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。

+ 登录Nebula Graph package [https://github.com/vesoft-inc/nebula/actions/workflows/package.yaml]页面，单击顶部最新的**package**。



+ 在**Artifacts**区域找到机器运行所需的安装包，下载文件到机器上。

安装Nebula Graph

- 安装RPM包

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

- 安装DEB包

```
$ sudo dpkg -i --instdir==<installation_path> <package_name>
```

> **说明**：如果不设置安装路径，默认安装路径为 `/usr/local/nebula/` 。

4.3 部署Nebula Graph集群

Nebula Graph暂不提供官方的集群部署工具，需要您手动部署，下文将介绍手动部署的简单流程。

4.3.1 前提条件

准备用于部署集群的服务器。

4.3.2 手动部署流程

安装Nebula Graph

在集群的每一台服务器上安装Nebula Graph。安装方式请参见：

- 使用RPM或DEB安装包安装Nebula Graph [#4.deployment-and-installation/2.compile-and-install-nebula-graph/2.install-nebula-graph-by-rpm-or-deb/:]
- 编译安装Nebula Graph [#4.deployment-and-installation/2.compile-and-install-nebula-graph/1.install-nebula-graph-by-compiling-the-source-code/:]

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`，您可以只修改对应服务的配置文件。配置文件的详细说明请参见：

- Meta服务配置 [#5.configurations-and-logs/1.configurations/2.meta-config/:]
- Graph服务配置 [#5.configurations-and-logs/1.configurations/3.graph-config/:]
- Storage服务配置 [#5.configurations-and-logs/1.configurations/4.storage-config/:]

启动集群

启动各个服务器上的对应服务。启动Nebula Graph服务的命令如下：

```
sudo /usr/local/nebula/scripts/nebula.service start <metad|graphd|storaged|all>
```

`/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 [#2.quick-start/5.start-stop-service/:]。

检查集群

连接Graph服务，执行命令`SHOW HOSTS`检查集群状态。

4.4 升级 Nebula Graph 历史版本至 v2.0.0

4.4.1 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。
- **支持**如下版本的升级。
  - Nebula Graph v1.2.0 [https://github.com/vesoft-inc/nebula/releases/tag/v1.2.0] 升级至 Nebula Graph v2.0.0 [https://github.com/vesoft-inc/nebula-graph/releases/tag/v2.0.0].
  - Nebula Graph v2.0.0-RC1 [https://github.com/vesoft-inc/nebula-graph/releases/tag/v2.0.0-rc1] 升级至 Nebula Graph 2.0.0.
- **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"的三**倍**。
- 已知会造成数据丢失的4种场景，和 alter schema 以及 default value 相关，具体见 github known issues [https://github.com/vesoft-inc/nebula-graph/issues/857].
- 所有的客户端均需要升级，通信协议不兼容。
- 升级时间大约需要30分钟（取决于具体配置），参见文末测试环境。
- 数据目录不要使用软连接切换，避免失效。
- 升级操作需要有 sudo 权限。

4.4.2 前置条件说明

旧版本安装目录

默认情况下，旧版本安装的根目录为 `/usr/local/nebula/` (下文记为 `${nebula-old}` 。默认配置文件目录为 `${nebula-old}/etc/` .

- 在 `${nebula-old}/etc/nebula-storaged.conf` 中找到 `--data_path` 项，其默认值为 `data/storage` ; 为 **storaged 数据目录**位置。
- 在 `${nebula-old}/etc/nebula-metad.conf` 中找到 `--data_path` 项，其默认值为 `data/meta` ; 为 **metad 数据目录**位置。

> **说明**: Nebula Graph的实际安装路径可能和本文示意不同，请使用实际路径。您也可以用 `ps -ef | grep nebula` 中的参数来找到实际使用的配置文件地址。

新版本安装目录

本文中新版本安装目录记为 `${nebula-new}` (例如 `/usr/local/nebula-new/`).

```
> mkdir -p ${nebula-new}
```

4.4.3 升级步骤

1. **停止所有客户端访问**。也可以通过在每台服务器上关闭 graphd 服务避免脏写。在每台服务器上运行如下命令。

```
> ${nebula-old}/scripts/nebula.service stop graphd
[INFO] Stopping nebula-graphd...
[INFO] Done
```

1. 停止旧版本服务。在每台服务器上运行如下命令。

```
> ${nebula-old}/scripts/nebula.service stop all
[INFO] Stopping nebula-metad...
[INFO] Done
[INFO] Stopping nebula-graphd...
[INFO] Done
[INFO] Stopping nebula-storaged...
[INFO] Done
```

运行 `ps -ef | grep nebula` 检查所有 nebula 服务都已停止。 `storaged` 进程 flush 数据可能要等待 1 分钟。

> 如果超过 20 分钟不能停止服务，**放弃本次升级**，并在论坛提交问题。

1. 在每台服务器上运行如下命令。

   3.1 安装新的二进制文件

   - 如果从 rpm/deb 安装,从release page [https://github.com/vesoft-inc/nebula-graph/releases]下载对应操作系统的安装包。

```
> sudo rpm --force -i --prefix=${nebula-new}  ${nebula-package-name.rpm} # for centos/redhat
> sudo dpkg -i --instdir==${nebula-new} ${nebula-package-name.deb} # for ubuntu
```

具体步骤可以见从RPM/DEB安装 [https://docs.nebula-graph.io/2.0/4.deployment-and-installation/2.compile-and-install-nebula-graph/2.install-nebula-graph-by-rpm-or-deb/]。

   - 如果从源代码安装。具体步骤见从源代码安装 [https://docs.nebula-graph.io/2.0/4.deployment-and-installation/2.compile-and-install-nebula-graph/1.install-nebula-graph-by-compiling-the-source-code/]。这里列出几个关键命令：

     + clone 源代码

```
> git clone --branch v2.0.0 https://github.com/vesoft-inc/nebula-graph.git
```

     + 设置 CMake

```
> cmake -DCMAKE_INSTALL_PREFIX=${nebula-new} -DENABLE_BUILD_STORAGE=on -DENABLE_TESTING=OFF -DCMAKE_BUILD_TYPE=Release -
DNEBULA_COMMON_REPO_TAG=v2.0.0  -DNEBULA_STORAGE_REPO_TAG=v2.0.0 ..
```

   3.2 拷贝配置文件

```

```

```
> cp -rf ${nebula-old}/etc ${nebula-new}/
```

1. 在曾经运行 metad 的服务器上（通常为3台），拷贝 metad 数据、配置文件到新目录。

   - 拷贝 metad 数据

     在 `${nebula-old}/etc/nebula-metad.conf` 中找到 `--data_path` 项（其默认值为 `data/meta`）

     + 如果旧版本配置**未更改** `--data_path` 项，则可以运行如下命令，将metad数据拷贝到新目录。

     ```
     > mkdir -p ${nebula-new}/data/meta/
     > cp -r ${nebula-old}/data/meta/* ${nebula-new}/data/meta/
     ```

     + 如果旧版本配置更改了默认的 metad 目录，请根据实际目录拷贝。

   - 拷贝并修改配置文件

     + 编辑新的 metad 配置文件：

     ```
       ```bash
       > vim ${nebula-new}/nebula-metad.conf
       ```

       * [可选]增加配置项：

       `--null_type=false`: 升级后的schema的属性是否要支持 [`NULL`](https://docs.nebula-graph.io/2.0/3.ngql-guide/3.data-types/5.null/)，**默认为 true**。不希望支持 NULL 的话，设置为 false。此时，升级后的 schema 如果要增加属性（alter tag/edge)必须指定 [default 值](https://docs.nebula-graph.io/2.0/3.ngql-guide/10.tag-statements/1.create-tag/)，否则会读不出数据。

        `--string_index_limit=32`: 升级后 string 对应的[索引的长度](https://docs.nebula-graph.io/2.0/3.ngql-guide/14.native-index-statements/1.create-native-index/)，不加的话系统默认为 64。
     ```

> **检查**：每个 metad 都完成操作。

1. 在每个 storaged 服务器上，修改 storaged 配置文件。

   + [可选]如果旧版本 storaged 数据目录 `--data_path=data/storage` 不是默认值，有更改。

   ```
   > vim ${nebula-new}/nebula-storaged.conf
   ```

   `--data_path` 设置为新的 storaged 数据目录地址。

   + 创建新版本 storaged 数据目录。

   ```
   > mkdir -p ${nebula-new}/data/storage/
   ```

   如果 `${nebula-new}/etc/nebula-storaged.conf` 中的 `--data_path` 有改动，请按实际路径创建。

2. 启动新版本的 metad 进程。

   - 在每个 metad 的服务器上运行如下命令。

   ```
   $ sudo ${nebula-new}/scripts/nebula.service start metad
   [INFO] Starting nebula-metad...
   [INFO] Done
   ```

- 检查每个 metad 进程是否正常。

```
$ ps -ef |grep nebula-metad
```

- 检查 metad 日志 `${nebula-new}/logs/` 下的 ERROR 日志。

> 如果服务异常：请查看目录 `${nebula-new}/logs` 内的 metad 相关日志，并在论坛提交问题。**放弃本次升级，在原目录正常启动 nebula 服务**。

1. 升级 storaged 数据格式。

   - 在每个 storaged 服务器运行如下命令。

```
$ sudo ${nebula-new}/bin/db_upgrader  \
--src_db_path=<old_storage_directory_path> \
--dst_db_path=<new_storage_directory_path>  \
--upgrade_meta_server=<meta_server_ip1>:<port1>[,<meta_server_ip2>:<port2>,...] \
--upgrade_version=<old_nebula_version> \
```

   参数说明：

   `--src_db_path` 为**老版本**storaged的数据目录的绝对路径，多个目录用逗号分隔，不加空格。

   `--dst_db_path` 为**新版本**storaged的数据目录的绝对路径，多个目录用逗号分隔。逗号分隔的目录必须和 `--src_db_path` 中一一对应。

   > **警告**: 两者顺序不要搞反了，否则会写坏旧版本的数据。

   `--upgrade_meta_server` 为步骤6中启动的所有新 metad 的地址.

   `--upgrade_version` 如果老版本为v1.2.0，则填写1；如果老版本为v2.0.0-RC，则填写2.不可填写其他数字。

   例如，从 v1.2.0 升级：

```
$ sudo /usr/local/nebula_new/bin/db_upgrader \
--src_db_path=/usr/local/nebula/data/storage/data1/,/usr/local/nebula/data/storage/data2/ \
--dst_db_path=/usr/local/nebula_new/data/storage/data1/,/usr/local/nebula_new/data/storage/data2/\
--upgrade_meta_server=192.168.8.14:45500,192.168.8.15:45500,192.168.8.16:45500 \
--upgrade_version=1
```

   从 v2.0.0-RC 升级：

```
$ sudo /usr/local/nebula_new/bin/db_upgrader \
--src_db_path=/usr/local/nebula/data/storage/ \
--dst_db_path=/usr/local/nebula_new/data/storage/ \
--upgrade_meta_server=192.168.8.14:9559,192.168.8.15:9559,192.168.8.16:9559 \
--upgrade_version=2
```

   > 如果工具抛出异常请在论坛提交问题。**放弃本次升级，关闭所有已经启动的 metad，在原目录正常启动 nebula 服务**。
   >
   > **检查**：每个 storaged 服务器都完成数据升级。

1. 在每个 storaged 服务器启动新版本的 storaged 服务。

```
$ sudo ${nebula-new}/scripts/nebula.service start storaged
$ sudo ${nebula-new}/scripts/nebula.service status storaged
```

> 说明：如果有 storaged 未正常启动，请将日志 `${nebula-new}/logs/` 在论坛提交问题。**放弃本次升级，关闭所有已经启动的 metad和storaged，在原目录正常启动 nebula 服务**。

2. 在每个 graphd 服务器启动新版本的 graphd 服务。

```
$ sudo ${nebula-new}/scripts/nebula.service start graphd
$ sudo ${nebula-new}/scripts/nebula.service status graphd
```

> 说明：如果有graphd未正常启动，请将日志 `${nebula-new}/logs/` 在论坛提交问题。**放弃本次升级，关闭所有已经启动的 metad,storaged,graphd. 在原目录正常启动 nebula 服务**。

3. 使用新版本Nebula Console [https://github.com/vesoft-inc/nebula-console] 连接新的Nebula Graph，验证服务是否可用、数据是否正常。命令行参数，如 graphd 的 IP、端口都不变。

```
nebula> SHOW HOSTS;
nebula> SHOW SPACES;
nebula> USE <space_name>
nebula> SHOW PARTS;
nebula> SUBMIT JOB STATS;
nebula> SHOW STATS;
```

> **禁止**：不可使用v2.0.0之前的 nebula console。

4. 升级其他客户端

所有的客户端都必须升级到对应 v2.0.0 版本。包括但不限于studio [https://github.com/vesoft-inc/nebula-docker-compose], python [https://github.com/vesoft-inc/nebula-python], java [https://github.com/vesoft-inc/nebula-java], go [https://github.com/vesoft-inc/nebula-go], c++ [https://github.com/vesoft-inc/nebula-cpp], flink-connector [https://github.com/vesoft-inc/nebula-flink-connector], spark-util [https://github.com/vesoft-inc/nebula-spark-utils], benchmark [https://github.com/vesoft-inc/nebula-bench]。请找到各 repo 对应的 v2.0.0 branch。

> 不兼容旧版本的通信协议。需重新源代码编译或者下载二进制包。
>
> (运维)提醒：升级后的数据目录为 `${nebula-new}/` 。如有硬盘容量监控、日志 ELK 等，相应改动。

4.4.4 升级失败回滚

如果升级失败，请停止新版本的所有服务，启动旧版的所有服务。

所有周边客户端切换为**旧版**。

4.4.5 附1：升级测试环境

本文测试升级的环境如下：

- 机器配置：32核CPU、62 GB内存、SSD
- 数据规模：Nebula Graph 1.2 版本 LDBC 测试数据 100 GB（1个图空间、24个分片、data目录 92 GB）
- 并发参数：`--max_concurrent=5`、`--max_concurrent_parts=24`、`--write_batch_num=100`

升级共耗时**21分钟**（其中 compaction 耗时13分钟）。工具并发参数说明如下：

参数名称	默认值
`--max_concurrent`	5
`--max_concurrent_parts`	10
`--write_batch_num`	100

4.4.6 附2：Nebula Graph v2.0.0 代码地址和 commit id

地址	commit id
graphd [https://github.com/vesoft-inc/nebula-graph/releases/tag/v2.0.0]	91639db
storaged 和 metad [https://github.com/vesoft-inc/nebula-storage/tree/v2.0.0]	761f22b
common [https://github.com/vesoft-inc/nebula-common/tree/v2.0.0]	b2512aa

4.4.7 FAQ

Q：升级过程中是否可以通过客户端写入数据？

A：不可以。这个过程中写入的数据状态是未定义的。

Q: 除了 v1.2.0 和 v2.0.0-RC 外，其他版本是否支持升级？

A: 未验证过。理论上 v1.0.0 - v1.2.0 都可以采用 v1.2.0 的升级版本。 v2.x 的日常研发版本(nightly)无升级方案。

Q: 升级完服务端后，客户端如何升级？

A：所有的客户端都必须升级到对应 v2.0.0 版本。包括但不限于 console [https://github.com/vesoft-inc/nebula-console], studio [https://github.com/vesoft-inc/nebula-docker-compose], python [https://github.com/vesoft-inc/nebula-python], java [https://github.com/vesoft-inc/nebula-java], go [https://github.com/vesoft-inc/nebula-go], c++ [https://github.com/vesoft-inc/nebula-cpp], flink-connector [https://github.com/vesoft-inc/nebula-flink-connector], spark-util [https://github.com/vesoft-inc/nebula-spark-utils], benchmark [https://github.com/vesoft-inc/nebula-bench]。请找到各repo对应的v2.0.0 branch (TODO： 部分branch还未发布）。不兼容旧版本的通信协议。

Q: 如果某台机器只有 graphd 服务，没有 storaged 服务，如何升级？

A: 那只需要升级 graphd 对应的 binary （或者rpm包）。

Q: 操作报错 `Permission denied`。

A: 部分命令需要有 sudo 权限。

Q: 是否有 gflags 发生改变？

A: 目前已知的 gflags 改变整理在github issues [https://github.com/vesoft-inc/nebula-graph/issues/858].

Q: 删除数据重新安装，和升级有何不同？

A: v2.x 的默认配置（包括端口）与 v1.x 不同。升级方案沿用老的配置，删除重新安装沿用新的配置。

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。

5. 配置和日志

5.1 配置

5.1.1 配置简介

本文简单介绍Nebula Graph的配置，以及如何正确使用配置文件。

获取配置

大多数配置都是gflags [https://gflags.github.io/gflags/]。您可以通过命令获取所有的gflags和解释：

```
binary --help
```

例如：

```
/usr/local/nebula/bin/nebula-metad  --help
/usr/local/nebula/bin/nebula-graphd --help
/usr/local/nebula/bin/nebula-storaged --help
./nebula-console --help
```

> **说明**：二进制文件的具体位置请根据实际情况修改，例如nebula-metad的默认路径为 `/usr/local/nebula/bin` 。

另外，您也可以使用 `curl` 获取运行中的gflags的值，例如：

```
curl 127.0.0.1:19559/flags # Meta服务
curl 127.0.0.1:19669/flags # Graph服务
curl 127.0.0.1:19779/flags # Storage服务
```

> **说明**：具体的IP地址和端口请根据实际情况修改。

修改配置

虽然默认情况下，Nebula Graph会从Meta服务获取配置并启动。 但是根据过往实践，建议您让Nebula Graph从本地文件获取配置。步骤如下：

1. 在**每一个**配置文件的开头添加 `--local_config=true` ，配置文件默认路径为 `/usr/local/nebula/etc/` 。
2. 保存修改内容，关闭配置文件。
3. 重启**所有**Nebula Graph服务，确保修改生效。

历史兼容性

v2.x `curl` 命令不兼容历史版本v1.x. 命令和参数都有改变。

5.1.2 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`，方便您在不同场景中使用。文件的默认路径为 `/usr/local/nebula/etc/`。

配置文件使用方式

首次启动时，Meta服务会从配置文件 `nebula-metad.conf` 中读取配置信息。您需要把初始配置文件的后缀 `.default` 或 `.production` 删除，Meta服务才能将其识别为配置文件。

如果修改了配置文件，希望新配置生效，请在配置文件开头添加 `--local_config=true` 再重启服务，否则会从缓存中读取过期配置。

配置文件参数值说明

如果配置文件内没有设置某个参数，表示参数使用的是默认值。

配置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。

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。

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）。建议您在调试时设置为 `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`）。

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_h2_port	19560	HTTP2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。

> **NOTE**: 建议您在配置文件中使用真实的IP地址，因为某些情况下 127.0.0.1 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。

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。

5.1.3 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`，方便您在不同场景中使用。文件的默认路径为 `/usr/local/nebula/etc/`。

配置文件使用方式

首次启动时，Graph服务会从配置文件 `nebula-graphd.conf` 中读取配置信息。您需要把初始配置文件的后缀 `.default` 或 `.production` 删除，Graph服务才能将其识别为配置文件。

如果修改了配置文件，希望新配置生效，请在配置文件开头添加 `--local_config=true` 再重启服务，否则会从缓存中读取过期配置。

配置文件参数值说明

如果配置文件内没有设置某个参数，表示参数使用的是默认值。

配置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。

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。

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）。建议您在调试时设置为 `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`）。

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。

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	0	空闲连接的超时时间。 0 表示永不超时。单位：秒。
session_idle_timeout_secs	0	空闲会话的超时时间。 0 表示永不超时。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。
ws_h2_port	19670	HTTP2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，如需使用请手动添加。默认值为 60000 毫秒。

> **NOTE**: 建议您在配置文件中使用真实的IP地址，因为某些情况下 127.0.0.1 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 [#7.data-security/1.authentication/1.authentication/:]。
`auth_type`	`password`	用户登录的身份验证方式。取值为 `password` 、 `ldap` 、 `cloud` 。

5.1.4 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`，方便您在不同场景中使用。文件的默认路径为 `/usr/local/nebula/etc/` 。

> **说明**：Raft Listener的配置和Storage服务配置不同，详情请参见部署Raft listener [#4.deployment-and-installation/6.deploy-text-based-index/3.deploy-listener/:]。

配置文件使用方式

首次启动时，Storage服务会从配置文件 `nebula-storaged.conf` 中读取配置信息。您需要把初始配置文件的后缀 `.default` 或 `.production` 删除，Storage服务才能将其识别为配置文件。

如果修改了配置文件，希望新配置生效，请在配置文件开头添加 `--local_config=true` 再重启服务，否则会从缓存中读取过期配置。

配置文件参数值说明

如果配置文件内没有设置某个参数，表示参数使用的是默认值。

配置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` 为准。

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。

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。可选值为 `0` （INFO）、`1` （WARNING）、`2` （ERROR）、`3` （FATAL）。建议您在调试时设置为 `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`	`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` ）。

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。
ws_h2_port	19780	HTTP2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。

> **NOTE**: 建议您在配置文件中使用真实的IP地址，因为某些情况下 127.0.0.1 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的生存时间。单位：秒。

disk配置

名称	默认值	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。
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（MB）。
engine_type	rocksdb	存储引擎类型。
rocksdb_compression	lz4	压缩算法，可选值为 no 、 snappy 、 lz4 、 lz4hc 、 zlib 、 bzip2 和 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。可选值为 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）和 kAll （收集所有统计数据）。
enable_rocksdb_prefix_filtering	false	是否启用prefix bloom filter。
enable_rocksdb_whole_key_filtering	true	是否启用whole key bloom filter。
rocksdb_filtering_prefix_length	12	每个key的prefix长度。可选值为 12 （分片ID+点ID）和 16 （分片ID+点ID+标签ID/边类型ID）。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
```

参数的详细说明请参见RocksDB官方文档 [https://rocksdb.org/]。

5.1.5 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，建议您设置为`unlimited`，命令如下：

```
ulimit -c unlimited
```

ULIMIT -N

`ulimit -n`用于限制打开文件的数量，建议您设置为超过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，建议您设置为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虚拟机保留的最小空闲千字节数。如果您的系统内存足够，建议您设置较大值。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。如果您的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，您可以根据需要进行调整（吞吐量优先或延迟优先），建议您使用系统默认值。

TRANSPARENT HUGE PAGE

为了降低延迟，您必须关闭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
```

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，建议您设置为 `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，建议您设置为不低于 `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，建议您设置为不低于 `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` ，建议您设置为 `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，建议您设置为 `带宽(GB)*往返时延(ms)` 。

SCHEDULER

对于SSD设备，建议您将 `scheduler` 设置为 `noop` 或者 `none` ，路径为 `/sys/block/DEV_NAME/queue/scheduler` 。

其他参数

KERNEL.CORE_PATTERN

建议您设置为 `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。

5.2 日志

5.2.1 日志配置

Nebula Graph使用glog [https://github.com/google/glog]打印日志，使用gflags [https://gflags.github.io/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。

日志目录

日志的默认目录为 `/usr/local/nebula/logs/` 。

如果您在Nebula Graph运行过程中删除日志目录，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）。建议您在调试时设置为 `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'
  ```

修改日志级别

使用如下命令修改日志级别：

```
$ curl -X PUT -H "Content-Type: application/json" -d '{"<key>":<value>[,"<key>":<value>]}' "<ws_ip>:<ws_port>/flags"
```

参数	说明
`key`	待修改的日志类型，可选值请参见配置说明 [#5.configurations-and-logs/2.log-management/logs/:_3]。
`value`	日志级别，可选值请参见配置说明 [#5.configurations-and-logs/2.log-management/logs/:_3]。
`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
```

如果您在Nebula Graph运行时修改了日志级别，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 [#5.configurations-and-logs/1.configurations/1.configurations/:]。

6. 监控

6.1 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。

6.1.1 监控指标说明

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`。指标说明如下。

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、COUNT、AVG、RATE和P分位数（P75、P95、P99、P99.9）。
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。

不同的Nebula Graph服务支持查询的监控指标也不同，详情请参见Nebula Graph服务监控指标（TODO: doc）。

6.1.2 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。

> **说明**：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

  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
```

7. 数据安全

7.1 验证和授权

7.1.1 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。

> **说明**：默认情况下，身份验证功能是关闭的，输入任意用户名和密码都可以连接到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 `--enable_authorize=true` 并保存退出。
2. 重启Nebula Graph服务。

> **说明**：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 [#7.data-security/1.authentication/3.role-list/:]。

LDAP验证

轻型目录访问协议（LDAP）是用于访问目录服务的轻型客户端-服务器协议，可以实现账号集中管理。 启用LDAP验证后，LDAP中存储的用户优先级高于本地用户。例如本地和LDAP都有一个名为 `Amber` 的用户，则优先从LDAP读取该用户的设置和角色信息。

启用LDAP验证

当前仅企业版支持集成LDAP进行身份验证，详情请参见使用LDAP进行身份验证（TODO: doc）。

7.1.2 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 [#7.data-security/1.authentication/1.authentication/:]后，您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 [#7.data-security/1.authentication/3.role-list/:]。

> **说明**：
>
> - 默认情况下，身份验证功能是关闭的，输入任意用户名和密码都可以连接到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>'];
```

- 示例

```
nebula> CREATE USER user1 WITH PASSWORD 'nebula';
```

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 [#7.data-security/1.authentication/3.role-list/:]。

- 语法

```
GRANT ROLE <role_type> ON <space_name> TO <user_name>;
```

- 示例

```
nebula> GRANT ROLE USER ON nba TO user1;
```

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 [#7.data-security/1.authentication/3.role-list/:]。

- 语法

```
REVOKE ROLE <role_type> ON <space_name> FROM <user_name>;
```

- 示例

```
nebula> REVOKE ROLE USER ON nba FROM user1;
```

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';
```

修改用户密码（ALTER USER）

执行 ALTER USER 语句可以修改用户密码，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>';
```

- 示例

```
nebula> ALTER USER user1 WITH PASSWORD 'nebula';
```

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。

> **说明**：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   |
+-----------+
| "test1"   |
+-----------+
| "test2"   |
+-----------+
| "test3"   |
+-----------+
```

7.1.3 内置角色权限

所谓角色，就是一组相关权限的集合。您可以把角色分配给创建的用户 [#7.data-security/1.authentication/2.management-user/:]，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。

> **说明**：
>
> - 暂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 [#7.data-security/1.authentication/2.management-user/:]。

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`
Write space	Y					`CREATE SPACE`、`DROP SPACE`、`CREATE SNAPSHOT`、`DROP SNAPSHOT`、`BALANCE`、`ADMIN`、`CONFIG`、`INGEST`、`DOWNLOAD`
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`、`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		`BUILD TAG INDEX`、`BUILD EDGE INDEX`、`INSERT VERTEX`、`UPDATE VERTEX`、`INSERT EDGE`、`UPDATE EDGE`、`DELETE VERTEX`、`DELETE EDGES`
Show operations	Y	Y	Y	Y	Y	`SHOW`、`CHANGE PASSWORD`

> **注意：**
>
> - 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。

7.2 备份恢复

7.2.1 什么是Backup&Restore

Backup&Restore（简称BR）是一款命令行界面（CLI）工具，可以帮助您备份Nebula Graph的图空间数据，或者通过备份文件恢复数据。

功能

- 支持备份一个或多个图空间的数据。
- 支持基于本地磁盘（SSD或HDD）、Hadoop分布式文件系统（HDFS）、阿里云对象存储（Alibaba Cloud OSS）或亚马逊对象存储（Amazon S3）中的备份文件恢复数据。

限制

- Nebula Graph版本需要为v2.0.0-RC或更新版本。
- 数据备份暂时仅支持全量备份，不支持增量备份。
- 数据备份过程中，指定图空间中的DDL和DML语句将会阻塞，我们建议您在业务低峰期进行操作，例如凌晨2点至5点。
- 数据恢复仅支持在相同拓扑的集群上进行，即原集群和目标集群的主机数量必须相同。
- 数据恢复需要删除数据并重启，建议离线进行。
- 备份或恢复部署在Docker中的数据时，需要做好网络配置，例如IP和端口的映射。

工作原理

备份

为了备份数据，BR会发送备份请求给leader的metad进程，触发备份。详细说明如下：

1. 验证BR访问Meta服务器和Storage服务器的SSH登录信息。

   > **说明**：如果必须使用云存储，例如Alibaba Cloud OSS或Amazon S3，还需要验证它们的客户端安装和配置。

2. BR发起请求创建备份文件。
3. leader的metad进程被锁定。

   > **注意**：从此时起至第9步结束，您无法在指定图空间内执行任何nGQL的DDL语句。

4. leader的metad进程阻塞指定图空间的写请求。

   > **注意**：从此时起至第7步结束，您无法在指定图空间内执行任何nGQL的DML语句，但是可以执行DQL语句。

5. leader的metad进程发送请求至storaged进程，请求快照文件名称。
6. leader的metad进程扫描本地RocksDB文件，输出为SST（Static Sorted Table）格式文件。
7. leader的metad进程解除阻塞指定图空间的写请求。

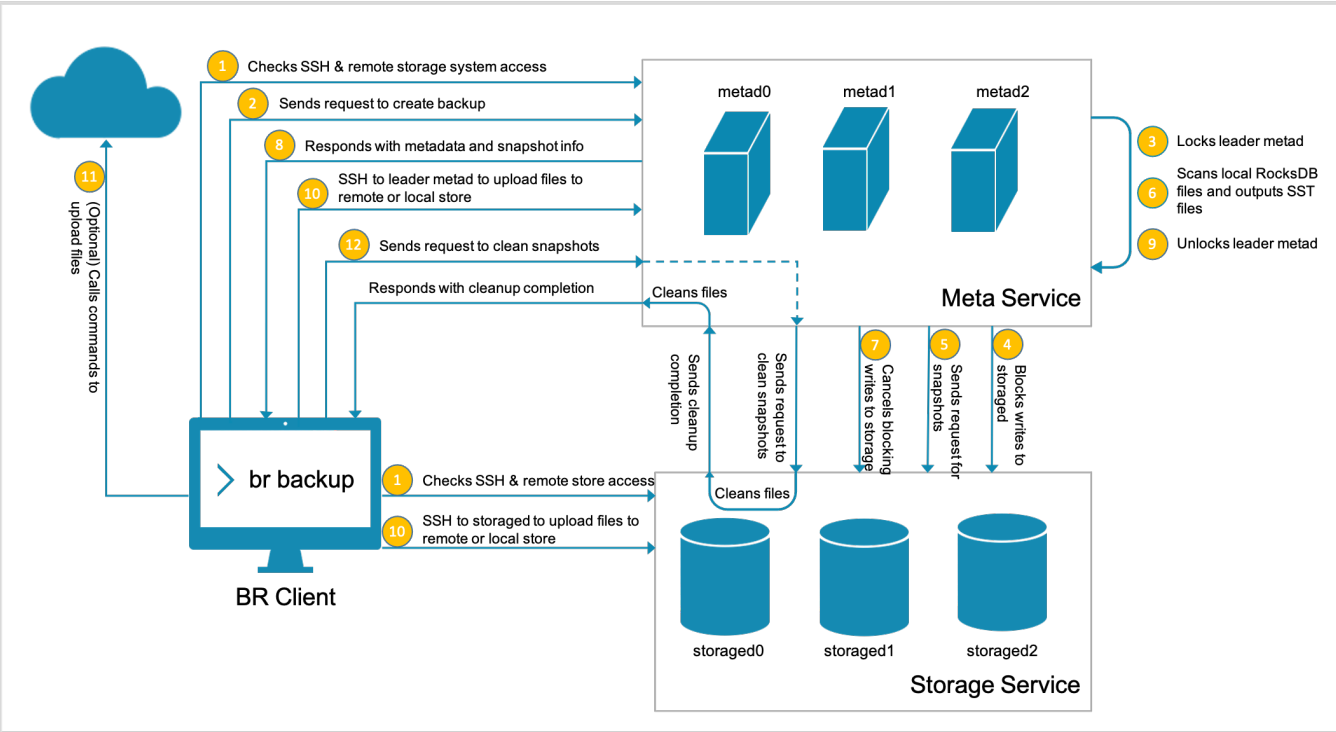
   > **说明**：从此时起，您可以在指定图空间内执行nGQL的DML语句。

8. leader的metad进程回应BR，包含的Meta数据和快照信息如下：
   - thrift格式信息
   - 图空间分区信息
   - 每个分区的Raft日志提交ID
   - 快照信息（每个快照存储进程的目录）
   - Meta服务器SST格式文件名称
   - 备份文件名称
9. leader的metad进程解除锁定。

   > **说明**：从此时起，您可以在指定图空间内执行任何nGQL的DDL语句。

10. BR通过SSH登录至leader所在的Meta服务器和所有Storage服务器，然后备份文件。
11. 如果使用的是Alibaba Cloud OSS或Amazon S3，BR会调用命令将备份文件上传至云存储中。 **>注意**：本步骤会大量读取磁盘，建议使用万兆网络保证速率。如果上传过程中出现网络错误，备份会失败，必须重新执行备份操作。目前备份过程不支持断点续传。
12. BR发起请求清理Storage服务器的快照，备份完成。

下图展示了备份的具体流程。



备份文件生成时会自动命名，命名格式为 `BACKUP_YY_MM_DD_HH_mm_SS` ：

- `BACKUP` 表示该文件是备份文件。
- `YY_MM_DD_HH_mm_SS` 表示该文件的生成时间。

恢复

> **警告**：恢复过程中，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。建议您提前备份目标集群上的数据。

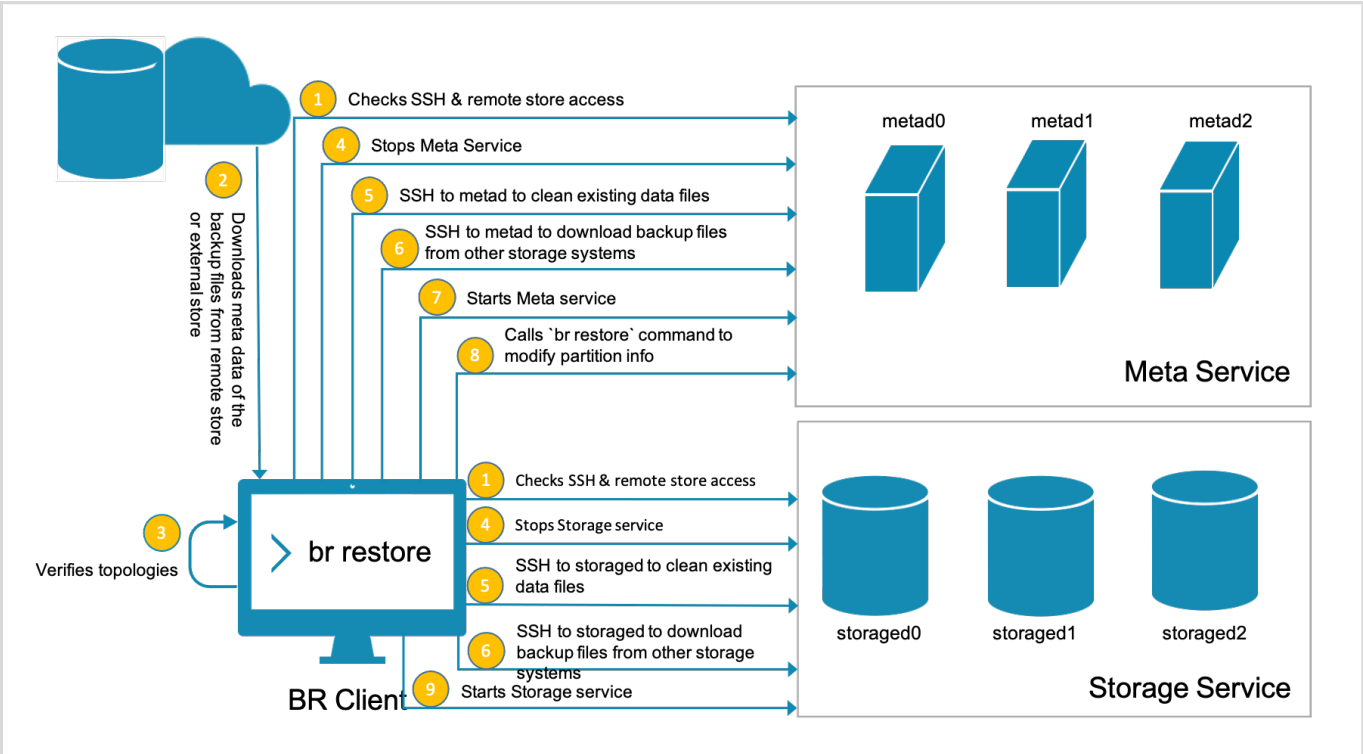
恢复过程的详细说明如下：

1. 验证BR访问Meta服务器和Storage服务器的SSH登录信息。

   > **说明**：如果必须使用云存储，例如Alibaba Cloud OSS或Amazon S3，还需要验证它们的客户端安装和配置。

2. BR从外部存储或云存储中下载Meta信息（非完整数据）。
3. BR验证集群的拓扑结构，确保原集群和目标集群的主机数量一致。
4. BR远程停止Meta服务和Storage服务。
5. BR通过SSH登录至leader所在的Meta服务器和所有Storage服务器，然后删除现有的数据文件。
6. 现有数据文件删除后，Meta服务器和所有Storage服务器从外部存储或云存储中下载备份文件。
7. 备份文件下载完成后，BR启动Meta服务。
8. BR调用 `br restore` 命令更改指定metad进程的分区信息。
9. BR启动Storage服务，恢复过程完成。

下图展示了恢复的具体流程。



如何使用BR

您可以按照如下步骤使用BR：

1. 编译BR [#7.data-security/2.backup-restore/2.compile-br/:]
2. 使用BR备份数据 [#7.data-security/2.backup-restore/3.br-backup-data/:]
3. 使用BR恢复数据 [#7.data-security/2.backup-restore/4.br-restore-data/:]

7.2.2 编译BR

目前，BR还没有作为一个包提供。您需要先编译BR。

准备工作

- 安装Go [https://github.com/golang/go] 1.14或更新版本。
- 安装make。

操作步骤

1. 克隆 `nebula-storage` 库至您的机器。

```
git clone https://github.com/vesoft-inc/nebula-storage.git
```

2. 进入 `br` 目录。

```
cd nebula-storage/util/br
```

3. 编译BR。

```
make build && make test
```

如果BR编译成功，您可以在 `nebula-storage/util/br/bin/` 目录内找到二进制文件 `br` 。

7.2.3 使用BR备份数据

BR编译成功后，您可以备份指定图空间的数据，本文介绍如何使用BR备份数据。

准备工作

- BR编译完成。如何编译BR，请参见编译BR [#7.data-security/2.backup-restore/2.compile-br/:]。
- 确认Nebula Graph服务正在运行中。
- 确认需要备份的图空间名称。本文示例为 `nba` 。
- 确认Nebula Graph的绝对安装路径。本文示例为 `/usr/local/nebula/` 。
- 确认Meta服务器和Storage服务器的IP和端口。
  - 在Nebula Graph的安装路径内查看 `nebula-metad.conf` 文件(例如 `/usr/local/nebula/etc/nebula-metad.conf`)，确认Meta服务器的IP和端口。本文示例为 `192.168.8.161:9559` 。
  - 在Nebula Graph的安装路径内查看和 `nebula-storaged.conf` 文件(例如 `/usr/local/nebula/etc/nebula-storaged.conf`)，确认Storage服务器的IP和端口。本文示例为 `192.168.8.161:9779` 。

  > **注意**：确保配置文件中使用的是实际的IP地址，而不是 `127.0.0.1` 。
- BR可以免密登录服务器，即您在BR机器上的账号可以通过SSH免密登录到Meta服务器和Storage服务器。详情请参见SSH tunnels with keys [http://alexander.holbreich.org/ssh-tunnel-without-password/]。本文示例账号名称为 `nebula` 。
- 如果您使用Alibaba Cloud OSS或Amazon S3保存备份文件，请确保Meta服务器、Storage服务器和BR机器都已安装相应的客户端。详情请参见Alibaba Cloud ossutil文档 [https://www.alibabacloud.com/help/zh/doc-detail/120075.htm#concept-303829]和Amazon S3 CLI文档 [https://docs.amazonaws.cn/cli/latest/userguide/cli-services-s3.html]。

  > **注意**：请创建软链接方便使用ossutil命令。命令为 `ln -s /<ossutil_tool_installation_path>/<ossutil64 or ossutil> /usr/local/bin/ossutil` ，根据实际路径和系统替换内容。
- 如果您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。本文示例为 `/home/user/backup/` 。

  > **注意**：在生产环境中，我们建议您将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。否则当您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 [#7.data-security/2.backup-restore/4.br-restore-data/:]。

操作步骤

1. 在 `nebula-storage/util/br/config` 目录内创建配置文件 `backup.yaml` 。您可以参考如下配置，目录内也有示例文件 `backup_example.yaml` 。

   ```yaml meta_nodes: - # 配置一个Meta服务器的IP和端口 addrs: "192.168.8.161:9559" # 配置Nebula Graph的绝对安装路径 root: "/usr/local/nebula/" # 配置metad进程数据目录的绝对路径 data: "/usr/local/nebula/data/meta" # 配置可以免密登录服务器的账号名称，需要对备份存储目录有读写权限 user: "nebula" #- # 如果存在多个metad进程，请
   ```

参考上述配置继续添加 # addrs: "192.168.8.161:9559" # root: "/usr/local/nebula/" # data: "/usr/local/nebula/data/meta" # user: "nebula" #- addrs: "192.168.8.161:9559" # root: "/usr/local/nebula/" # data: "/usr/local/nebula/data/meta" # user: "nebula"

storage_nodes: - # 配置一个Storage服务器的IP和端口 addrs: "192.168.8.161:9779" # 配置Nebula Graph的绝对安装路径 root: "/usr/local/nebula/" # 配置storaged进程数据目录的绝对路径 data: "/usr/local/nebula/data/storage" # 配置可以免密登录服务器的账号名称，需要对备份存储目录有读写权限 user: "nebula" #- # 如果存在多个storaged进程，请参考上述配置继续添加 # addrs: "192.168.8.161:9779" # root: "/usr/local/nebula/" # data: "/usr/local/nebula/data/storage" # user: "nebula" #- addrs: "192.168.8.161:9779" # root: "/usr/local/nebula/" # data: "/usr/local/nebula/data/storage" # user: "nebula"

设置备份文件的存储路径 # 如果在本地存储备份文件，请设置： backend: "local:///home/user/backup/" # 如果使用HDFS存储备份文件，请设置： # hdfs://namenode:port/nebulabackup # 如果使用Alibaba Cloud OSS存储备份文件，请设置： # backend: "oss://nebulabackup" # 如果使用Amazon S3存储备份文件，请设置： # backend: "s3://nebulabackup"

设置要备份的图空间 # 如果需要备份多个图空间，请设置： # space_names: ["space_name1", "space_name2", ..., "space_name"] space_name: ["nba"] ```

2. 切换到 `nebula-storage/util/br/bin/` 目录。

```
cd nebula-storage/util/br/bin/
```

3. 执行如下命令备份数据。

```
./br backup full --config "/home/vesoft/nebula-storage/util/br/config/backup.yaml"
```

- `backup full`：表示备份数据。
- `--config "/home/vesoft/nebula-storage/util/br/config/backup.yaml"`：填写配置文件的绝对路径，通过配置文件进行备份。

4. （可选）默认情况下，备份完成后，所有快照都会被删除。如果删除过程中出错，您可以执行如下命令手动删除快照。

```
./br cleanup --backup_name [backup file name] --meta 192.168.8.161:9559
```

- `cleanup`：表示删除Meta服务器和Storage服务器上的所有临时文件，包括快照。
- `--backup_name BACKUP_YY_MM_DD_HH_mm_SS`：指定要删除的备份文件名称。
- `--meta <IP address:port>`：指定Meta服务器的IP和端口。

下一步

备份文件生成后，您可以使用BR将备份文件的数据恢复到Nebula Graph中。具体操作，请参见使用BR恢复数据 [#7.data-security/2.backup-restore/4.br-restore-data/:]。

7.2.4 使用BR恢复数据

如果您使用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。

> **警告**：恢复过程中，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。建议您提前备份目标集群上的数据。
>
> **注意**：数据恢复需要离线进行。

准备工作

- BR编译完成。如何编译BR，请参见编译BR [#7.data-security/2.backup-restore/2.compile-br/:]。
- 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。
- 确认进行数据恢复的备份文件夹名称。本文示例为 `BACKUP_2020_12_21_01_17_53` 。
- 确认目标集群中Meta服务器和Storage服务器的IP和端口。
  - 在Nebula Graph的安装路径内查看 `nebula-metad.conf` 文件(例如 `/usr/local/nebula/etc/nebula-metad.conf`)，确认Meta服务器的IP和端口。本文示例为 `192.168.8.161:9559` 。
  - 在Nebula Graph的安装路径内查看和 `nebula-storaged.conf` 文件(例如 `/usr/local/nebula/etc/nebula-storaged.conf`)，确认Storage服务器的IP和端口。本文示例为 `192.168.8.161:9779` 。

  > **注意**：确保配置文件中使用的是实际的IP地址，而不是 `127.0.0.1` 。
- BR可以免密登录服务器，即您在BR机器上的账号可以通过SSH免密登录到Meta服务器和Storage服务器。详情请参见SSH tunnels with keys [http://alexander.holbreich.org/ssh-tunnel-without-password/]。本文示例账号名称为 `nebula` 。

  > **注意**:该账号必须有Nebula Graph安装目录的写权限。
- 如果您使用Alibaba Cloud OSS或Amazon S3保存备份文件，请确保Meta服务器、Storage服务器和BR机器都已安装相应的客户端。详情请参见Alibaba Cloud ossutil文档 [https://www.alibabacloud.com/help/zh/doc-detail/120075.htm#concept-303829]和Amazon S3 CLI文档 [https://docs.amazonaws.cn/cli/latest/userguide/cli-services-s3.html]。

  > **注意**：请创建软链接方便使用ossutil命令。命令为 `ln -s /<ossutil_tool_installation_path>/<ossutil64 or ossutil> /usr/local/bin/ossutil` ，根据实际路径和系统替换内容。
- 如果您的备份文件保存在本地磁盘，需要将BR机器上备份的Meta文件手动合并到Meta服务器和Storage服务器的备份目录内（和meta、storage目录平级）。本文示例为 `/home/user/backup/` 。

操作步骤

1. 在 `nebula-storage/util/br/config` 目录内修改配置文件 `backup.yaml` 。您可以参考如下配置，目录内也有示例文件 `restore_example.yaml` 。

   ```yaml meta_nodes: - # 配置一个Meta服务器的IP和端口 addrs: "192.168.8.161:9559" # 配置Nebula Graph的绝对安装路径 root: "/usr/local/nebula/" # 配置metad进程数据目录的绝对路径 data: "/usr/local/nebula/data/meta" # 配置可以免密登录服务器的账号名称，需要对备份存储目录有读写权限 user: "nebula" #- # 如果存在多个metad进程，请
   ```

参考上述配置继续添加 # addrs: "192.168.8.161:9559" # root: "/usr/local/nebula/" # data: "/usr/local/nebula/data/meta" # user: "nebula" #- addrs: "192.168.8.161:9559" # root: "/usr/local/nebula/" # data: "/usr/local/nebula/data/meta" # user: "nebula"

storage_nodes: - # 配置一个Storage服务器的IP和端口 addrs: "192.168.8.161:9779" # 配置Nebula Graph的绝对安装路径 root: "/usr/local/nebula/" # 配置storaged进程数据目录的绝对路径 data: "/usr/local/nebula/data/storage" # 配置可以免密登录服务器的账号名称，需要对备份存储目录有读写权限 user: "nebula" #- # 如果存在多个storaged进程，请参考上述配置继续添加 # addrs: "192.168.8.161:9779" # root: "/usr/local/nebula/" # data: "/usr/local/nebula/data/storage" # user: "nebula" #- addrs: "192.168.8.161:9779" # root: "/usr/local/nebula/" # data: "/usr/local/nebula/data/storage" # user: "nebula"

设置备份文件的存储路径 # 如果是本地存储备份文件，请设置： backend: "local:///home/user/backup/" # 如果使用HDFS存储备份文件，请设置： # hdfs://namenode:port/nebulabackup # 如果使用Alibaba Cloud OSS存储备份文件，请设置： # backend: "oss://nebulabackup" # 如果使用Amazon S3存储备份文件，请设置： # backend: "s3://nebulabackup"

设置要恢复的备份文件夹 backup_name: "BACKUP_2020_12_21_01_17_53" ```

2. 切换到 `nebula-storage/util/br/bin/` 目录。

```
cd nebula-storage/util/br/bin/
```

3. 执行如下命令恢复数据。

```
./br restore full --config "/home/vesoft/nebula-storage/util/br/config/backup.yaml"
```

- `restore full`：表示恢复数据。
- `--config "/home/vesoft/nebula-storage/util/br/config/backup.yaml"`：填写配置文件的绝对路径，通过配置文件进行恢复。

> **注意**：如果恢复过程中leader发生变更，会导致数据恢复出错，为保证数据正确性，请重新执行本步骤命令。

恢复成功后，您可以在Nebula Graph安装目录下的 `data/storage` 目录内找到数据文件，例如 `/usr/local/nebula/data/storage/`。

4. 等待一段时间，直至Meta数据和Schema同步，您可以登录数据库验证数据。例如运行`SHOW STATS` [#7.data-security/3.ngql-guide/7.general-query-statements/6.show/14.show-stats/:] 验证图空间中点和边的数量是否正确。

> **注意**：恢复完成后，您需要注意以下两点：
>
> - 如果执行 `USE <space_name>` 返回错误，建议您重启Nebula Graph服务。
> - 如果查询数据时报错 `Storage Error: part: 2, error code: -3.`，请检查Storage服务的状态，或者重启Storage服务后再次尝试查询。

7.3 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。

7.3.1 前提条件

Nebula Graph的身份认证 [#7.data-security/1.authentication/1.authentication/:]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 [#7.data-security/1.authentication/3.role-list/:]。

7.3.2 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。
- 暂不支持自动回收创建失败的快照垃圾文件，需要您手动删除。后续将在Meta服务中开发cluster checker功能，通过异步线程检查集群状态，并自动回收创建失败的快照垃圾文件。
- 暂不支持指定快照保存路径，默认路径为 `/usr/local/nebula/data` 。

7.3.3 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
...
```

7.3.4 创建快照

命令 `CREATE SNAPSHOT` 可以创建集群当前时间点的快照。暂时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。

> **说明**：如果快照创建失败，请删除快照 [#7.data-security/3.manage-snapshot/:_7]重新创建。

```
nebula> CREATE SNAPSHOT;
```

7.3.5 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" |
+--------------------------------+---------+-----------------+
```

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。|

7.3.6 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" |
+--------------------------------+---------+-----------------+
```

7.3.7 恢复快照

当前暂未提供恢复快照命令，需要您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Meta服务器和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，您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。

7.3.8 相关文档

除了使用快照，您还可以使用备份恢复工具Backup&Restore（BR）备份或恢复Nebula Graph数据。详情请参见 Backup&Restore [#7.data-security/2.backup-restore/1.what-is-br/:]。

8. 调整服务

8.1 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` ）。

> **说明：** `Compaction` 操作会长时间占用硬盘的IO，建议您在业务低峰期（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` 。

8.1.1 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。如果需要完全手动控制 `Compaction` 操作，您可以关闭自动 `Compaction` 。

关闭自动Compaction

> **警告：** 命令 `UPDATE CONFIGS` 会将未设置的参数恢复为默认值，因此修改前您需要使用 `SHOW CONFIGS STORAGE` 查看 `rocksdb_column_family_options` 配置，然后一起重新传入值。

```
# 查看当前rocksdb_column_family_options设置，复制value列内容。
nebula> SHOW CONFIGS STORAGE;
+-----------+-----------------------------------+-------+-----------
+-------------------------------------------------------------------------------------------------------+
| module    | name                              | type  | mode      |
value                                                                                                   |
+-----------+-----------------------------------+-------+-----------
+-------------------------------------------------------------------------------------------------------+
| "STORAGE" | "v"                               | "int" | "MUTABLE" |
0                                                                                                       |
+-----------+-----------------------------------+-------+-----------
+-------------------------------------------------------------------------------------------------------+
...
+-----------+-----------------------------------+-------+-----------
+-------------------------------------------------------------------------------------------------------+
| "STORAGE" | "rocksdb_column_family_options"     | "map" | "MUTABLE" | {max_bytes_for_level_base: "268435456",
max_write_buffer_number: "4", write_buffer_size: "67108864"} |
+-----------+-----------------------------------+-------+-----------
+-------------------------------------------------------------------------------------------------------+
...

# 修改rocksdb_column_family_options设置，在复制的value内容中添加disable_auto_compactions: true
nebula> UPDATE CONFIGS storage:rocksdb_column_family_options = {disable_auto_compactions: true, max_bytes_for_level_base:
"268435456", max_write_buffer_number: "4", write_buffer_size: "67108864"};

# 查看是否修改成功。
nebula> SHOW CONFIGS STORAGE;
```

```
+-----------+-----------------------------------+-------+-----------
+-------------------------------------------------------------------------------------------------------------------------------------------
+
| module    | name                              | type  | mode      |
value
|
+-----------+-----------------------------------+-------+-----------
+-------------------------------------------------------------------------------------------------------------------------------------------
+
| "STORAGE" | "v"                               | "int" | "MUTABLE" |
0
|
+-----------+-----------------------------------+-------+-----------
+-------------------------------------------------------------------------------------------------------------------------------------------
+
...
+-----------+-----------------------------------+-------+-----------
+-------------------------------------------------------------------------------------------------------------------------------------------
+
| "STORAGE" | "rocksdb_column_family_options"     | "map" | "MUTABLE" | {disable_auto_compactions: true,
max_bytes_for_level_base: "268435456", max_write_buffer_number: "4", write_buffer_size: "67108864"} |
+-----------+-----------------------------------+-------+-----------
+-------------------------------------------------------------------------------------------------------------------------------------------
+
...
```

8.1.2 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，该操作需要您手动发起。使用如下语句执行全量 `Compaction` 操作：

> **说明**：建议您在业务低峰期（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，您可以使用如下命令查看 `Compaction` 状态：

```
nebula> SHOW JOB <job_id>;
```

8.1.3 操作建议

为保证Nebula Graph的性能，请参考如下操作建议：

- 数据写入时为避免浪费IO，请在大量数据写入前关闭自动 `Compaction` 。详情请参见关闭自动`Compaction` [#8.service-tuning/compaction/:compaction_2]。
- 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` 。
- 白天时设置 `disable_auto_compactions` 为 `flase` ，提升短时间内的读取性能。
- 为控制 `Compaction` 的读写速率，请在配置文件 `nebula-storaged.conf` 中设置如下两个参数：

```
# 设置为从本地配置文件读取配置。
--local-config=true
# 读写速率限制为20MB/S。
--rate_limit=20 (in MB/s)
```

8.1.4 FAQ

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？

如果您已经设置读写速率限制，例如 `rate_limit` 限制为20MB/S时，您可以通过 `硬盘使用量/rate_limit` 预估需要耗费的时间。如果您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`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。

8.2 Storage负载均衡

您可以使用 `BALANCE` 语句平衡分片和Raft leader的分布，或者删除冗余的Storage服务器。

8.2.1 前提条件

为了平衡分片和Raft leader的分布，Nebula Graph中图空间的副本数都必须大于1。

8.2.2 均衡分片分布

`BALANCE DATA` 语句会开始一个任务，将Nebula Graph集群中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。

> **禁止**：不要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。

示例

以横向扩容Nebula Graph为例，集群中增加新的Storage服务器后，新服务器上没有分片。

1. 执行命令SHOW HOSTS [#3.ngql-guide/7.general-query-statements/6.show/6.show-hosts/:]检查分片的分布。

```
nebual> SHOW HOSTS;
+-------------+------+----------+--------------+----------------------+------------------------+
| Host        | Port | Status   | Leader count | Leader distribution  | Partition distribution |
+-------------+------+----------+--------------+----------------------+------------------------+
| "storaged0" | 9779 | "ONLINE" | 4            | "nba:4"              | "nba:15"               |
+-------------+------+----------+--------------+----------------------+------------------------+
| "storaged1" | 9779 | "ONLINE" | 8            | "nba:8"              | "nba:15"               |
+-------------+------+----------+--------------+----------------------+------------------------+
| "storaged2" | 9779 | "ONLINE" | 3            | "nba:3"              | "nba:15"               |
+-------------+------+----------+--------------+----------------------+------------------------+
| "storaged3" | 9779 | "ONLINE" | 0            | "No valid partition" | "No valid partition"   |
+-------------+------+----------+--------------+----------------------+------------------------+
| "storaged4" | 9779 | "ONLINE" | 0            | "No valid partition" | "No valid partition"   |
+-------------+------+----------+--------------+----------------------+------------------------+
| "Total"     |      |          | 15           | "nba:15"             | "nba:45"               |
+-------------+------+----------+--------------+----------------------+------------------------+
```

2. 执行命令 BALANCE DATA 将所有分片均衡分布。

```
nebula> BALANCE DATA;
+------------+
| ID         |
+------------+
| 1614237867 |
+------------+
```

3. 根据返回的任务ID，执行命令 BALANCE DATA <balance_id> 检查任务状态。

```
nebula> BALANCE DATA 1614237867;
+-------------------------------------------------------------+-------------------+
| balanceId, spaceId:partId, src->dst                         | status            |
+-------------------------------------------------------------+-------------------+
| "[1614237867, 11:1, storaged1:9779->storaged3:9779]"        | "SUCCEEDED"       |
+-------------------------------------------------------------+-------------------+
| "[1614237867, 11:1, storaged2:9779->storaged4:9779]"        | "SUCCEEDED"       |
+-------------------------------------------------------------+-------------------+
| "[1614237867, 11:2, storaged1:9779->storaged3:9779]"        | "SUCCEEDED"       |
+-------------------------------------------------------------+-------------------+
...
+-------------------------------------------------------------+-------------------+
| "Total:22, Succeeded:22, Failed:0, In Progress:0, Invalid:0" | 100               |
+-------------------------------------------------------------+-------------------+
```

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。

> **说明：** BALANCE DATA 不会均衡leader的分布。均衡leader请参见均衡leader分布 [#8.service-tuning/load-balance/:leader]。

```
nebula> SHOW HOSTS;
+-------------+------+----------+--------------+----------------------+------------------------+
| Host        | Port | Status   | Leader count | Leader distribution  | Partition distribution |
+-------------+------+----------+--------------+----------------------+------------------------+
| "storaged0" | 9779 | "ONLINE" | 4            | "nba:4"              | "nba:9"                |
+-------------+------+----------+--------------+----------------------+------------------------+
| "storaged1" | 9779 | "ONLINE" | 8            | "nba:8"              | "nba:9"                |
+-------------+------+----------+--------------+----------------------+------------------------+
| "storaged2" | 9779 | "ONLINE" | 3            | "nba:3"              | "nba:9"                |
+-------------+------+----------+--------------+----------------------+------------------------+
```

```
| "storaged3" | 9779 | "ONLINE" | 0            | "No valid partition" | "nba:9"                |
+-------------+------+----------+--------------+----------------------+------------------------+
| "storaged4" | 9779 | "ONLINE" | 0            | "No valid partition" | "nba:9"                |
+-------------+------+----------+--------------+----------------------+------------------------+
| "Total"     |      |          | 15           | "nba:15"             | "nba:45"               |
+-------------+------+----------+--------------+----------------------+------------------------+
```

如果有子任务失败，请重新执行 `BALANCE DATA` 。如果重做负载均衡仍然不能解决问题，请到Nebula Graph社区 [https://discuss.nebula-graph.io/]寻求帮助。

8.2.3 停止负载均衡

停止负载均衡，请执行命令 `BALANCE DATA STOP` 。

- 如果没有正在执行的负载均衡任务，会返回错误。
- 如果有正在执行的负载均衡任务，会返回停止的任务ID（ `balance_id` ）。

`BALANCE DATA STOP` 不会停止正在执行的子任务，而是取消所有后续子任务。您可以执行命令 `BALANCE DATA <balance_id>` 检查停止的任务状态。

一旦所有子任务都完成或停止，您可以再次执行命令 `BALANCE DATA` 。

- 如果前一个负载均衡任务的任何一个子任务失败，Nebula Graph会重新启动之前的负载均衡任务。
- 如果前一个负载均衡任务的任何一个子任务都没有失败，Nebula Graph会启动一个新的的负载均衡任务。

8.2.4 移除Storage服务器

移除指定的Storage服务器来缩小集群规模，可以使用命令 `BALANCE DATA REMOVE <host_list>` 。

示例

如果需要移除以下两台Storage服务器。

服务器名称	IP地址	端口
storage3	192.168.0.8	19779
storage4	192.168.0.9	19779

请执行如下命令：

```
BALANCE DATA REMOVE 192.168.0.8:19779,192.168.0.9:19779;
```

Nebula Graph将启动一个负载均衡任务，迁移storage3和storage4中的分片，然后将服务器从集群中移除。

8.2.5 均衡leader分布

`BALANCE DATA` 只能均衡分片分布，不能均衡Raft leader分布。您可以使用命令 `BALANCE LEADER` 均衡leader分布。

示例

```
nebula> BALANCE LEADER;
```

您可以执行 SHOW HOSTS 检查结果。

```
nebula> SHOW HOSTS;
+-------------+------+----------+--------------+----------------------+------------------------+
| Host        | Port | Status   | Leader count | Leader distribution  | Partition distribution |
+-------------+------+----------+--------------+----------------------+------------------------+
| "storaged0" | 9779 | "ONLINE" | 3            | "nba:3"              | "nba:9"                |
+-------------+------+----------+--------------+----------------------+------------------------+
| "storaged1" | 9779 | "ONLINE" | 3            | "nba:3"              | "nba:9"                |
+-------------+------+----------+--------------+----------------------+------------------------+
| "storaged2" | 9779 | "ONLINE" | 3            | "nba:3"              | "nba:9"                |
+-------------+------+----------+--------------+----------------------+------------------------+
| "storaged3" | 9779 | "ONLINE" | 3            | "nba:3"              | "nba:9"                |
+-------------+------+----------+--------------+----------------------+------------------------+
| "storaged4" | 9779 | "ONLINE" | 3            | "nba:3"              | "nba:9"                |
+-------------+------+----------+--------------+----------------------+------------------------+
| "Total"     |      |          | 15           | "nba:15"             | "nba:45"               |
+-------------+------+----------+--------------+----------------------+------------------------+
```

9. 社区贡献

9.1 如何贡献代码和文档

9.1.1 开始之前

github或社区提交问题

欢迎您为项目贡献任何代码或文档，但是建议您先在github [https://github.com/vesoft-inc/nebula-graph]或社区 [https://discuss.nebula-graph.io/]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 [https://www.apache.org/licenses/contributor-agreements.html]？

签署协议链接：vesoft inc. Contributor License Agreement [https://cla-assistant.io/vesoft-inc/]

单击按钮**Sign in with GitHub to agree**签署协议。

如果您有任何问题，请发送邮件至 `info@vesoft.com` 。

9.1.2 Step 1：通过GitHub fork仓库

Nebula Graph项目有很多仓库 [https://github.com/vesoft-inc]，以nebula-graph仓库 [https://github.com/vesoft-inc/nebula-graph]为例：

1. 访问https://github.com/vesoft-inc/nebula-graph [https://github.com/vesoft-inc/nebula-graph]。
2. 在右上角单击按钮 `Fork` ，然后单击用户名，即可fork出nebula-graph仓库。

9.1.3 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-graph.git
# 或：git clone git@github.com:$user/nebula-graph.git

cd $working_dir/nebula
git remote add upstream https://github.com/vesoft-inc/nebula-graph.git
```

```
# 或：git remote add upstream git@github.com:vesoft-inc/nebula-graph.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:$(user)/nebula-graph.git (fetch)
# origin    git@github.com:$(user)/nebula-graph.git (push)
# upstream  https://github.com/vesoft-inc/nebula-graph (fetch)
# upstream  no_push (push)
git remote -v
```

定义pre-commit hook

请将Nebula Graph的pre-commit hook连接到您的 `.git` 目录。

hook将检查您的commit，包括格式、构建、文档生成等。

```
cd $working_dir/nebula-graph/.git/hooks
ln -s $working_dir/nebula-graph/.linters/cpp/hooks/pre-commit.sh .
```

pre-commit hook有时候可能无法正常执行，您必须手动执行。

```
cd $working_dir/nebula-graph/.git/hooks
chmod +x pre-commit
```

9.1.4 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
```

> **说明**：由于一个PR通常包含多个commit，在合并至master时容易被挤压（squash），因此强烈建议您创建一个独立的分支进行更改。合并后，这个分支可以被丢弃，因此可以使用上述rebase命令将本地master与upstream同步。此外，如果直接将commit提交至 master，您可以需要在master分支使用hard reset，例如：

```
git fetch upstream
git checkout master
git reset --hard upstream/master
git push --force origin master
```

9.1.5 Step 4：开发

代码风格

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

请为您的新功能或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 [#4.deployment-and-installation/2.compile-and-install-nebula-graph/1.install-nebula-graph-by-compiling-the-source-code/:]。

> **说明**：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)
```

9.1.6 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，您需要更新head分支。

9.1.7 Step 6：Commit

提交代码更改：

```
git commit -a
```

您可以使用命令 `--amend` 重新编辑之前的代码。

9.1.8 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
```

9.1.9 Step 8：创建pull request

1. 访问fork出的仓库 `https://github.com/$user/nebula-graph` (替换此处的用户名 `$user`)。
2. 单击 `myfeature` 分支旁的按钮 `Compare & pull request` 。

9.1.10 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。

https://docs.nebula-graph.com.cn/