开源分布式图数据库 Nebula Graph 完全指南

book

吴敏,周瑶,梁振亚,张莹

2021 Vesoft Inc.

Table of contents

1. 本书更新版本 5
1.1 本书更新版本 5
2. 前言 6
2.1 图 6
2.2 图数据库的市场概况 18
2.3 相关技术 27
3. 简介 30
3.1 新一代开源分布式图数据库 Nebula Graph 30
3.2 数据模型 33
3.3 Nebula Graph架构 35
4. 快速入门 48
4.1 快速入门 48
4.2 Docker Compose部署Nebula Graph 49
4.3 管理Nebula Graph服务 54
4.4 连接Nebula Graph 57
4.5 基础操作语法 60
5. nGQL指南 70
5.1 nGQL概述 70
5.2 数据类型 75
5.3 变量和复合查询 87
5.4 运算符 92
5.5 函数和表达式 105
5.6 通用查询语句 126
5.7 子句和选项 162
5.8 图空间语句 183
5.9 标签语句 189
5.10 边类型语句 195
5.11 点语句 201
5.12 边语句 208
5.13 原生索引 215
5.14 全文索引 226
5.15 子图和路径 234
5.16 查询调优 240
5.17 运维 243

2021 Vesoft Inc.

6. 安装部署 247
6.1 准备编译、安装和运行Nebula Graph的环境 247
6.2 编译安装Nebula Graph 253
6.3 部署Nebula Graph集群 260
6.4 卸载Nebula Graph 265
7. 配置和日志 267
7.1 配置 267
7.2 日志 280
8. 监控 282
8.1 查询Nebula Graph监控指标 282
9. 数据安全 284
9.1 验证和授权 284
9.2 管理快照 289
10. 调整服务 291
10.1 Compaction 291
10.2 Storage负载均衡 293
11. Nebula Graph Studio 297
11.1 认识 Nebula Graph Studio 297
11.2 安装与登录 302
11.3 快速开始 311
11.4 操作指南 319
11.5 故障排查 339
12. Nebula Importer 341
12.1 Nebula Importer 341
12.2 有表头配置说明 347
12.3 无表头配置说明 350
13. Nebula Spark Connector 353
13.1 适用场景 353
13.2 优势 353
13.3 获取Nebula Spark Connector 353
13.4 使用方法 354
14. Nebula Flink Connector 358
14.1 适用场景 358
15. Nebula Algorithm 359
15.1 使用限制 359
15.2 支持算法 359
15.3 实现方法 359

2021 Vesoft Inc.

15.4 获取Nebula Algorithm 359
15.5 使用方法 360
16. Nebula Exchange 364
16.1 认识Nebula Exchange 364
16.2 编译Exchange 367
16.3 参数说明 369
16.4 使用Nebula Exchange 375
16.5 Exchange常见问题 428
17. 附录 430
17.1 常见问题 430
17.2 图建模与系统设计 434
17.3 系统设计建议 435
17.4 关于 Raft 的简单介绍 436
17.5 分片ID 437
17.6 生态工具概览 438
17.7 点VID 441
17.8 注释 442
17.9 大小写区分 443
17.10 关键字 444
17.11 Nebula Graph 年表 447

2021 Vesoft Inc.

1. 本书更新版本

1.1 本书更新版本

> **本书更新时间2021-6-3, GitHub commit 774f7cf1.**

最后更新: 2021年5月19日

2021 Vesoft Inc.

2. 前言

2.1 图

图是计算机科学研究的主要领域之一。图能够高效地解决目前存在的诸多问题。本章将说明图数据库在使用方面的优点以及图数据库在现代应用程序开发中的巨大潜力。并介绍分布式图数据库的区别和几种其他类型的数据库。当前，从计算机行业巨头（例如 Amazon 和 Facebook）到小型研究团队，都投入了大量的资源探索图数据库在解决各种数据关系问题上的潜力。当然你也可以选择像他们这样进行尝试。现在可供选择的数据库有很多，那么图数据库在数据库领域里处于什么位置呢？

2.1.1 图、图片与图论

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

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

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

2021 Vesoft Inc.

但是——总会有但是——我们在本书中讨论的图是另外一个概念——“图论”。

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

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

"这才是本书所说的图"

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

2021 Vesoft Inc.

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

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

2021 Vesoft Inc.

2.1.2 属性图

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

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

A.无向图

B.有向图

- 此外，在节点和边上，还可以有属性（properties）。

在现实生活中，有很多属性图的例子。

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

2021 Vesoft Inc.

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

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

2021 Vesoft Inc.

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

2021 Vesoft Inc.

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

2021 Vesoft Inc.

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

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

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

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

代码依赖分析
- 图谱数据
  - 将公司代码库中代码的依赖关系写入图谱
  - 包含method、field、class、interface等4类顶点、7类关系
  - 点边数量在千万级别，实时写入
- 用于QA精准测试
  - RD向代码仓库提交PR后，能查询出所修改代码能影响到的外部接口，并展示调用路径

```
//查找最外层method到某个method的所有无环路径
(GO 1 to 30 STEPS FROM 2946345526231222882 OVER
method_call_method REVERSELY YIELD DISTINCT
method_call_method._dst AS id
MINUS
GO 1 to 30 STEPS FROM 2946345526231222882 OVER
method_call_method REVERSELY YIELD DISTINCT
method_call_method._src as id )
| FIND NOLOOP path FROM $-.id TO
2946345526231222882 OVER method_call_method UPTO
30 STEPS
```

```
//确认两个method间是否有路径
FIND SINGLE SHORTEST PATH FROM hash("method1")
TO hash("method2") OVER method_call_method UPTO
30 STEPS
```

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

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

2021 Vesoft Inc.

同样的方法也可以用于探索发现一些加密货币的流向。

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

2021 Vesoft Inc.

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

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

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

2021 Vesoft Inc.

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

2021 Vesoft Inc.

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

---

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

最后更新: 2021年5月20日

2021 Vesoft Inc.

2.2 图数据库的市场概况

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

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

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

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

2.2.1 三方机构的统计和预测

DB-Engines 的统计

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

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

Gartner 的预测

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

2021 Vesoft Inc.

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

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

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

2021 Vesoft Inc.

> **趋势八：图技术使一切产生关联（Graph Relates Everything）**
>
> 图技术已成为许多现代数据和分析能力的基础，能够在不同的数据资产中发现人、地点、事物、事件和位置之间的关系。数据和分析领导者依靠图技术快速回答需要在了解情况并理解多个实体之间的联系和优势的性质后才能回答的复杂业务问题。
>
> Gartner预测，到2025年图技术在数据和分析创新中的占比将从2021年的10%上升到80%。该技术将促进整个企业机构的快速决策。

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

2021 Vesoft Inc.

对于市场规模的预测

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

2.2.2 市场参与者

（第一代）图数据库的先行者 Neo4j

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

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

> **关于图查询查询(Graph Query Language，GQL) 和国际标准的制定**
>
> 熟悉数据库的读者可能都知道结构化查询语言SQL。通过使用SQL，人们以接近自然语言的方式访问数据库。在 SQL 被广泛采用和标准化之前，关系型数据库的市场是非常碎片和割裂的——各家厂商的产品都有完全不同的接入访问方式，数据库产品自身的开发人员、数据库产品周边工具的开发人员、数据库最终的使用人员，都不得不学习各个厂商的完全不同的产品，在不同产品之间迁移极其困难。当1989年SQL-89标准被制定后，整个关系型数据库的市场快速收敛到SQL-89上。这大大降低了上述各种人员的学习曲线。
>
> 类似的，在图数据库领域，图语言(GQL)承担了类似于SQL的作用，是一种用户与图数据库主要的交互方式。但不同于SQL-89这种国际标准，GQL还没有任何国际标准。目前有两种主流的图语言： Neo4j的Cypher (及其后续——ISO正在制定过程中的GQL-standard 草案)和Apache TinkerPop的Gremlin。前者通常被称为声明式语言(Declarative query language)——也即用户只需要告诉系统“要什么”，而不管“怎么做”；后者通常被称为命令式语言(Imperative query language)，用户会显式地指定系统的操作。
>
> GQL国际标准正在制定过程中。

年表简述

- 2000 年，Neo4j 的创始人产生将数据建模成网络（network）的想法。

2021 Vesoft Inc.

- 2001 年，Neo4j 开发了最早的核心部分代码。
- 2007 年，Neo4j 开始以一个公司的方式运作。
- 2009 年，Neo4j 团队借鉴 XPath 作为图查询语言，Gremlin[10]最初也是基于这个想法。
- 2010 年，Neo4j 的员工 Marko Rodriguez 采用术语属性图（Property Graph）来描述 Neo4j 和 Tinkerpop / Gremlin 的数据模型。
- 2011 年，第一个公开发行版本 Neo4j 1.4; 并发布了Cypher的第一个版本。
- 2012 年，Neo4j 1.8 为 Cypher 增加写入图的能力。Neo4j 2.0 增加了标签和索引，Cypher 成为一种声明式的语言。
- 2015 年，Neo4j 将 Cypher 开源为 openCypher。
- 2017 年，ISO WG3 工作组开始讨论如何将属性图查询能力引入 SQL。
- 2018 年 12 月，从 Neo4j 3.5 开始其核心部分转为闭源。
- 2019 年, ISO 正式立项两个项目(ISO/IEC JTC 1 N 14279和ISO/IEC JTC 1/SC 32 N 3228），启动关于图数据库语言国际标准的制定工作。

NEO4J 的早期历史

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

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

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

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

GREMLIN 的创造

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

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

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

CYPHER 的创造

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

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

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

2021 Vesoft Inc.

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

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

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

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

2021 Vesoft Inc.

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

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

后续

Cypher 启发了一系列后续的图查询语言，包括

2015 年，Oracle 发布图引擎PGX使用的图语言PGQL。

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

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

2019 年，国际标准组织 ISO 启动两个项目，基于 openCypher, PGQL, GSQL[11], and G-CORE 等现有业界成果，启动图语言国际标准的制定过程

2020 年，Nebula Graph 以 openCypher 为基础发布其扩展的图语言 Nebula Graph Query Language, nGQL。

分布式图数据库

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

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

> **说明**
>
> 严格说，Neo4j 也提供了不少的分布式的能力，但都和业界意义上的分布式系统有较大的不同。
>
> Neo4j 3.X 要求全量数据必须存放在单机中。虽然其也提供多机之间(Master-slave/slave)做全量复制和高可用，但数据不可切分为不同子图存放。
>
> Neo4j 4.X 允许在不同机器上各存放一部分数据（子图），然后在应用层需通过一定方式拼装后(其称为编织Fabric)，将读写分发到各个机器上。这种做法需要应用层代码有大量的参与和工作。例如，设计如何把不同子图应该放置在哪些机器上，如何将从各机器获取的部分结果重新编织为最终的结果。

第二代（分布式）图数据库：TITAN 和其后继者 JANUSGRAPH

2011 年，Aurelius 公司成立，致力于开发一个开源的分布式图数据库 Titan[12]。到 2015 年 Titan 的第一个正式版发布，Titan 后端可以支持多种主流的分布式存储架构（例如 Cassandra, HBase, Elasticsearch, BerkeleyDB)，并可以复用 Hadoop 生态的诸多便利，前端以 Gremlin 为统一的查询语言。对于程序员使用、开发和社区参与都很方便。大规模的图，可以分片后存放在 HBase 或者 Cassarndar上(这些当时都已经是相对成熟的分布式存储方案)，Gremlin 语言虽然略微冗长但相对功能完备。整个方案在当时(2011-2015)体现了不错的竞争力。

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

2021 Vesoft Inc.

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

在 Aurelius(Titan) 被收购后，市场对于开源分布式的图数据库一直仍有比较强烈的需求，而当时市场上成熟和活跃的产品并不多。大数据时代，数据仍在远快于摩尔定律的速度，源源不断的产生。Linux 基金会以及一些技术巨头(Expero, Google, GRAKN.AI, Hortonworks, IBM and Amazon) 在2017年，复制并分叉(fork)了原有的Titan项目，并启动为一个新项目 JanusGraph[13]。之后大多数的社区工作，包括开发、测试、发布和推广都逐步转移到了新的 JanusGraph。

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

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

2021 Vesoft Inc.

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

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

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

传统巨头微软、亚马逊和甲骨文纷纷入场

除了聚焦于图产品的厂商外，传统巨头也纷纷进入这个领域。

Microsoft Azure Cosmos DB[14] 是一个在微软云上的多模数据库云服务，可以提供SQL、文档、图、key-value等多种能力； Amazon AWS Neptune[15] 是一种由 AWS 提供图数据库云服务，可以提供属性图和 RDF 两种数据模型； Oracle graph[16] 是关系型数据库巨头 Oracle 在图技术与图数据库方向的产品。

新一代开源分布式图数据库 NEBULA GRAPH

下一节，我们将正式介绍新一代开源分布式图数据库 Nebula Graph。

---

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

最后更新: 2021年5月27日

2021 Vesoft Inc.

2.3 相关技术

本节主要介绍两个和分布式图数据库关系密切的领域，数据库方面和图技术方面。

2.3.1 数据库方面

关系型数据库

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

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

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

关系型数据库中的关系

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

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

关系型数据库起源

上文提到，关系型数据模型最早是由 IBM 的工程师埃德加·科德（Edgar Frank Codd）于 1970 年提出的。科德写了几篇数据库管理系统方面的论文，论述了关系型数据模型的潜力。关系型数据模型不依赖于数据链接列表（网状数据或层级数据），而是更多依赖于数据集。他使用元组演算

2021 Vesoft Inc.

（tuple calculus）的数学方法论证了这些数据集能够完成与导航数据库管理系统相同的任务。唯一的要求是，关系型数据模型需要一种合适的查询语言，以保证数据库的一致性要求。这就为后来声明型的结构化查询语言（SQL）提供了灵感来源。IBM 的 R 系统是关系型数据模型的最早使用者之一。然而，由前 IBM 员工拉里·埃里森创办的名叫软件开发实验室的小公司在市场上击败了 IBM。该公司的产品就是后来为我们熟知的 Oracle。

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

NoSQL 数据库

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

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

下面将分别介绍这四类数据库。

键值存储

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

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

列式存储

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

文档存储

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

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

图存储

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

2.3.2 图技术方面

来看一张 2020 年的图技术全景[1]

2021 Vesoft Inc.

和图有关联的技术有很多，可以大致分为这么几类：

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

1. https://graphaware.com/graphaware/2020/02/17/graph-technology-landscape-2020.html ↩

最后更新: 2021年5月27日

2021 Vesoft Inc.

3. 简介

3.1 新一代开源分布式图数据库 Nebula Graph

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

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

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。

3.1.1 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 社区的设计和开发中来，欢迎访问Nebula Graph GitHub主页参与开源项目。目前最新发布的版本为 book.

2021 Vesoft Inc.

高性能

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

易扩展

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

易开发

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

高可靠访问控制

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

生态多样化

Nebula Graph开放了越来越多的原生工具，例如Nebula Graph Studio、Nebula Console、Nebula 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。

灵活数据建模

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

广受欢迎

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

3.1.2 适用场景

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

欺诈检测

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

实时推荐

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

知识图谱

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

2021 Vesoft Inc.

社交网络

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

3.1.3 相关链接

- 官方网站
- 文档首页
- 博客首页
- 论坛
- GitHub

最后更新: 2021年5月24日

2021 Vesoft Inc.

3.2 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。

3.2.1 数据结构

Nebula Graph数据模型使用6种基本的数据结构：

- 图空间(space)

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

- 点（vertex）

  点用来保存实体对象，特点如下：

  - 点是用点标识符（`VID`）标识的。`VID`在同一图空间中唯一。VID 是一个 int64, 或者 fixed_string(N)。
  - 点必须有至少一个标签（Tag），也可以有多个标签。

- 边（edge）

  边是用来连接点的，表示两个点之间的关系或行为，特点如下：

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

- 标签（tag）

  标签由一组事先预定义的属性构成。

- 边类型（edgetype）

  边类型由一组事先预定义的属性构成。

- 属性（properties）

  属性是指以键值对（key-value）形式存储的信息。

> **Note**
>
> Tag和edgetype的作用，类似于关系型数据库中点表和边表的表结构。

3.2.2 有向属性图

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

2021 Vesoft Inc.

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

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

Note

Nebula Graph 中没有无向边，只支持有向边。

最后更新: 2021年5月20日

2021 Vesoft Inc.

3.3 Nebula Graph架构

3.3.1 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，用户可以使用这些二进制文件在一个或多个计算机上部署Nebula Graph集群。

下图展示了Nebula Graph集群的经典架构。

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。

2021 Vesoft Inc.

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，可以进一步节约前期成本。

- 更多可能性

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

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

最后更新: 2021年5月14日

2021 Vesoft Inc.

3.3.2 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，用户可以根据场景配置nebula-metad进程数量：

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

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

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

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

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。

2021 Vesoft Inc.

更多Nebula Graph的访问控制说明，请参见身份验证。

管理分片

Meta服务负责存储和管理分片的位置信息，并且保证分片的负载均衡。

管理图空间

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

管理SCHEMA信息

Nebula Graph是强类型图数据库，它的Schema包括标签、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。

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。

最后更新: 2021年5月14日

2021 Vesoft Inc.

3.3.3 Graph服务

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

Graph服务架构

查询请求发送到Graph服务后，会由如下模块依次处理：

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

Parser

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

例如 `GO FROM "Tim" OVER like WHERE like.likeness > 8.0 YIELD like._dst` 语句转换的AST如下。

2021 Vesoft Inc.

Validator

Validator模块对生成的AST进行语义校验，主要包括：

- 校验元数据信息

  校验语句中的元数据信息是否正确。

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

- 校验上下文引用信息

  校验引用的变量是否存在或者引用的属性是否属于变量。

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

- 校验类型推断

  推断表达式的结果类型，并根据子句校验类型是否正确。

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

- 校验 `*` 代表的信息

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

  例如语句 `GO FROM "Tim" OVER * YIELD like._dst, like.likeness, serve._dst` ，校验 `OVER` 子句时需要校验所有的边类型，如果边类型包含 `like` 和 `serve` ，该语句会展开为 `GO FROM "Tim" OVER like,serve YIELD like._dst, like.likeness, serve._dst` 。

- 校验输入输出

  校验管道符（|）前后的一致性。

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

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

Planner

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

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

2021 Vesoft Inc.

- 优化前

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

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

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

- 优化过程

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

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

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

> **Note**
>
> Nebula Graph book 默认没有打开优化。

Executor

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

2021 Vesoft Inc.

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

代码结构

Nebula Graph的代码层次结构如下：

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

最后更新: 2021年5月18日

2021 Vesoft Inc.

3.3.4 Storage服务

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

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

优势

- 高性能（自研KVStore）
- 易扩展（Shared-nothing架构）
- 强一致性（Raft）
- 高可用性（Raft）
- 支持向第三方系统进行同步（例如全文索引）

Storage服务架构

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

2021 Vesoft Inc.

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

- Storage interface层

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

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

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

- Consensus层

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

- Store Engine层

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

下文将基于架构介绍Storage服务的部分特性。

自研KVStore

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

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

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

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

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

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

数据存储格式

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

2021 Vesoft Inc.

由于Nebula Graph 2.0的数据存储格式在1.x的基础上做了修改，下文将在介绍数据存储格式时同时介绍不同版本的差异。

- 点数据存储格式

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

- 边数据存储格式

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

2021 Vesoft Inc.

> **Nebula Graph 2.0和1.x的差异**
>
> - 1.x中，点和边的 `Type` 值相同，而在2.0中进行了区分，即在物理上分离了点和边，方便快速查询某个点的所有标签。
> - 1.x中， `VertexID` 仅支持int类型，而在2.0中新增了string类型。
> - 2.0中取消了1.x中的保留字段 `Timestamp` 。
> - 2.0中边数据新增字段 `PlaceHolder` 。
> - 2.0中修改了索引的格式，以便支持范围查询。

属性说明

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

数据分片

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

对于在线图查询来说，最常见的操作便是从一个点开始向外拓展（广度优先），查询一个点的出边或者入边是最基本的操作，同时可能会根据属性进行过滤，Nebula Graph通过将属性与点边存储在一起，保证了整个操作的高效。

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

Raft

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

MULTI GROUP RAFT

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

实现Multi Group Raft有2个关键点：

- 共享Transport层

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

- 共享线程池

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

批量（BATCH）操作

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

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

LEARNER角色

learner角色的存在主要是为了应对**扩容**，扩容时新增的机器需要很长时间去同步数据，如果以follower角色同步数据，会导致整个集群的高可用性下降。

2021 Vesoft Inc.

新增learner角色后，会写入command类型的WAL，leader在写WAL时如果发现有 `add learner` 的command，会将learner加入自己的peers，并将它标记为learner，在统计多数派的时候，不会算上learner，但是日志还是会照常发送给它们，learner本身也不会主动发起选举。

LEADER切换（TRANSFER LEADERSHIP）

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

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

成员变更

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

LISTENER

Raft listener进程可以从Storage服务获取数据，然后将它们写入Elasticsearch集群，以便实现全文搜索。详情请参见部署Raft listener。

与HDFS的区别

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

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

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

最后更新: 2021年5月19日

2021 Vesoft Inc.

4. 快速入门

4.1 快速入门

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

1. Docker Compose部署Nebula Graph
2. 连接Nebula Graph
3. 基础操作语法

最后更新: 2021年5月18日

2021 Vesoft Inc.

4.2 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。

4.2.1 阅读指南

本节将回答如下问题：

- 部署Nebula Graph之前需要做什么准备工作？
- 如何通过Docker Compose快速部署Nebula Graph？
- 如何检查Nebula Graph服务的状态和端口？
- 如何检查Nebula Graph服务的数据和日志？
- 如何停止Nebula Graph服务？
- 如何通过其他方式部署Nebula Graph？

4.2.2 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
Docker Compose	最新版本	Install Docker Compose
Git	最新版本	Download Git

- 如果使用非root用户部署Nebula Graph，请授权该用户Docker相关的权限。详细信息，请参见Manage Docker as a non-root user。
- 启动主机上的Docker服务。
- 如果已经通过Docker Compose在主机上部署了另一个版本的Nebula Graph，为避免兼容性问题，需要删除目录 `nebula-docker-compose/data` 。

4.2.3 部署和连接Nebula Graph

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

> **请不要在生产环境使用此版本**
>
> `master` 分支包含最新的未测试代码。请**不要**在生产环境使用此版本。

```
$ git clone 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。

> **Note**
>
> 如果长期未更新镜像，请先更新Nebula Graph镜像和Nebula Console镜像。

2021 Vesoft Inc.

```
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
```

Note

上述服务的更多信息，请参见架构总览。

4. 连接Nebula Graph。

   a. 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
```

Note

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
```

   b. 通过Nebula Console连接Nebula Graph。

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

Note

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。

   c. 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。

4.2.4 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
```


2021 Vesoft Inc.

```
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。

4.2.5 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
```

4.2.6 停止Nebula Graph服务

用户可以执行如下命令停止Nebula Graph服务：

```
$ docker-compose down
```

如果返回如下信息，表示已经成功停止服务。

```
Stopping nebula-docker-compose_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
```

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

2021 Vesoft Inc.

4.2.7 常见问题排查

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`

可能遇到如下错误：

`ERROR: toomanyrequests: You have reached your pull rate limit. You may increase the limit by authenticating and upgrading: https://www.docker.com/increase-rate-limit`

以上错误表示已达到Docker Hub的速率限制。解决方案请参见Understanding Docker Hub Rate Limiting。

如何更新Nebula Console？

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 book-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。

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。

4.2.8 相关文档

- 使用源码安装部署Nebula Graph

2021 Vesoft Inc.

- 使用RPM或DEB安装包安装Nebula Graph
- 多种方式连接Nebula Graph

最后更新: 2021年5月19日

2021 Vesoft Inc.

4.3 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。

4.3.1 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。

4.3.2 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
```


2021 Vesoft Inc.

4.3.3 停止Nebula Graph服务

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

非容器部署

执行如下命令停止Nebula Graph服务：

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

容器部署

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

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

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

4.3.4 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
  ```


2021 Vesoft Inc.

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

容器部署

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

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

如果服务有异常，用户可以先确认异常的容器名称（例如 `nebula-docker-compose_graphd2_1`），然后执行 `docker ps` 查看对应的 `CONTAINER ID`(示例为 `2a6c56c405f5`)，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]#
```

最后更新: 2021年5月14日

2021 Vesoft Inc.

4.4 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Nebula Console。

4.4.1 Nebula Graph客户端

用户可以使用已支持的客户端或者命令行工具来连接Nebula Graph数据库。

4.4.2 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。
- 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页面，确认需要的版本，单击**Assets**。

> **Note**
>
> 建议选择**最新**版本。

2. 在**Assets**区域找到机器运行所需的二进制文件，下载文件到机器上。

2021 Vesoft Inc.

3. （可选）为方便使用，重命名文件为 `nebula-console` 。

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

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

> **Note**
>
> Windows系统请跳过此步骤。

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

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

用户可以在项目仓库找到更多细节.

2021 Vesoft Inc.

4.4.3 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：

> **Note**
>
> - 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
```

4.4.4 使用Nebula Console断开连接

用户可以使用 `:EXIT` 或者 `:QUIT` 从Nebula Graph断开连接。为方便使用，Nebula Console支持使用不带冒号（:）的小写命令，例如 `quit`。

```
nebula> :QUIT

Bye root!
```

4.4.5 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页面的说明。

最后更新: 2021年5月25日

2021 Vesoft Inc.

4.5 基础操作语法

本文介绍Nebula Graph基础操作的语法。

4.5.1 图空间和Schema

一个Nebula Graph集群由一个或多个图空间组成。每个图空间都是物理隔离的，用户可以在同一个集群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。

本文将使用下图的数据集演示基础操作的语法。

2021 Vesoft Inc.

4.5.2 检查Nebula Graph集群的机器状态

首先建议检查机器状态，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。

2021 Vesoft Inc.

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`

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

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）。

4.5.3 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 `basketballplayer` 的图空间。

```
nebula> CREATE SPACE basketballplayer(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。

2021 Vesoft Inc.

```
nebula> SHOW HOSTS;
+-------------+-----------+-----------+--------------+---------------------------------+------------------------+
| Host        | Port      | Status    | Leader count | Leader distribution             | Partition distribution |
+-------------+-----------+-----------+--------------+---------------------------------+------------------------+
| "storaged0" | 9779      | "ONLINE"  | 5            | "basketballplayer:5"            | "basketballplayer:5"   |
+-------------+-----------+-----------+--------------+---------------------------------+------------------------+
| "storaged1" | 9779      | "ONLINE"  | 5            | "basketballplayer:5"            | "basketballplayer:5"   |
+-------------+-----------+-----------+--------------+---------------------------------+------------------------+
| "storaged2" | 9779      | "ONLINE"  | 5            | "basketballplayer:5"            | "basketballplayer:5"   |
+-------------+-----------+-----------+--------------+---------------------------------+------------------------+
| "Total"     | __EMPTY__ | __EMPTY__ | 15           | "basketballplayer:15"           | "basketballplayer: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。

3. 选择图空间 `basketballplayer` 。

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

用户可以执行命令 `SHOW SPACES` 查看创建的图空间。

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

4.5.4 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
```


2021 Vesoft Inc.

4.5.5 插入点和边

用户可以使用 `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
```


2021 Vesoft Inc.

4.5.6 查询数据

- GO语句可以根据指定的条件遍历数据库。`GO` 语句从一个或多个点开始，沿着一条或多条边遍历，返回 `YIELD` 子句中指定的信息。
- FETCH语句可以获得点或边的属性。
- LOOKUP语句是基于索引的，和 `WHERE` 子句一起使用，查找符合特定条件的数据。
- MATCH语句是查询图数据最常用的，但是它依赖索引去匹配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" |
```


2021 Vesoft Inc.

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

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

```
nebula> $var = GO FROM "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)
```


2021 Vesoft Inc.

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

4.5.7 修改点和边

用户可以使用 `UPDATE` 语句或 `UPSERT` 语句修改现有数据。

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

> **Note**
>
> `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。

```
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"}) |
```


2021 Vesoft Inc.

```
+-----------------------------------------------+
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)
```

4.5.8 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)
```

4.5.9 索引

用户可以通过CREATE 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，除非很清楚使用索引对业务的影响。
- **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。

2021 Vesoft Inc.

nGQL语法

- 创建索引

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

- 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;
```

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

基于索引的`LOOKUP`和`MATCH`示例

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

找到标签为 `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)
```

最后更新: 2021年5月24日

2021 Vesoft Inc.

5. nGQL指南

5.1 nGQL概述

5.1.1 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，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可以做什么

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

示例数据 Basketballplayer

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

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 JTC 1/SC 32 N 3228 (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。

2021 Vesoft Inc.

本文中 nGQL 语法符号的说明如下。

符号	含义
< >	语法元素的名称。
::=	定义元素的公式。
[ ]	可选元素。
{ }	显示指定元素。
|	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);
```

原生 nGQL 和 OpenCypher 的关系

`原生 nGQL` 是由 Nebula Graph 自行创造和实现的图查询语言。OpenCypher 是由 `OpenCypher Implementers Group` 组织所开源和维护的图查询语言，最新版本为 OpenCypher 9。由于 nGQL 语言部分兼容了 OpenCypher，这个部分在本文中称为 `OpenCypher 兼容语句`。

> **Note**
>
> `nGQL 语言` = `原生 nGQL 语句` + `OpenCypher 兼容语句`

NGQL 对于 OPENCYPHER 9 的兼容性

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（不支持 `OPTIONAL MATCH` 也不支持多 `MATCH`）。
label与tag是不同的概念	label用于寻找点(点的索引)	tag用于定义点的一种类型及相应的属性，无索引功能

> **Note**
>
> 请注意OpenCypher 9和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。nGQL 不支持事务。

2021 Vesoft Inc.

是否支持TINKERPOP GREMLIN？

不支持。

是否支持 W3C 的RDF（SPARQL） 或 GRAPHQL ？

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。

nGQL 不支持 SPARQL 和 GraphQL。

找到更多NGQL的示例

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

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

```
Feature: Match seek by tag

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

  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。如果返回结果和文档不同，请提交issue通知Nebula Graph团队。
`And`	描述执行 `When` 内语句的副作用。
`@skip`	跳过这个示例。通常表示测试代码还没有准备好。

最后更新: 2021年5月24日

2021 Vesoft Inc.

5.1.2 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

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

模式除了简单地描述图中的点之外，还可以描述点的标签。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)
```


2021 Vesoft Inc.

边模式

描述一条边最简单的方法是使用箭头连接两个点。

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

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

和点一样，边也可以命名。一对方括号用于分隔箭头，变量放在两者之间。例如：

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

和点上的标签一样，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。

也可以忽略最小长度，只指定最大长度，例如：

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

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

路径变量

一系列连接的点和边称为 `路径` 。nGQL允许使用变量来命名路径，例如：

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

可以在MATCH语句中使用路径变量。

最后更新: 2021年5月24日

2021 Vesoft Inc.

5.2 数据类型

5.2.1 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` 。

最后更新: 2021年3月18日

2021 Vesoft Inc.

5.2.2 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`。

最后更新: 2021年3月19日

2021 Vesoft Inc.

5.2.3 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。

```
# 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。

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

最后更新: 2021年5月24日

2021 Vesoft Inc.

5.2.4 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` 。

在插入时间类型的属性值时，Nebula Graph会根据配置文件中 `timezone_name` 参数指定的时区，将该时间值（ `TIMESTAMP` 类型例外）转换成相应的世界协调时间（UTC）时间。在查询中返回的时间类型值为UTC时间。

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

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"));
```


2021 Vesoft Inc.

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());
```

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

最后更新: 2021年5月14日

2021 Vesoft Inc.

5.2.5 NULL

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

NULL的逻辑操作

`AND`、`OR`、`XOR` 和 `NOT` 的真值表如下。

a	b	a AND b	a OR b	a XOR	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);
```


2021 Vesoft Inc.

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

最后更新: 2021年5月24日

2021 Vesoft Inc.

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

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


2021 Vesoft Inc.

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

最后更新: 2021年5月24日

2021 Vesoft Inc.

5.2.7 映射

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

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）。

最后更新: 2021年5月14日

2021 Vesoft Inc.

5.2.8 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     |
+-------+
```


2021 Vesoft Inc.

最后更新: 2021年5月24日

2021 Vesoft Inc.

5.3 变量和复合查询

5.3.1 复合查询（子句结构）

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

Nebula Graph支持三种方式进行复合查询（或子查询）：

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

OpenCypher兼容性

在复合查询中，请**不要**混用OpenCypher 兼容语句和原生nGQL语句，例如 `MATCH ... | GO ... | YIELD ...`，混用两种语句，行为是未定义的。

- 如果使用OpenCypher 兼容语句（`MATCH`、`RETURN`、`WITH` 等），请不要使用管道符或分号组合子句。
- 如果使用原生nGQL语句（`FETCH`、`GO`、`LOOKUP` 等），必须使用管道符或分号组合子句。

复合查询不支持事务

例如一个查询由三个子查询A、B、C组成，A是一个读操作，B是一个计算操作，C是一个写操作，如果在执行过程中，任何一个操作执行失败，则整个结果是未定义的：没有回滚，而且写入的内容取决于执行程序。

> **Note**
>
> OpenCypher没有事务要求。

示例

- OpenCypher 兼容语句

```
# 子句连接多个查询。
nebula> MATCH p=(v:player{name:"Tim Duncan"})--() \
        WITH nodes(p) AS n \
        UNWIND n AS n1 \
        RETURN DISTINCT n1;
```

- 原生nGQL（分号）

```
# 只返回边。
nebula> SHOW TAGS; SHOW EDGES;

# 插入多个点。
nebula> INSERT VERTEX player(name, age) VALUES "player100":("Tim Duncan", 42); \
INSERT VERTEX player(name, age) VALUES "player101":("Tony Parker", 36); \
INSERT VERTEX player(name, age) VALUES "player102":("LaMarcus Aldridge", 33);
```

- 原生nGQL（管道符）

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


2021 Vesoft Inc.

最后更新: 2021年5月24日

2021 Vesoft Inc.

5.3.2 自定义变量

Nebula Graph允许将一条语句的结果作为自定义变量传递给另一条语句。

OpenCypher兼容性

当引用一个变量的点、边或路径，需要先给它命名。例如：

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

示例中的 `v` 就是自定义变量。

原生nGQL

原生nGQL的自定义变量可以表示为 `$var_name`， `var_name` 由字母、数字或下划线（_）构成，不允许使用其他字符。

自定义变量仅在当前执行有效，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。

用户可以在复合查询中使用自定义变量。复合查询的详细信息请参见复合查询。

> **Note**
>
> 自定义变量区分大小写。

示例

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

最后更新: 2021年5月20日

2021 Vesoft Inc.

5.3.3 引用属性

用户可以在 `WHERE` 和 `YIELD` 子句中引用点或边的属性。

> **Note**
>
> 本功能仅适用于原生nGQL。

引用点的属性

起始点

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

参数	说明
`$^`	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
`$$`	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

2021 Vesoft Inc.

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

最后更新: 2021年5月25日

2021 Vesoft Inc.

5.4 运算符

5.4.1 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`。

> **Note**
>
> 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。

> **Note**
>
> nGQL中的相等符号是 `==`，OpenCypher中的相等符号是 `=`。

```
nebula> RETURN 'A' == 'a', toUpper('A') == toUpper('a'), toLower('A') == toLower('a');
+------------+----------------------------+----------------------------+
| ("A"=="a") | (toUpper("A")==toUpper("a")) | (toLower("A")==toLower("a")) |
+------------+----------------------------+----------------------------+
```


2021 Vesoft Inc.

```
| 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 |
+--------------+------------+-------------+--------------+------------------+-----------+-------------+------------------+
| __NULL__     | __NULL__   | 0           | true         | false            | __NULL__  | __NULL__    | false            |
+--------------+------------+-------------+--------------+------------------+-----------+-------------+------------------+

nebula> WITH {name: null} AS map \
        RETURN map.name IS NOT NULL;
+----------------------+
| map.name IS NOT NULL |
+----------------------+
| false                |
```


2021 Vesoft Inc.

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

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

最后更新: 2021年5月24日

2021 Vesoft Inc.

5.4.2 布尔符

Nebula Graph支持的布尔符如下。

符号	说明
AND	逻辑与
OR	逻辑或
NOT	逻辑非
XOR	逻辑异或

对于以上运算的优先级，请参见运算优先级。

对于带有 `NULL` 的逻辑运算，请参见NULL。

历史兼容问题

在Nebula Graph 2.0中，非0数字不能转换为布尔值。

最后更新: 2021年3月24日

2021 Vesoft Inc.

5.4.3 管道符

nGQL支持使用管道符（|）将多个查询组合起来。

OpenCypher兼容性

管道符仅适用于原生nGQL。

语法

nGQL和SQL之间的一个主要区别是子查询的组成方式。

- 在SQL中，子查询是嵌套在查询语句中的。
- 在nGQL中，子查询是通过类似shell中的管道符（`|`）实现的。

示例

```
nebula> GO FROM "player100" OVER follow \
YIELD follow._dst AS dstid, $$.player.name AS Name | \
GO FROM $-.dstid OVER follow;

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

用户可以使用 `YIELD` 显式声明需要返回的结果，如果不使用 `YIELD`，默认返回目标点ID。

必须在 `YIELD` 子句中为需要的返回结果设置别名，才能在管道符右侧使用引用符 `$-`，例如示例中的 `$-.dstid` 。

性能提示

Nebula Graph book 中的管道对性能有影响，以 `A | B` 为例，体现在以下几个方面：

1. 管道是同步操作。也即需要管道之前的子句 `A` 执行完毕后，数据才能整体进入管道子句。
2. 管道本身是需要序列化和反序列化的，这个是单线程执行的。
3. 如果 `A` 发大量数据给 `|`，整个查询请求的总体时延可能会非常大。此时可以尝试拆分这个语句：
   a. 应用程发送 `A`，
   b. 将收到的返回结果在应用程序拆分，
   c. 并发发送给多个 graphd，
   d. 每个 graphd 执行部分 B。

   这样通常比单个graphd 执行完整地 `A | B` 要快很多。

最后更新: 2021年5月24日

2021 Vesoft Inc.

5.4.4 引用符

nGQL提供引用符来表示 `WHERE` 和 `YIELD` 子句中的属性，或者复合查询中管道符之前的语句输出结果。

OpenCypher兼容性

引用符仅适用于原生nGQL。

引用符列表

引用符	说明
`$^`	引用起始点。更多信息请参见引用属性。
`$$`	引用目的点。更多信息请参见引用属性。
`$-`	引用复合查询中管道符之前的语句输出结果。更多信息请参见管道符。

示例

```
# 返回起始点和目的点的年龄。
nebula> GO FROM "player100" OVER follow YIELD $^.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"   |
+-----------------+-----------+
```

最后更新: 2021年5月24日

2021 Vesoft Inc.

5.4.5 集合运算符

合并多个请求时，可以使用集合运算符，包括 `UNION` 、 `UNION ALL` 、 `INTERSECT` 和 `MINUS` 。

所有集合运算符的优先级相同，如果一个nGQL语句中有多个集合运算符，Nebula Graph会从左到右进行计算，除非用括号指定顺序。

OpenCypher兼容性

集合运算符仅适用于原生nGQL。

UNION、UNION DISTINCT、UNION ALL

```
<left> UNION [DISTINCT | ALL] <right> [ UNION [DISTINCT | ALL] <right> ...]
```

- 运算符 `UNION DISTINCT` （或使用缩写 `UNION` ）返回两个集合A和B的并集，不包含重复的元素。
- 运算符 `UNION ALL` 返回两个集合A和B的并集，包含重复的元素。
- `left` 和 `right` 必须有相同数量的列和数据类型。如果需要转换数据类型，请参见类型转换。

示例

```
# 返回两个查询结果的并集，不包含重复的元素。
nebula> GO FROM "player102" OVER follow \
    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。

示例

2021 Vesoft Inc.

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

```
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。

2021 Vesoft Inc.

最后更新: 2021年5月24日

2021 Vesoft Inc.

5.4.6 字符串运算符

Nebula Graph支持使用字符串运算符进行连接、搜索、匹配运算。支持的运算符如下。

名称	说明
+	连接字符串。
CONTAINS	在字符串中执行搜索。
(NOT) IN	字符串是否匹配某个值。
(NOT) STARTS WITH	在字符串的开头执行匹配。
(NOT) ENDS WITH	在字符串的结尾执行匹配。
正则表达式	通过正则表达式匹配字符串。

Note

所有搜索或匹配都区分大小写。

示例

+

```
nebula> RETURN 'a' + 'b';
+-------+
| (a+b) |
+-------+
| "ab"  |
+-------+
nebula> UNWIND 'a' AS a UNWIND 'b' AS b RETURN a + b;
+-------+
| (a+b) |
+-------+
| "ab"  |
+-------+
```

CONTAINS

`CONTAINS` 要求待运算的左右两边都是字符串类型。

```
nebula> MATCH (s:player)-[e:serve]->(t:team) WHERE id(s) == "player101" \
    AND t.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"]) |
+----------------+-----------------------------------+-------------------------------+
```


2021 Vesoft Inc.

```
| 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

> **Note**
>
> 当前仅OpenCypher 兼容语句（`MATCH`、`WITH` 等）支持正则表达式，原生nGQL语句（`FETCH`、`GO`、`LOOKUP` 等）不支持正则表达式。

Nebula Graph支持使用正则表达式进行过滤，正则表达式的语法是继承自 `std::regex`，用户可以使用语法 `=~ '<regexp>'` 进行正则表达式匹配配。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" |
+---------------+
```

最后更新: 2021年5月24日

2021 Vesoft Inc.

5.4.7 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" |
+--------+
```

最后更新: 2021年3月23日

2021 Vesoft Inc.

5.4.8 运算符优先级

nGQL运算符的优先级从高到低排列如下（同一行的运算符优先级相同）：

- `-`（负数）
- `!`、`NOT`
- `*`、`/`、`%`
- `-`、`+`
- `==`、`>=`、`>`、`<=`、`<`、`<>`、`!=`
- `AND`
- `OR`、`XOR`
- `=`（赋值）

如果表达式中有相同优先级的运算符，运算是从左到右进行，只有赋值操作是例外（从右到左运算）。

运算符的优先级决定运算的顺序，要显式修改运算顺序，可以使用圆括号。

示例

```
nebula> RETURN 2+3*5;
+-----------+
| (2+(3*5)) |
+-----------+
| 17        |
+-----------+

nebula> RETURN (2+3)*5;
+-----------+
| ((2+3)*5) |
+-----------+
| 25        |
+-----------+
```

OpenCypher兼容性

在OpenCypher中，比较操作可以任意连接，例如 `x < y <= z` 等价于 `x < y AND y <= z`。

在nGQL中，`x < y <= z` 等价于 `(x < y) <= z`，`(x < y)` 的结果是一个布尔值，再将布尔值和 `z` 比较，最终结果是 `NULL`。

最后更新: 2021年5月24日

2021 Vesoft Inc.

5.5 函数和表达式

2021 Vesoft Inc.

5.5.1 内置数学函数

Nebula Graph支持以下内置数学函数。

2021 Vesoft Inc.

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

2021 Vesoft Inc.

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

Note

如果参数为 `NULL`，则输出结果是未定义的。

最后更新: 2021年5月14日

2021 Vesoft Inc.

5.5.2 内置字符串函数

Nebula Graph支持以下内置字符串函数。

> **Note**
>
> 和SQL一样，nGQL的字符索引（位置）从 `1` 开始。但是C语言的字符索引是从 `0` 开始的。

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

> **Note**
>
> 如果参数为 `NULL`，则输出结果是未定义的。

2021 Vesoft Inc.

`substr()`和`substring()`的返回说明

- 字符索引（位置）从 `0` 开始。
- 如果 `pos` 为0，则返回整个字符串。
- 如果 `pos` 大于最大字符索引，则返回空字符串。
- 如果 `pos` 是负数，则返回 `BAD_DATA` 。
- 如果省略 `count` ，则返回从 `pos` 位置开始到字符串末尾的子字符串。
- 如果 `count` 为0，则返回空字符串。
- 使用 `NULL` 作为任何参数会出现错误。

> **openCypher兼容性**
>
> - 在openCypher中，如果字符串 `a` 为 `null` ，会返回 `null` 。
> - 在openCypher中，如果 `pos` 为0，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。

最后更新: 2021年5月27日

2021 Vesoft Inc.

5.5.3 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 |
+------------+------------+--------------+-------------------------+
```

最后更新: 2021年5月24日

2021 Vesoft Inc.

5.5.4 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"}) |
+--------------------------------------------------+----------------------------------+
```

最后更新: 2021年3月24日

2021 Vesoft Inc.

5.5.5 CASE表达式

`CASE` 表达式使用条件来过滤nGQL查询语句的结果，常用于 `YIELD` 和 `RETURN` 子句中。和OpenCypher一样，nGQL提供两种形式的 `CASE` 表达式：简单形式和通用形式。

`CASE` 表达式会遍历所有条件，并在满足第一个条件时停止读取后续条件，然后返回结果。如果不满足任何条件，将通过 `ELSE` 子句返回结果。如果没有 `ELSE` 子句且不满足任何条件，则返回 `NULL` 。

简单形式

语法

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

> **Caution**
>
> `CASE` 表达式一定要用 `END` 结尾。

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

示例

```
nebula> RETURN \
        CASE 2+3 \
        WHEN 4 THEN 0 \
        WHEN 5 THEN 1 \
        ELSE -1 \
        END \
        AS result;
+--------+
| result |
+--------+
| 1      |
+--------+
```

```
nebula> GO FROM "player100" OVER follow \
        YIELD $$.player.name AS Name, \
        CASE $$.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>
```


2021 Vesoft Inc.

```
[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，用户需要了解它们的差异。请参见如下示例：

```
nebula> GO FROM "player100" OVER follow \
        YIELD $$.player.name AS Name, $$.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` 。

最后更新: 2021年5月24日

2021 Vesoft Inc.

5.5.6 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。

Note

如果参数为 `NULL`，则输出结果是未定义的。

示例

```
nebula> WITH [NULL, 4923, 'abc', 521, 487] AS ids \
        RETURN reverse(ids), tail(ids), head(ids), last(ids), coalesce(ids);
+----------------------------------+-------------------------+------------+------------+---------------+
| 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);
+------------------------------
```

最后更新: 2021年5月8日

2021 Vesoft Inc.

5.5.7 count函数

`count()` 函数可以计数指定的值或行数。

- （原生nGQL）用户可以同时使用 `count()` 和 `GROUP BY` 对指定的值进行分组和计数，再使用 `YIELD` 返回结果。
- （OpenCypher兼容语句）用户可以使用 `count()` 对指定的值进行计数，再使用 `RETURN` 返回结果。不需要使用 `GROUP BY` 。

语法

```
count({expr | *})
```

- count(*)返回总行数（包括NULL）。
- count(expr)返回满足表达式的非空值的总数。
- `count()` 和 `size()` 是不同的。

示例

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

```
# 返回player101 follow的人，以及follow player101的人，即双向查询。
nebula> GO FROM "player101" OVER follow BIDIRECT \
        YIELD $$.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。

因为测试数据集 `basketballplayer` 中没有重复的姓名， `COUNT(*)` 列中数字 `2` 表示该行的人和 `player101` 是互相 `follow` 的关系。

```
# 方法一：统计数据库中的年龄分布情况。
nebula> LOOKUP ON player \
        YIELD player.age As playerage \
        | GROUP BY $-.playerage \
        YIELD $-.playerage as age, count(*) AS number \
        | ORDER BY number DESC, age DESC;
+-----+--------+
| age | number |
+-----+--------+
| 34  | 4      |
+-----+--------+
| 33  | 4      |
+-----+--------+
| 30  | 4      |
+-----+--------+
| 29  | 4      |
+-----+--------+
| 38  | 3      |
+-----+--------+
```


2021 Vesoft Inc.

```
...

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

最后更新: 2021年5月20日

2021 Vesoft Inc.

5.5.8 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"]                                        |
+-----+-------------------------------------------------------------------------+
...
```

最后更新: 2021年3月24日

2021 Vesoft Inc.

5.5.9 reduce函数

`reduce()` 将表达式逐个应用于列表中的元素，然后和累加器中的当前结果累加，最后返回完整结果。该函数将遍历给定列表中的每个元素 `e`，在 `e` 上运行表达式并和累加器的当前结果累加，将新的结果存储在累加器中。这个函数类似于函数式语言（如Lisp和Scala）中的fold或reduce方法。

OpenCypher兼容性

在OpenCypher中，`reduce()` 函数没有定义。nGQL使用了Cypher方式实现 `reduce()` 函数。

语法

```
reduce(<accumulator> = <initial>, <variable> IN <list> | <expression>)
```

参数	说明
accumulator	在遍历列表时保存累加结果。
initial	为 `accumulator` 提供初始值的表达式或值。
variable	为列表引入一个变量，决定使用列表中的哪个元素。
list	列表或列表表达式。
expression	该表达式将对列表中的每个元素运行一次，并将结果累加至 `accumulator` 。

Note

返回值的类型取决于提供的参数，以及表达式的语义。

示例

```
nebula> RETURN reduce(totalNum = 10, n IN range(1, 3) | totalNum + n) AS r;
+----+
| r  |
+----+
| 16 |
+----+

nebula> RETURN reduce(totalNum = -4 * 5, n IN [1, 2] | totalNum + n * 2) AS r;
+-----+
| r   |
+-----+
| -14 |
+-----+

nebula> MATCH p = (n:player{name:"LeBron James"})<-[:follow]-(m) \
        RETURN  nodes(p)[0].age AS src1,  nodes(p)[1].age AS dst2,  \
        reduce(totalAge = 100, n IN nodes(p) | totalAge + n.age) AS sum;
+------+------+-----+
| src1 | dst2 | sum |
+------+------+-----+
| 34   | 31   | 165 |
+------+------+-----+
| 34   | 29   | 163 |
+------+------+-----+
| 34   | 33   | 167 |
+------+------+-----+
| 34   | 26   | 160 |
+------+------+-----+
| 34   | 34   | 168 |
+------+------+-----+
| 34   | 37   | 171 |
+------+------+-----+

nebula> LOOKUP ON player WHERE player.name == "Tony Parker" \
        |  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     |
```


2021 Vesoft Inc.

```
+---------------------+-----+--------+
| "LaMarcus Aldridge" | 33  | 90     |
+---------------------+-----+--------+
| "Manu Ginobili"     | 41  | 95     |
+---------------------+-----+--------+
```

最后更新: 2021年5月24日

2021 Vesoft Inc.

5.5.10 hash函数

`hash()` 函数返回参数的哈希值。其参数可以是数字、字符串、列表、布尔值、NULL等类型的值，或者计算结果为这些类型的表达式。

`hash()` 函数采用MurmurHash2算法，种子（seed）为 `0xc70f6907UL` 。用户可以在 `MurmurHash2.h` 中查看其源代码。

在Java中的调用方式如下：

```
MurmurHash2.hash64("to_be_hashed".getBytes(),"to_be_hashed".getBytes().length, 0xc70f6907)
```

历史版本兼容性

nGQL 1.0不支持字符串类型的VID，一种常用的处理方式是用hash函数获取字符串的哈希值，然后将该值设置为VID。但nGQL 2.0同时支持了字符串和整数类型的VID，所以无需再使用这种方式设置VID。

计算数字的哈希值

```
nebula> YIELD hash(-123);
+--------------+
| hash(-(123)) |
+--------------+
| -123         |
+--------------+
```

计算字符串的哈希值

```
nebula> YIELD hash("to_be_hashed");
+----------------------+
| hash(to_be_hashed)   |
+----------------------+
| -1098333533029391540 |
+----------------------+
```

计算列表的哈希值

```
nebula> YIELD hash([1,2,3]);
+----------------+
| hash([1,2,3])  |
+----------------+
| 11093822460243 |
+----------------+
```

计算布尔值的哈希值

```
nebula> YIELD hash(true);
+------------+
| hash(true) |
+------------+
| 1          |
+------------+

nebula> YIELD hash(false);
+-------------+
| hash(false) |
+-------------+
| 0           |
+-------------+
```

计算NULL的哈希值

```
nebula> YIELD hash(NULL);
+------------+
| hash(NULL) |
+------------+
| -1         |
+------------+
```


2021 Vesoft Inc.

计算表达式的哈希值

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

最后更新: 2021年5月24日

2021 Vesoft Inc.

5.5.11 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`。

> **Note**
>
> 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 |
+----------------+-------------------+-------+
```


2021 Vesoft Inc.

```
| "LeBron James" | "Dejounte Murray" | false |
+---------------+------------------+-------+
| "LeBron James" | "Chris Paul"      | true  |
+---------------+------------------+-------+
| "LeBron James" | "Kyrie Irving"    | true  |
+---------------+------------------+-------+
| "LeBron James" | "Carmelo Anthony" | true  |
+---------------+------------------+-------+
| "LeBron James" | "Dwyane Wade"     | false |
+---------------+------------------+-------+

nebula> MATCH p = (n:player{name:"LeBron James"})-[:follow]->(m) \
        RETURN single(n IN nodes(p) WHERE n.age > 40) AS b;
+------+
| b    |
+------+
| true |
+------+

nebula> MATCH (n:player) \
        RETURN exists(n.id), n IS NOT NULL;
+-------------+--------------+
| exists(n.id) | n IS NOT NULL |
+-------------+--------------+
| false       | true         |
+-------------+--------------+
...
```

最后更新: 2021年5月24日

2021 Vesoft Inc.

5.6 通用查询语句

5.6.1 MATCH

`MATCH` 语句提供基于模式（pattern）匹配的搜索功能。

一个 `MATCH` 语句定义了一个搜索模式，用该模式匹配存储在Nebula Graph中的数据，然后用 `RETURN` 子句检索数据。

本文示例使用测试数据集basketballplayer进行演示。

语法

与 `GO` 或 `LOOKUP` 等其他查询语句相比， `MATCH` 的语法更灵活。 `MATCH` 语法可以概括如下：

```
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，或者在 `WHERE` 子句中用id()函数指定了特定点的VID**。如何创建索引，请参见创建原生索引。
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不支持遍历所有点和边，例如 `MATCH (v) RETURN v` 。

但是，建立相应Tag的索引后，可以遍历对应Tag的所有点，例如 `MATCH (v:T1) 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，或者其中指定了VID。如果需要创建索引，但是已经有相关的点、边或属性，用户必须在创建索引后重建索引，索引才能生效。

> **Caution**
>
> 索引会导致写性能大幅降低（降低90%甚至更多）。请**不要随意**在生产环境中使用索引，除非很清楚使用索引对业务的影响。

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


2021 Vesoft Inc.

```
| 2             | "storaged1"         | "FINISHED" | 1607073046 | 1607073046 |
+---------------+---------------------+------------+------------+------------+
```

匹配点

用户可以在一对括号中使用自定义变量来表示模式中的点。例如 `(v)` 。

匹配标签

> **Note**
>
> 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，用户无法基于该标签执行 `MATCH` 语句。

用户可以在点的右侧用 `:<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

用户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

用户可以使用点ID去匹配点。 `id()` 函数可以检索点的ID。

```
nebula> MATCH (v) WHERE id(v) == 'player101' RETURN v;
+------------------------------------------------+
| v                                              |
+------------------------------------------------+
| (player101) player.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})      |
+----------------------------------------------------------+
```


2021 Vesoft Inc.

匹配连接的点

用户可以使用 `--` 符号表示两个方向的边，并匹配这些边连接的点。

> **Note**
>
> 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"       |
+---------------------+
...
```

用户可以在 `--` 符号上增加 `<` 或 `>` 符号指定边的方向。

```
# -->表示边从v开始，指向v2。对于点v来说是出边，对于点v2来说是入边。
nebula> MATCH (v:player{name:"Tim Duncan"})-->(v2) \
        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"     |
+---------------------+
...
```


2021 Vesoft Inc.

匹配路径

连接起来的点和边构成了路径。用户可以使用自定义变量命名路径。

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-->(v2) \
        RETURN p;
+-----------------------------------------+
| p                                       |
+-----------------------------------------+
| <("player100" :player{age: 42, name: "Tim Duncan"})-[: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，用户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，用户可以用`:<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}]   |
+-------------------------------------------------------------------------+
```

用户可以用`{<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}]   |
+-------------------------------------------------------------------------+
```


2021 Vesoft Inc.

匹配多条边

用户可以扩展模式，匹配路径中的多条边。

```
nebula> MATCH (v:player{name:"Tim Duncan"})-[]->(v2)<-[e:serve]-(v3) \
        RETURN v2, v3;
+-----------------------------------+-------------------------------------------------------+
| v2                                | v3                                                    |
+-----------------------------------+-------------------------------------------------------+
| ("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

用户可以在模式中使用 `:<edge_type>*<hop>` 匹配定长路径。 `hop` 必须是一个非负整数。

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[e:follow*2]->(v2) \
        RETURN DISTINCT v2 AS Friends;
+-------------------------------------------------------+
| Friends                                               |
+-------------------------------------------------------+
| ("player100" :player{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

用户可以在模式中使用 `:<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"}) |
+-------------------------------------------------------+
```

用户可以使用 `DISTINCT` 关键字聚合重复结果。

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[e:follow*1..3]->(v2:player) \
        RETURN DISTINCT v2 AS Friends, count(v2);
+-------------------------------------------------------+-----------+
| Friends                                               | COUNT(v2) |
+-------------------------------------------------------+-----------+
| ("player125" :player{age: 41, name: "Manu Ginobili"})    | 3         |
+-------------------------------------------------------+-----------+
```


2021 Vesoft Inc.

```
| ("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

用户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" |
+------------+
```


2021 Vesoft Inc.

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。

2021 Vesoft Inc.

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


2021 Vesoft Inc.

最后更新: 2021年5月27日

2021 Vesoft Inc.

5.6.2 LOOKUP

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

- 根据 `WHERE` 子句搜索特定数据。
- 通过标签列出点: 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

本文操作仅适用于原生nGQL。

前提条件

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

> **Caution**
>
> 索引会导致写性能大幅降低（降低90%甚至更多）。请不要随意在生产环境中使用索引，除非很清楚使用索引对业务的影响。

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。
- `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;
```


2021 Vesoft Inc.

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

通过标签列出所有的对应的点/通过边类型列出边

如果需要通过标签列出所有的点，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。

2021 Vesoft Inc.

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。

- 查找所有标签为 `player` 的点 VID。

```
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);

# 列出所有的 player. 类似于 MATCH (n:player) RETURN id(n) /*, n */

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

- 查找边类型为 `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);

# 列出所有的 like 边。类似于 MATCH (s)-[e:like]->(d) RETURN id(s), rank(e), id(d) /*, type(e) */

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

统计点或边

统计标签为 `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           |
+-------------+
```

> **用户也可以使用 `show-stats` 命令实现类似的功能。**

最后更新: 2021年5月24日

2021 Vesoft Inc.

5.6.3 GO

`GO` 用指定的过滤条件遍历图，并返回结果。

OpenCypher兼容性

本文操作仅适用于原生nGQL。

语法

```
GO [[<M> TO] <N> STEPS ] FROM <vertex_list>
OVER <edge_type_list> [{REVERSELY | BIDIRECT}]
[ WHERE <conditions> ]
[YIELD [DISTINCT] <return_list>]
[| 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。
- `<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。用户可以在起始点、目的点和边使用 `WHERE` 子句，还可以结合 `AND` 、 `OR` 、 `NOT` 、 `XOR` 一起使用。详情请参见WHERE。

  > **Note**
  >
  > 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。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。

  > **Note**
  >
  > 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。
- `GROUP BY` ：根据指定属性的值将输出分组。详情请参见GROUP BY。

示例

2021 Vesoft Inc.

```
# 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)  \
```


2021 Vesoft Inc.

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


2021 Vesoft Inc.

最后更新: 2021年5月24日

2021 Vesoft Inc.

5.6.4 FETCH

`FETCH` 可以获取指定点或边的属性值。

OpenCypher兼容性

本文操作仅适用于原生nGQL。

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` 。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);
```


2021 Vesoft Inc.

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

用户可以在 `FETCH` 语句中组合多个标签和多个点。

```
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。

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


2021 Vesoft Inc.

```
| "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` 与原生nGQL结合使用是一种常见的方式，例如和 `GO` 一起。

```
# 返回从点player101开始的follow边的degree值。
nebula> GO FROM "player101" OVER follow \
        YIELD 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           |
+-------------+-------------+-------------+--------------+
```

用户也可以通过自定义变量构建类似的查询。

```
nebula> $var = GO FROM "player101" OVER follow \
        YIELD 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）。

2021 Vesoft Inc.

最后更新: 2021年5月24日

2021 Vesoft Inc.

5.6.5 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> 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);
+------------------
```

最后更新: 2021年4月20日

2021 Vesoft Inc.

5.6.6 SHOW

SHOW CHARSET

`SHOW CHARSET` 语句显示当前的字符集。

目前可用的字符集为 `utf8` 和 `utf8mb4` 。默认字符集为 `utf8` 。Nebula Graph扩展 `uft8` 支持四字节字符，因此 `utf8` 和 `utf8mb4` 是等价的。

语法

```
SHOW CHARSET;
```

示例

```
nebula> SHOW CHARSET;
+---------+-----------------+-------------------+--------+
| Charset | Description     | Default collation | Maxlen |
+---------+-----------------+-------------------+--------+
| "utf8"  | "UTF-8 Unicode" | "utf8_bin"        | 4      |
+---------+-----------------+-------------------+--------+
```

参数	说明
`Charset`	字符集名称。
`Description`	字符集说明。
`Default collation`	默认排序规则。
`Maxlen`	存储一个字符所需的最大字节数。

最后更新: 2021年3月29日

2021 Vesoft Inc.

SHOW COLLATION

`SHOW COLLATION` 语句显示当前的排序规则。

目前可用的排序规则为 `utf8_bin` 、 `utf8_general_ci` 、 `utf8mb4_bin` 和 `utf8mb4_general_ci` 。

- 当字符集为 `utf8` ，默认排序规则为 `utf8_bin` 。
- 当字符集为 `utf8mb4` ，默认排序规则为 `utf8mb4_bin` 。
- `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。

最后更新: 2021年3月29日

2021 Vesoft Inc.

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。

语法

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

示例

```
nebula> SHOW CREATE SPACE basketballplayer;
+--------------------+-------------------
```

SHOW CREATE TAG/EDGE

`SHOW CREATE TAG` 语句显示指定标签的基本信息。标签的更多详细信息，请参见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。

语法

```
SHOW CREATE {TAG <tag_name> | EDGE <edge_name>};
```

示例

```
nebula> SHOW CREATE TAG player;
+----------+----------------------------------+
| Tag      | Create Tag                       |
+----------+----------------------------------+
| "player" | "CREATE TAG `player` (           |
|          |  `name` string NULL,             |
|          |  `age` int64 NULL                |
|          | ) ttl_duration = 0, ttl_col = "" |
+----------+----------------------------------+

nebula> SHOW CREATE EDGE follow;
+----------+----------------------------------+
| Edge     | Create Edge                      |
+----------+----------------------------------+
| "follow" | "CREATE EDGE `follow` (          |
|          |  `degree` int64 NULL             |
|          | ) ttl_duration = 0, ttl_col = """ |
+----------+----------------------------------+
```

最后更新: 2021年3月29日

2021 Vesoft Inc.

SHOW HOSTS

`SHOW HOSTS` 语句可以显示Graph、Storage、Meta服务主机信息。

语法

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

不添加服务名，直接使用 `SHOW HOSTS`，会显示Storage服务主机信息，以及leader总数、leader分布和分片分布。

示例

```
nebula> SHOW HOSTS;
+-------------+-------+----------+--------------+---------------------------------+-----------------------------+
| Host        | Port  | Status   | Leader count | Leader distribution             | Partition distribution      |
+-------------+-------+----------+--------------+---------------------------------+-----------------------------+
| "storaged0" | 9779  | "ONLINE" | 8            | "docs:5, basketballplayer:3"    | "docs:5, basketballplayer:3"|
+-------------+-------+----------+--------------+---------------------------------+-----------------------------+
| "storaged1" | 9779  | "ONLINE" | 9            | "basketballplayer:4, docs:5"    | "docs:5, basketballplayer:4"|
+-------------+-------+----------+--------------+---------------------------------+-----------------------------+
| "storaged2" | 9779  | "ONLINE" | 8            | "basketballplayer:3, docs:5"    | "docs:5, basketballplayer: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"    |
```

最后更新: 2021年5月26日

2021 Vesoft Inc.

SHOW INDEX STATUS

`SHOW INDEX STATUS` 语句显示重建原生索引的作业状态，以便确定重建索引是否成功。

语法

```
SHOW {TAG | EDGE} INDEX STATUS;
```

示例

```
nebula> SHOW TAG INDEX STATUS;
+----------------+--------------+
| Name           | Index Status |
+----------------+--------------+
| "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
- REBUILD NATIVE INDEX

最后更新: 2021年3月29日

2021 Vesoft Inc.

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

最后更新: 2021年3月29日

2021 Vesoft Inc.

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

最后更新: 2021年3月29日

2021 Vesoft Inc.

SHOW ROLES

`SHOW ROLES` 语句显示分配给用户的角色信息。

根据登录的用户角色，返回的结果也有所不同：

- 如果登录的用户角色是 `GOD`，或者有权访问该图空间的 `ADMIN`，则返回该图空间内除 `GOD` 之外的所有用户角色信息。
- 如果登录的用户角色是有权访问该图空间 `DBA`、 `USER` 或 `GUEST`，则返回自身的角色信息。
- 如果登录的用户角色没有权限访问该图空间，则返回权限错误。

关于角色的详情请参见内置角色权限。

语法

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

示例

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

最后更新: 2021年4月13日

2021 Vesoft Inc.

SHOW SNAPSHOTS

`SHOW SNAPSHOTS` 语句显示所有快照信息。

快照的使用方式请参见管理快照。

角色要求

只有 `GOD` 角色的用户（即 `root` ）才能执行 `SHOW SNAPSHOTS` 语句。

语法

```
SHOW SNAPSHOTS;
```

示例

```
nebula> SHOW SNAPSHOTS;
+--------------------------------+---------+-------------------------------------------------------+
| Name                           | Status  | Hosts                                                 |
+--------------------------------+---------+-------------------------------------------------------+
| "SNAPSHOT_2020_12_16_11_13_55" | "VALID" | "storaged0:9779, storaged1:9779, storaged2:9779"      |
+--------------------------------+---------+-------------------------------------------------------+
| "SNAPSHOT_2020_12_16_11_14_10" | "VALID" | "storaged0:9779, storaged1:9779, storaged2:9779"      |
+--------------------------------+---------+-------------------------------------------------------+
```

最后更新: 2021年3月29日

2021 Vesoft Inc.

SHOW SPACES

`SHOW SPACES` 语句显示现存的图空间。

如何创建图空间，请参见CREATE SPACE。

语法

```
SHOW SPACES;
```

示例

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

最后更新: 2021年4月13日

2021 Vesoft Inc.

SHOW STATS

`SHOW STATS` 语句显示最近 `STATS` 作业收集的图空间统计信息。

图空间统计信息包含：

- 点的总数
- 边的总数
- 每个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。

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

语法

```
SHOW STATS;
```

示例

```
# 选择图空间。
nebula> USE basketballplayer;

# 执行SUBMIT JOB STATS。
nebula> SUBMIT JOB STATS;
+------------+
| New Job Id |
+------------+
| 98         |
+------------+

# 确认作业执行成功。
nebula> SHOW JOB 98;
+----------------+---------------+------------+------------+------------+
| Job Id(TaskId) | Command(Dest) | Status     | Start Time | Stop Time  |
+----------------+---------------+------------+------------+------------+
| 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   |
+---------+------------+-------+
```

最后更新: 2021年5月14日

2021 Vesoft Inc.

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

最后更新: 2021年3月29日

2021 Vesoft Inc.

SHOW USERS

`SHOW USERS` 语句显示用户信息。

角色要求

只有 `GOD` 角色的用户（即 `root` ）才能执行 `SHOW USERS` 语句。

语法

```
SHOW USERS;
```

示例

```
nebula> SHOW USERS;
+---------+
| Account |
+---------+
| "root"  |
+---------+
| "user1" |
+---------+
```

最后更新: 2021年3月29日

2021 Vesoft Inc.

5.7 子句和选项

5.7.1 GROUP BY

`GROUP BY` 子句可以用于聚合数据。

OpenCypher兼容性

本文操作仅适用于原生nGQL。

用户也可以使用OpenCypher兼容语句的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          |
+--------------------+------------+
```


2021 Vesoft Inc.

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。

最后更新: 2021年5月24日

2021 Vesoft Inc.

5.7.2 LIMIT

`LIMIT` 子句限制输出结果的行数。

- 在原生nGQL中，必须使用管道符（|），可以忽略偏移量。
- 在OpenCypher兼容语句中，不允许使用管道符，可以使用 `SKIP` 指明偏移量。

> **Note**
>
> 在原生nGQL或OpenCypher兼容语句中使用 `LIMIT` 时，使用 `ORDER BY` 子句限制输出顺序非常重要，否则会输出一个不可预知的子集。

原生nGQL语法

在原生nGQL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兼容语句

```
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。

> **Note**
>
> 两个整数组成的分数表达式会自动向下取整。例如 `8/6` 向下取整为1。

2021 Vesoft Inc.

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

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

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

用户也可以同时使用 `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  |
+-----------------+-----+
```

最后更新: 2021年5月24日

2021 Vesoft Inc.

5.7.3 ORDER BY

`ORDER BY` 子句指定输出结果的排序规则。

- 在原生nGQL中，必须在 `YIELD` 子句之后使用管道符（|）和 `ORDER BY` 子句。
- 在OpenCypher兼容语句中，不允许使用管道符。在 `RETURN` 子句之后使用 `ORDER BY` 子句。

排序规则分为如下两种：

- `ASC` （默认）：升序。
- `DESC`：降序。

原生nGQL语法

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

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兼容语句

```
<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"      |
+-----+-------------------+
...
```


2021 Vesoft Inc.

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

最后更新: 2021年5月20日

2021 Vesoft Inc.

5.7.4 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兼容语句中，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兼容语句。关于原生nGQL如何定义输出结果，请参见 `YIELD` 。

`RETURN` 不支持如下openCypher功能：

- 使用不在英文字母表中的字符作为变量名。例如：

```
MATCH (`点1`:player) \
RETURN `点1`;
```

- 设置一个模式，并返回该模式匹配的所有元素。例如：

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

历史版本兼容性

- 在nGQL 1.x中， `RETURN` 适用于原生nGQL，语法为 `RETURN <var_ref> IF <var_ref> IS NOT NULL` 。
- 在nGQL 2.0中， `RETURN` 不适用于原生nGQL。

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}]       |
+-----------------------------------------------------------------------------+
...
```


2021 Vesoft Inc.

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


2021 Vesoft Inc.

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

最后更新: 2021年5月20日

2021 Vesoft Inc.

5.7.5 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

本文操作仅适用于原生nGQL。

注意事项

- 不能修改带有TTL选项的属性的Schema。
- 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过程中回收硬盘空间。

> **Note**
>
> 如果关闭TTL选项，上一次Compaction之后的过期数据将可以被查询到。

TTL选项

nGQL支持的TTL选项如下。

选项	说明
`ttl_col`	指定要设置存活时间的属性。属性的数据类型必须是 `int` 或者 `timestamp` 。
`ttl_duration`	指定时间戳差值，单位：秒。时间戳差值必须为64位非负整数。属性值和时间戳差值之和如果小于当前时间戳，属性就会过期。如果 `ttl_duration` 为 `0` ，属性永不过期。

使用TTl选项

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;
```


2021 Vesoft Inc.

```
# 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和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，属性永不过期，且属性的Schema无法修改。

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

最后更新: 2021年5月24日

2021 Vesoft Inc.

5.7.6 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：

- 原生nGQL，例如 `GO` 和 `LOOKUP` 语句。
- OpenCypher兼容语句，例如 `MATCH` 和 `WITH` 语句。

OpenCypher兼容性

- 不支持在模式中使用 `WHERE` 子句，例如 `WHERE (v)-->(v2)` 。
- 过滤rank是原生nGQL功能。只支持在原生nGQL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

> **Note**
>
> 下文示例中的 `$$` 、 `$^` 等是引用符号，详情请参见引用符。

用布尔运算符定义条件

在 `WHERE` 子句中使用布尔运算符 `NOT` 、 `AND` 、 `OR` 和 `XOR` 定义条件。关于运算符的优先级，请参见运算符优先级。

```
nebula> MATCH (v:player) \
        WHERE v.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" |
+-------------+
```


2021 Vesoft Inc.

过滤属性

在 `WHERE` 子句中使用点或边的属性定义条件。

- 过滤点属性：

```
nebula> MATCH (v:player)-[e]->(v2) \
        WHERE v2.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    |
+-------------------------+-------+
```


2021 Vesoft Inc.

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


2021 Vesoft Inc.

```
| "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

用户可以结合布尔运算符 `NOT` 一起使用，否定字符串匹配条件。

```
nebula> MATCH (v:player) \
        WHERE NOT v.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  |
```


2021 Vesoft Inc.

```
+---------------------+-----+
| "Paul George"       | 28  |
+---------------------+-----+
| "Ricky Rubio"       | 28  |
+---------------------+-----+
...
```

最后更新: 2021年5月25日

2021 Vesoft Inc.

5.7.7 YIELD

`YIELD` 定义nGQL查询的输出结果。

`YIELD` 可以引导子句或语句：

- `YIELD` 子句可以用于原生nGQL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

本文操作仅适用于原生nGQL。关于openCypher兼容语句如何定义输出结果，请参见 `RETURN` 。

`YIELD` 在nGQL和OpenCypher中有不同的函数：

- 在OpenCypher中， `YIELD` 用于在 `CALL[...YIELD]` 子句中指定过程调用的输出。

  > **Note**
  >
  > nGQL不支持 `CALL[...YIELD]` 。

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

> **Note**
>
> 下文示例中的 `$$` 、 `$-` 等是引用符号，详情请参见引用符。

YIELD子句

语法

```
YIELD [DISTINCT] <col> [AS <alias>] [, <col> [AS <alias>] ...];
```

参数	说明
`DISTINCT`	聚合输出结果，返回去重后的结果集。
`col`	要返回的字段。如果没有为字段设置别名，返回结果中的列名为 `col` 。
`alias`	`col` 的别名。使用关键字 `AS` 进行设置，设置后返回结果中的列名为该别名。

使用YIELD子句

- `GO` 语句中使用 `YIELD` ：

```
nebula> GO FROM "player100" OVER follow \
        YIELD $$.player.name AS Friend, $$.player.age AS Age;
+-----------------+-----+
| Friend          | Age |
+-----------------+-----+
| "Tony Parker"   | 36  |
+-----------------+-----+
```


2021 Vesoft Inc.

```
| "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` 。

复合查询中使用YIELD语句

在复合查询中， `YIELD` 语句可以接收、过滤、修改之前语句的结果集，然后输出。

```
# 查找player100关注的player，并计算他们的平均年龄。
nebula> GO FROM "player100" OVER follow \
        YIELD 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           |
```


2021 Vesoft Inc.

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

nebula> YIELD "Hel" + "\tlo" AS string1, ", World!" AS string2;
+------------+-----------+
| string1    | string2   |
+------------+-----------+
| "Hel    lo" | ", World!" |
+------------+-----------+

nebula> YIELD hash("Tim") % 100;
+----------------+
| (hash(Tim)%100) |
+----------------+
| 42             |
+----------------+

nebula> YIELD \
      CASE 2+3 \
      WHEN 4 THEN 0 \
      WHEN 5 THEN 1 \
      ELSE -1 \
      END \
      AS result;
+--------+
| result |
+--------+
| 1      |
+--------+
```

最后更新: 2021年5月18日

2021 Vesoft Inc.

5.7.8 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兼容语句。

> **Note**
>
> 在原生nGQL中，有与 `WITH` 类似的管道符，但它们的工作方式不同。不要在OpenCypher兼容语句中使用管道符，也不要在原生nGQL中使用 `WITH` 子句。

组成复合查询

使用 `WITH` 子句可以组合语句，将一条语句的输出转换为另一条语句的输入。

示例1

1. 匹配一个路径。
2. 通过 `nodes()` 函数将路径上的所有点输出到一个列表。
3. 将列表拆分为行。
4. 去重后返回点的信息。

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})--() \
        WITH nodes(p) AS n \
        UNWIND n AS n1 \
        RETURN DISTINCT n1;
+-----------------------------------------------------------------+
| n1                                                              |
+-----------------------------------------------------------------+
| ("player100" :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;
+----------+
```


2021 Vesoft Inc.

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

结合RETURN语句使用

在 `WITH` 子句中设置别名，并通过 `RETURN` 子句输出结果。

```
nebula> WITH [1, 2, 3] AS list  RETURN 3 IN list AS r;
+------+
| r    |
+------+
| true |
+------+

nebula> WITH 4 AS one, 3 AS two RETURN one > two AS result;
+--------+
| result |
+--------+
| true   |
+--------+
```

最后更新: 2021年5月20日

2021 Vesoft Inc.

5.8 图空间语句

5.8.1 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。

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。

> **Note**
>
> 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

用户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。

  > **Caution**
  >
  > 如果将副本数设置为1，用户将无法使用BALANCE命令为Nebula Graph的存储服务平衡负载或扩容。

- `vid_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。

2021 Vesoft Inc.

> **VID 类型变更与长度限制** `The VID must be a 64-bit integer or a string`
>
> 1. 在 Nebula Graph 1.x 中 VID 的类型只能为 int64，不支持字符型; 在Nebula Graph 2.x 中, VID 的默认类型更改为 `FIXED_STRING(<N>)`。如果继续使用 1.x 版本的 `INSERT` 语句，通常会报错不匹配 `The VID must be a 64-bit integer or a string`。如果仍要使用 int64 作为 VID, 必须指定参数 `vid_type=INT64`。
> 2. VID 最大长度必须为 `N`，不可任意长度；超过该长度也会报错 `The VID must be a 64-bit integer or a string`。

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

> **立刻尝试使用刚创建的图空间可能会失败，因为创建是异步实现的。**
>
> 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` 或 `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，请为所有配置文件修改参数 `heartbeat_interval_secs`。但过短的心跳周期（<5秒）可能会导致分布式系统中的机器误判对端失联。

检查分片分布情况

在大型集群中，由于启动时间不同，分片的分布可能不均衡。用户可以执行如下命令检查分片的分布情况：

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

如果需要均衡负载，请执行如下命令：

```
nebula> BALANCE LEADER;
```

最后更新: 2021年5月25日

2021 Vesoft Inc.

5.8.2 USE

`USE` 语句可以指定一个图空间，或切换到另一个图空间，将其作为后续查询的工作空间。

前提条件

执行 `USE` 语句指定图空间时，需要当前登录的用户拥有指定图空间的权限，否则会报错。

语法

```
USE <graph_space_name>;
```

示例

```
# 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;
```

> **Note**
>
> 不能在一条语句中同时操作两个图空间。

与Fabric Cypher不同，Nebula Graph的图空间彼此之间是完全隔离的，将一个图空间作为工作空间后，用户无法访问其他空间。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。

最后更新: 2021年5月14日

2021 Vesoft Inc.

5.8.3 SHOW SPACES

`SHOW SPACES` 语句可以列出Nebula Graph示例中的所有图空间。

语法

```
SHOW SPACES;
```

示例

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

创建图空间请参见CREATE SPACE。

最后更新: 2021年4月13日

2021 Vesoft Inc.

5.8.4 DESCRIBE SPACE

`DESCRIBE SPACE` 语句可以显示指定图空间的信息。

语法

你可以用 `DESC` 作为 `DESCRIBE` 的缩写。

```
DESC[RIBE] SPACE <graph_space_name>;
```

示例

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

最后更新: 2021年4月13日

2021 Vesoft Inc.

5.8.5 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。

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` 。

> **Caution**
>
> 请**谨慎**执行删除图空间操作。

最后更新: 2021年5月14日

2021 Vesoft Inc.

5.9 标签语句

5.9.1 CREATE TAG

`CREATE TAG` 语句可以通过指定名称创建一个标签。

OpenCypher兼容性

nGQL中的tag和OpenCypher中的label相似，但又有所不同，例如它们的创建方式。

- OpenCypher中的label需要在 `CREATE` 语句中与点一起创建。
- nGQL中的tag需要使用 `CREATE TAG` 语句独立创建。tag更像是MySQL中的表。

前提条件

执行 `CREATE TAG` 语句需要当前登录的用户拥有指定图空间的创建标签权限，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` ：用户可以使用 `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。

  > **Note**
  >
  > 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。

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、布尔等文档。

- `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。

2021 Vesoft Inc.

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。

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` 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，请为所有配置文件修改参数 `heartbeat_interval_secs`。

最后更新: 2021年5月24日

2021 Vesoft Inc.

5.9.2 DROP TAG

`DROP TAG` 语句可以删除当前工作空间内的指定标签。

点可以有一个或多个标签。

- 如果某个点只有一个标签，删除这个标签后，用户就**无法访问**这个点，下次Compaction操作时会删除该点，但点上的边仍然存在。
- 如果某点有多个标签，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

- 登录的用户必须拥有对应权限才能执行 `DROP TAG` 语句。详情请参见内置角色权限。
- 确保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。

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;
```

> **Note**
>
> nGQL中没有移除指定点的某个标签的语句。
>
> - 在openCypher中，可以使用 `REMOVE v:LABEL` 语句来移除该点 `v` 的标签 `LABLE` 。
> - 在nGQL中，通过 `CREATE TAG` 和 `INSERT VERTEX` 语句后，可以为该点增加一个 `TAG` ；但是没有任何办法移除这个点的 `TAG` 。
>
> 建议在schema中增加一个字段来标识软移除，例如为每个TAG的schema增加一个字段 `removed` 。

最后更新: 2021年5月25日

2021 Vesoft Inc.

5.9.3 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（Time-To-Live）。

前提条件

- 登录的用户必须拥有对应权限才能执行 `ALTER TAG` 语句。详情请参见内置角色权限。
- 确保要修改的属性不包含索引，否则 `ALTER TAG` 时会报冲突错误 `[ERROR (-8)]: Conflict!` 。删除索引请参见drop 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` 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，请为所有配置文件修改参数 `heartbeat_interval_secs` 。

最后更新: 2021年5月14日

2021 Vesoft Inc.

5.9.4 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，但是返回结果由登录的用户权限决定。

语法

```
SHOW TAGS;
```

示例

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

最后更新: 2021年5月14日

2021 Vesoft Inc.

5.9.5 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

登录的用户必须拥有对应权限才能执行 `DESCRIBE TAG` 语句。详情请参见内置角色权限。

语法

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

`DESCRIBE` 可以缩写为 `DESC` 。

示例

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

最后更新: 2021年5月14日

2021 Vesoft Inc.

5.10 边类型语句

5.10.1 CREATE EDGE

`CREATE EDGE` 语句可以通过指定名称创建一个边类型。

OpenCypher兼容性

nGQL中的边类型和OpenCypher中的关系类型相似，但又有所不同，例如它们的创建方式。

- OpenCypher中的关系类型需要在 `CREATE` 语句中与点一起创建。
- nGQL中的边类型需要使用 `CREATE EDGE` 语句独立创建。边类型更像是MySQL中的表。

前提条件

执行 `CREATE EDGE` 语句需要当前登录的用户拥有指定图空间的创建边类型权限，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。

  > **Note**
  >
  > 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。

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、布尔等文档。

- `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。

2021 Vesoft Inc.

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。

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";
```

最后更新: 2021年5月24日

2021 Vesoft Inc.

5.10.2 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，用户就**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

- 登录的用户必须拥有对应权限才能执行 `DROP EDGE` 语句。详情请参见内置角色权限。
- 确保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。

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;
```

最后更新: 2021年5月14日

2021 Vesoft Inc.

5.10.3 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（Time-To-Live）。

前提条件

- 登录的用户必须拥有对应权限才能执行 `ALTER EDGE` 语句。详情请参见内置角色权限。
- 确保要修改的属性不包含索引，否则 `ALTER EDGE` 时会报冲突错误 `[ERROR (-8)]: Conflict!` 。删除索引请参见drop index。

语法

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

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` 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，请为所有配置文件修改参数 `heartbeat_interval_secs` 。

最后更新: 2021年5月14日

2021 Vesoft Inc.

5.10.4 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，但是返回结果由登录的用户权限决定。

语法

```
SHOW EDGES;
```

示例

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

最后更新: 2021年5月14日

2021 Vesoft Inc.

5.10.5 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

登录的用户必须拥有对应权限才能执行 `DESCRIBE EDGE` 语句。详情请参见内置角色权限。

语法

```
DESC[RIBE] EDGE <edge_type_name>
```

`DESCRIBE` 可以缩写为 `DESC` 。

示例

```
nebula> DESCRIBE EDGE follow;
+----------+---------+-------+-----------+
| Field    | Type    | Null  | Default   |
+----------+---------+-------+-----------+
| "degree" | "int64" | "YES" | __EMPTY__ |
+----------+---------+-------+-----------+
```

最后更新: 2021年5月14日

2021 Vesoft Inc.

5.11 点语句

5.11.1 INSERT VERTEX

`INSERT VERTEX` 语句可以在Nebula Graph实例的指定图空间中插入一个或多个点。

前提条件

执行 `INSERT VERTEX` 语句需要当前登录的用户拥有指定图空间的插入点权限，否则会报错。

语法

```
INSERT VERTEX <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。
- `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。
- `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。

> **Caution**
>
> `INSERT VERTEX` 与openCypher中 `CREATE` 的语意不同：
>
> - `INSERT VERTEX` 语意更接近于NoSQL(key-value)方式的INSERT语意，或者SQL中的 `UPSERT` (`UPDATE` or `INSERT`)。
> - 相同 `VID` 和 `TAG` 的情况下，新写入的数据会覆盖旧数据，不存在时会新写入。
> - 相同 `VID` 但不同 `TAG` 的情况下，不同TAG对应的记录不会相互覆盖，不存在会新写入。

参考以下示例。

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");
```


2021 Vesoft Inc.

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__}) |
+---------------------------------------------+
```

最后更新: 2021年5月20日

2021 Vesoft Inc.

5.11.2 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。用户可以结合管道符一起使用，详情请参见管道符。

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先检索并删除目标点的所有出边和入边，然后删除目标点。

> **Caution**
>
> - 不支持原子性删除，如果发生错误请重试，避免出现部分删除的情况。
> - 删除超级节点耗时较多，为避免删除完成前连接超时，可以调整 `nebula-graphd.conf` 中的参数 `--storage_client_timeout_ms` 延长超时时间。

最后更新: 2021年5月26日

2021 Vesoft Inc.

5.11.3 UPDATE VERTEX

`UPDATE VERTEX` 语句可以修改点上标签的属性值。

Nebula Graph支持CAS（compare and set）操作。

> **Note**
>
> 一次只能修改一个标签。

语法

```
UPDATE VERTEX ON <tag_name> <vid>
SET <update_prop>
[WHEN <condition>]
[YIELD <output>]
```

参数	是否必须	说明	示例
`ON <tag_name>`	是	指定点的标签。要修改的属性必须在这个标签内。	`ON player`
`<vid>`	是	指定要修改的点ID。	`"player100"`
`SET <update_prop>`	是	指定如何修改属性值。	`SET age = age +1`
`WHEN <condition>`	否	指定过滤条件。如果 `<condition>` 结果为 `false`， `SET` 子句不会生效。	`WHEN name == "Tim"`
`YIELD <output>`	否	指定语句的输出格式。	`YIELD name AS Name`

示例

```
// 查看点”player101“的属性。
nebula> FETCH PROP ON player "player101";
+--------------------------------------------------+
| vertices_                                        |
+--------------------------------------------------+
| ("player101" :player{age: 36, name: "Tony Parker"}) |
+--------------------------------------------------+

// 修改属性age的值，并返回name和新的age。
nebula> UPDATE VERTEX ON player "player101" \
        SET age = age + 2 \
        WHEN name == "Tony Parker" \
        YIELD name AS Name, age AS Age;
+---------------+-----+
| Name          | Age |
+---------------+-----+
| "Tony Parker" | 38  |
+---------------+-----+
```

最后更新: 2021年5月10日

2021 Vesoft Inc.

5.11.4 UPSERT VERTEX

`UPSERT VERTEX` 语句结合 `UPDATE` 和 `INSERT`，如果点存在，会修改点的属性值；如果点不存在，会插入新的点。

> **Note**
>
> `UPSERT VERTEX` 一次只能修改一个标签。

`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。

> **Danger**
>
> 禁止在高并发写操作的情况下使用 `UPSERT` 语句，请使用 `UPDATE` 或 `INSERT` 代替。

语法

```
UPSERT VERTEX ON <tag> <vid>
SET <update_prop>
[WHEN <condition>]
[YIELD <output>]
```

参数	是否必须	说明	示例
`ON <tag>`	是	指定点的标签。要修改的属性必须在这个标签内。	`ON player`
`<vid>`	是	指定要修改或插入的点ID。	`"player100"`
`SET <update_prop>`	是	指定如何修改属性值。	`SET age = age +1`
`WHEN <condition>`	否	指定过滤条件。	`WHEN name == "Tim"`
`YIELD <output>`	否	指定语句的输出格式。	`YIELD name AS Name`

插入不存在的点

如果点不存在，无论 `WHEN` 子句的条件是否满足，都会插入点，同时执行 `SET` 子句，因此新插入的点的属性值取决于：

- `SET` 子句。
- 属性是否有默认值。

例如：

- 要插入的点包含基于标签 `player` 的属性 `name` 和 `age`。
- `SET` 子句指定 `age=30`。

不同情况下的属性值如下表。

是否满足`WHEN`子句条件	属性是否有默认值	`name`属性值	`age`属性值
是	是	默认值	`30`
是	否	`NULL`	`30`
否	是	默认值	`30`
否	否	`NULL`	`30`

2021 Vesoft Inc.

示例如下：

```
// 查看三个点是否存在，结果“Empty set”表示顶点不存在。
nebula> FETCH PROP ON * "player666", "player667", "player668";
Empty set

nebula> UPSERT VERTEX ON player "player666" \
        SET age = 30 \
        WHEN name == "Joe" \
        YIELD name AS Name, age AS Age;
+----------+----------+
| Name     | Age      |
+----------+----------+
| __NULL__ | 30       |
+----------+----------+

nebula> UPSERT VERTEX ON player "player666" SET age = 31 WHEN name == "Joe" YIELD name AS Name, age AS Age;
+----------+-----+
| Name     | Age |
+----------+-----+
| __NULL__ | 30  |
+----------+-----+

nebula> UPSERT VERTEX ON player "player667" \
        SET age = 31 \
        YIELD name AS Name, age AS Age;
+----------+-----+
| Name     | Age |
+----------+-----+
| __NULL__ | 31  |
+----------+-----+

nebula> UPSERT VERTEX ON player "player668" \
        SET name = "Amber", age = age + 1 \
        YIELD name AS Name, age AS Age;
+---------+----------+
| Name    | Age      |
+---------+----------+
| "Amber" | __NULL__ |
+---------+----------+
```

上面最后一个示例中，因为 `age` 没有默认值，插入点时， `age` 默认值为 `NULL` ，执行 `age = age + 1` 后仍为 `NULL` 。如果 `age` 有默认值，则 `age = age + 1` 可以正常执行，例如：

```
nebula> CREATE TAG player_with_default(name string, age int DEFAULT 20);
Execution succeeded

nebula> UPSERT VERTEX ON player_with_default "player101" \
        SET age = age + 1 \
        YIELD name AS Name, age AS Age;

+----------+-----+
| Name     | Age |
+----------+-----+
| __NULL__ | 21  |
+----------+-----+
```

修改存在的点

如果点存在，且满足 `WHEN` 子句的条件，就会修改点的属性值。

```
nebula> FETCH PROP ON player "player101";
+----------------------------------------------------+
| vertices_                                          |
+----------------------------------------------------+
| ("player101" :player{age: 42, name: "Tony Parker"}) |
+----------------------------------------------------+

nebula> UPSERT VERTEX ON player "player101" \
        SET age = age + 2 \
        WHEN name == "Tony Parker" \
        YIELD name AS Name, age AS Age;
+---------------+-----+
| Name          | Age |
+---------------+-----+
| "Tony Parker" | 44  |
+---------------+-----+
```

如果点存在，但是不满足 `WHEN` 子句的条件，修改不会生效。

```
nebula> FETCH PROP ON player "player101";
+----------------------------------------------------+
| vertices_                                          |
+----------------------------------------------------+
```


2021 Vesoft Inc.

```
| ("player101" :player{age: 44, name: "Tony Parker"}) |
+-----------------------------------------------------+

nebula> UPSERT VERTEX ON player "player101" \
        SET age = age + 2 \
        WHEN name == "Someone else" \
        YIELD name AS Name, age AS Age;
+---------------+-----+
| Name          | Age |
+---------------+-----+
| "Tony Parker" | 44  |
+---------------+-----+
```

最后更新: 2021年5月10日

2021 Vesoft Inc.

5.12 边语句

5.12.1 INSERT EDGE

`INSERT EDGE` 语句可以在Nebula Graph实例的指定图空间中插入一条或多条边。边是有方向的，从起始点（src_vid）到目的点（dst_vid）。

`INSERT EDGE` 的执行方式为覆盖式插入。如果已有边类型、起点、终点、rank都相同的边，则**覆盖**原边。

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。
- `<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。

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。

2021 Vesoft Inc.

```
# 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"}] |
+----------------------------------------+
```

Note

- 目前还不能保证操作的原子性，如果失败请重试，否则会发生部分写入。此时读取该数据的行为是未定义的。
- 并发写入同一条边会报 `edge conflict` 错误，可稍后重试。

最后更新: 2021年5月24日

2021 Vesoft Inc.

5.12.2 DELETE EDGE

`DELETE EDGE` 语句可以删除边。一次可以删除一条或多条边。用户可以结合管道符一起使用，详情请参见管道符。

如果需要删除一个点的所有出边，请删除这个点。详情请参见DELETE VERTEX。

> **Note**
>
> 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;
```

最后更新: 2021年5月14日

2021 Vesoft Inc.

5.12.3 UPDATE EDGE

`UPDATE EDGE` 语句可以修改边上边类型的属性。

Nebula Graph支持CAS（compare and set）操作。

语法

```
UPDATE EDGE ON <edge_type>
<src_vid> -> <dst_vid> [@<rank>]
SET <update_prop>
[WHEN <condition>]
[YIELD <output>]
```

参数	是否必须	说明	示例
`ON <edge_type>`	是	指定边类型。要修改的属性必须在这个边类型内。	`ON serve`
`<src_vid>`	是	指定边的起始点ID。	`"player100"`
`<dst_vid>`	是	指定边的目的点ID。	`"team204"`
`<rank>`	否	指定边的rank值。	`10`
`SET <update_prop>`	是	指定如何修改属性值。	`SET start_year = start_year +1`
`WHEN <condition>`	否	指定过滤条件。如果 `<condition>` 结果为 `false`， `SET` 子句不会生效。	`WHEN end_year < 2010`
`YIELD <output>`	否	指定语句的输出格式。	`YIELD start_year AS Start_Year`

示例

```
// 用GO语句查看边的属性值。
nebula> GO FROM "player100" \
        OVER serve \
        YIELD serve.start_year, serve.end_year;
+------------------+----------------+
| serve.start_year | serve.end_year |
+------------------+----------------+
| 1997             | 2016           |
+------------------+----------------+

// 修改属性start_year的值，并返回end_year和新的start_year。

nebula> UPDATE EDGE on serve "player100" -> "team204"@0 \
        SET start_year = start_year + 1 \
        WHEN end_year > 2010 \
        YIELD start_year, end_year;
+------------+----------+
| start_year | end_year |
+------------+----------+
| 1998       | 2016     |
+------------+----------+
```

最后更新: 2021年5月10日

2021 Vesoft Inc.

5.12.4 UPSERT EDGE

`UPSERT EDGE` 语句结合 `UPDATE` 和 `INSERT`，如果边存在，会更新边的属性；如果边不存在，会插入新的边。

`UPSERT EDGE` 性能远低于 `INSERT`，因为 `UPSERT` 是一组分片级别的读取、修改、写入操作。

> **Danger**
>
> 禁止在高并发写操作的情况下使用 `UPSERT` 语句，请使用 `UPDATE` 或 `INSERT` 代替。

语法

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

参数	是否必须	说明	示例
`ON <edge_type>`	是	指定边类型。要修改的属性必须在这个边类型内。	`ON serve`
`<src_vid>`	是	指定边的起始点ID。	`"player100"`
`<dst_vid>`	是	指定边的目的点ID。	`"team204"`
`<rank>`	否	指定边的rank值。	`10`
`SET <update_prop>`	是	指定如何修改属性值。	`SET start_year = start_year +1`
`WHEN <condition>`	否	指定过滤条件。	`WHEN end_year < 2010`
`YIELD <output>`	否	指定语句的输出格式。	`YIELD start_year AS Start_Year`

插入不存在的边

如果边不存在，无论 `WHEN` 子句的条件是否满足，都会插入边，同时执行 `SET` 子句，因此新插入的边的属性值取决于：

- `SET` 子句。
- 属性是否有默认值。

例如：

- 要插入的边包含基于边类型 `serve` 的属性 `start_year` 和 `end_year`。
- `SET` 子句指定 `end_year = 2021`。

不同情况下的属性值如下表。

是否满足`WHEN`子句条件	属性是否有默认值	`start_year`属性值	`end_year`属性值
是	是	默认值	`2021`
是	否	`NULL`	`2021`
否	是	默认值	`2021`
否	否	`NULL`	`2021`

示例如下：

2021 Vesoft Inc.

```
// 查看如下三个点是否有serve类型的出边，结果“Empty set”表示没有serve类型的出边。
nebula> GO FROM "player666", "player667", "player668" \
        OVER serve \
        YIELD serve.start_year, serve.end_year;
Empty set

nebula> UPSERT EDGE on serve \
        "player666" -> "team200"@0 \
        SET end_year = 2021 \
        WHEN end_year == 2010 \
        YIELD start_year, end_year;
+------------+----------+
| start_year | end_year |
+------------+----------+
| __NULL__   | 2021     |
+------------+----------+

nebula> UPSERT EDGE on serve \
        "player666" -> "team200"@0 \
        SET end_year = 2022 \
        WHEN end_year == 2010 \
        YIELD start_year, end_year;
+------------+----------+
| start_year | end_year |
+------------+----------+
| __NULL__   | 2021     |
+------------+----------+

nebula> UPSERT EDGE on serve \
        "player667" -> "team200"@0 \
        SET end_year = 2022 \
        YIELD start_year, end_year;
+------------+----------+
| start_year | end_year |
+------------+----------+
| __NULL__   | 2022     |
+------------+----------+

nebula> UPSERT EDGE on serve \
        "player668" -> "team200"@0 \
        SET start_year = 2000, end_year = end_year + 1 \
        YIELD start_year, end_year;
+------------+----------+
| start_year | end_year |
+------------+----------+
| 2000       | __NULL__ |
+------------+----------+
```

上面最后一个示例中，因为 `end_year` 没有默认值，插入边时，`end_year` 默认值为 `NULL`，执行 `end_year = end_year + 1` 后仍为 `NULL`。如果 `end_year` 有默认值，则 `end_year = end_year + 1` 可以正常执行，例如：

```
nebula> CREATE EDGE serve_with_default(start_year int, end_year DEFAULT 2010);
Execution succeeded

nebula> UPSERT EDGE on serve_with_default \
        "player668" -> "team200" \
        SET end_year = end_year + 1 \
        YIELD start_year, end_year;
+------------+----------+
| start_year | end_year |
+------------+----------+
| __NULL__   | 2011     |
+------------+----------+
```

修改存在的边

如果边存在，且满足 `WHEN` 子句的条件，就会修改边的属性值。

```
nebula> MATCH (v:player{name:"Ben Simmons"})-[e:serve]-(v2) \
        RETURN e;
+------------------------------------------------------------------------+
| e                                                                      |
+------------------------------------------------------------------------+
| [:serve "player149"->"team219" @0 {end_year: 2019, start_year: 2016}] |
+------------------------------------------------------------------------+

nebula> UPSERT EDGE on serve \
        "player149" -> "team219" \
        SET end_year = end_year + 1 \
        WHEN start_year == 2016 \
        YIELD start_year, end_year;
+------------+----------+
| start_year | end_year |
+------------+----------+
| 2016       | 2020     |
+------------+----------+
```


2021 Vesoft Inc.

如果边存在，但是不满足 `WHEN` 子句的条件，修改不会生效。

```
nebula> MATCH (v:player{name:"Ben Simmons"})-[e:serve]-(v2) \
        RETURN e;
+------------------------------------------------------------------------+
| e                                                                      |
+------------------------------------------------------------------------+
| [:serve "player149"->"team219" @0 {end_year: 2020, start_year: 2016}] |
+------------------------------------------------------------------------+


nebula> UPSERT EDGE on serve \
        "player149" -> "team219" \
        SET end_year = end_year + 1 \
        WHEN start_year != 2016 \
        YIELD start_year, end_year;
+------------+----------+
| start_year | end_year |
+------------+----------+
| 2016       | 2020     |
+------------+----------+
```

最后更新: 2021年5月10日

2021 Vesoft Inc.

5.13 原生索引

5.13.1 索引介绍

Nebula Graph支持两种类型索引：原生索引和全文索引。

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。
- LOOKUP不支持 `CONTAINS` 和 `STARTS WITH` 等字符串操作符。

原生索引操作

- CREATE INDEX
- SHOW CREATE INDEX
- SHOW INDEXES
- DESCRIBE INDEX
- REBUILD INDEX
- SHOW INDEX STATUS
- DROP INDEX
- LOOKUP
- MATCH

全文索引

全文索引用于对字符串属性进行前缀搜索、通配符搜索、正则表达式搜索和模糊搜索，有如下特点：

- 只允许创建一个属性的索引。
- 只能创建指定长度（不超过256字节）字符串的索引。
- 不支持逻辑操作，例如 `AND`、`OR`、`NOT`。

> **Note**
>
> 如果需要进行整个字符串的匹配，请使用原生索引。

全文索引操作

在对全文索引执行任何操作之前，请确保已经部署全文索引。详情请参见部署全文索引和部署listener。

部署完成后，Elasticsearch集群上会自动创建全文索引。不支持重建或修改全文索引。如果需要删除全文索引，请在Elasticsearch集群上手动删除。

2021 Vesoft Inc.

使用全文索引请参见使用全文索引查询。

NULL值说明

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。

最后更新: 2021年5月24日

2021 Vesoft Inc.

5.13.2 CREATE NATIVE INDEX

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和CREATE EDGE。

如何创建全文索引，请参见部署全文索引。

使用索引必读

> **Note**
>
> 索引的概念和使用限制都较为复杂。索引配合 `LOOKUP` 和 `MATCH` 语句使用。
>
> `CREATE INDEX` 语句用于对Tag、EdgeType或其属性创建原生索引。通常分别称为“标签索引”、“边类型索引”和“属性索引”。
>
> "标签索引"和"边类型索引"应用于和"标签"、"边类型"（自身）相关的查询,而不是"基于标签上某属性"的查询。例如用 `LOOKUP` 找到"某种边类型E"的所有的边。
>
> "属性索引"目的是"基于属性的查询"。例如基于属性 `Age` 找到 `Age == 19` 的所有的点VID。
>
> 如果已经为"某个标签T的属性A"建立过属性索引 `i_TA`，则没有必要对于 `标签T` 额外再建立一个标签索引 `i_T`。这是因为查询引擎可以使用 `i_TA` 来替代 `i_T`。边类型索引同理。但是，`i_T` 却不能替代 `i_TA` 用于属性查找。

> **Warning**
>
> 不要任意在生产环境中使用索引，除非很清楚使用索引对业务的影响。索引会导致写性能下降90%甚至更多。
>
> 索引并不用于查询加速。只用于：根据属性定位到点或边，或者统计点边数量。
>
> 长索引会降低Storage服务的扫描性能，以及占用更多内存。建议将索引长度设置为和要被索引的最长字符串相同。索引长度最长为255，超过部分会被截断。

如果必须使用索引，通常按照如下步骤：

1. 初次导入数据至Nebula Graph。
2. 创建索引。
3. 重建索引。
4. 使用LOOKUP或MATCH语句查询数据。不需要(也无法)指定使用哪个索引，Nebula Graph会自动计算。

> **Note**
>
> 如果先创建索引再导入数据，会因为写性能的下降导致导入速度极慢。
>
> 日常增量写入时保持 `--disable_auto_compaction = false`。
>
> 新创建的索引并不会立刻生效.创建新的索引并尝试立刻使用(例如 `LOOKUP` 或者 `REBUILD INDEX`)通常会失败（报错 `can't find xxx in the space`）。因为创建步骤是异步实现的，Nebula Graph要在下一个心跳周期才能完成索引的创建。可以使用如下方法之一：
>
> - 在 `SHOW TAG/EDGE INDEXES` 语句的结果中查找到新的索引。或者，
> - 等待两个心跳周期，例如20秒。如果需要修改心跳间隔，请为所有配置文件修改参数 `heartbeat_interval_secs`。

2021 Vesoft Inc.

> **Danger**
>
> 创建索引，或者删除并再次创建同名索引后，必须 `REBUILD INDEX` 。否则无法在 `MATCH` 和 `LOOKUP` 语句中返回这些数据。

语法

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

- `index_name` ：索引名。索引名在一个图空间中必须是唯一的。推荐的命名方式为 `i_tagName_propName` 。
- `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。
  - 为标签或边类型本身创建索引时，忽略 `prop_name_list` 。

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，用户可以使用 `LOOKUP` 语句查找 `带有该标签的` 所有点的VID，或者 `所有该类型的边` 的 `对应起始点VID、目的点VID、以及rank` 。详情请参见LOOKUP。

创建单属性索引

```
nebula> CREATE TAG INDEX 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。即只使用属性 `name` 的前10个字符来创建索引。

```
# 变长字符串需要指定索引长度。
nebula> CREATE TAG 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

`复合属性索引` 用于查找一个标签(或者边类型)中的多个属性（的组合）。

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

> **Warning**
>
> 不支持跨标签或边类型创建复合索引。
>
> 复合属性索引被(`LOOKUP` `或者` MATCH`)使用时，遵循"最左匹配原则"，必须从复合属性索引的最左侧开始匹配。请参见下方示例。

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

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

# 可以匹配到索引。
```


2021 Vesoft Inc.

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

最后更新: 2021年5月25日

2021 Vesoft Inc.

5.13.3 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" |
+------------------+
```

最后更新: 2021年4月1日

2021 Vesoft Inc.

5.13.4 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

用户可以先运行 `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`                                    |
|                 | )"                                           |
+----------------+----------------------------------------------+
```

最后更新: 2021年5月14日

2021 Vesoft Inc.

5.13.5 DESCRIBE INDEX

`DESCRIBE INDEX` 语句可以查看指定索引的信息，包括索引的属性名称（Field）和数据类型（Type）。

语法

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

示例

```
nebula> DESCRIBE TAG INDEX player_index_0;
+--------+--------------------+
| Field  | Type               |
+--------+--------------------+
| "name" | "fixed_string(30)" |
+--------+--------------------+

nebula> DESCRIBE TAG INDEX player_index_1;
+--------+--------------------+
| Field  | Type               |
+--------+--------------------+
| "name" | "fixed_string(10)" |
+--------+--------------------+
| "age"  | "int64"            |
+--------+--------------------+
```

最后更新: 2021年4月1日

2021 Vesoft Inc.

5.13.6 REBUILD INDEX

> **Danger**
>
> 索引功能不会自动对其创建之前已存在的存量数据生效————在索引重建完成之前，无法基于该索引使用 `LOOKUP` 和 `MATCH` 语句查询到存量数据。
>
> 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。

请在创建索引后，选择合适的时间为存量数据重建索引。使用索引的详情请参见CREATE INDEX。

语法

```
REBUILD {TAG | EDGE} INDEX [<index_name_list>];

<index_name_list>::=
    [index_name [, index_name] ...]
```

- 可以一次重建多个索引，索引名称之间用英文逗号（,）分隔。如果没有指定索引名称，将会重建所有索引。
- 重建完成后，用户可以使用命令 `SHOW {TAG | EDGE} INDEX STATUS` 检查索引是否重建完成。详情请参见SHOW INDEX STATUS。

示例

```
nebula> CREATE TAG 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。

历史版本兼容性

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

最后更新: 2021年5月25日

2021 Vesoft Inc.

5.13.7 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

> **Note**
>
> 如何创建索引请参见CREATE INDEX。

语法

```
SHOW {TAG | EDGE} INDEX STATUS;
```

示例

```
nebula> SHOW TAG INDEX STATUS;
+----------------------+--------------+
| Name                 | Index Status |
+----------------------+--------------+
| "player_index_0"     | "FINISHED"   |
+----------------------+--------------+
| "player_index_1"     | "FINISHED"   |
+----------------------+--------------+
```

最后更新: 2021年5月8日

2021 Vesoft Inc.

5.13.8 DROP INDEX

`DROP INDEX` 语句可以删除当前图空间中已存在的索引。

前提条件

执行 `DROP INDEX` 语句需要当前登录的用户拥有指定图空间的 `DROP TAG INDEX` 和 `DROP EDGE INDEX` 权限，否则会报错。

语法

```
DROP {TAG | EDGE} INDEX [IF EXISTS] <index_name>;
```

`IF NOT EXISTS` ：检测待删除的索引是否存在，只有存在时，才会删除索引。

示例

```
nebula> DROP TAG INDEX player_index_0;
```

最后更新: 2021年4月1日

2021 Vesoft Inc.

5.14 全文索引

5.14.1 全文索引限制

> **Caution**
>
> 本文介绍全文索引的限制，请在使用全文索引前仔细阅读。

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，超出的部分做截断处理。
- 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。
- 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。
- 从写入Nebula Graph，到写入listener，再到写入 Elasticsearch 并创建索引可能需要一段时间。如果访问全文索引时返回未找到索引，可等待索引生效（但是，该等待时间未知，也无返回码检查）。

最后更新: 2021年5月26日

2021 Vesoft Inc.

5.14.2 部署全文索引

Nebula Graph的全文索引是基于Elasticsearch实现，这意味着用户可以使用Elasticsearch全文查询语言来检索想要的内容。全文索引由内置的进程管理，当listener集群和Elasticsearch集群部署后，内置的进程只能为数据类型为定长字符串或变长字符串的属性创建全文索引。

注意事项

使用全文索引前，请确认已经了解全文索引的使用限制。

部署Elasticsearch集群

部署Elasticsearch集群请参见Elasticsearch官方文档。

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"}
```

用户可以配置Elasticsearch来满足业务需求，如果需要定制Elasticsearch，请参见Elasticsearch官方文档。

登录文本搜索客户端

部署Elasticsearch集群之后，可以使用 `SIGN IN` 语句登录Elasticsearch客户端。必须使用Elasticsearch配置文件中的IP地址和端口才能正常连接，同时登录多个客户端，请在多个 `elastic_ip:port` 之间用英文逗号（,）分隔。

语法

```
SIGN IN TEXT SERVICE [(<elastic_ip:port> [,<username>, <password>]), (<elastic_ip:port>), ...];
```

示例

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

> **Note**
>
> Elasticsearch默认没有用户名和密码，如果设置了用户名和密码，请在 `SIGN IN` 语句中指定。

显示文本搜索客户端

`SHOW TEXT SEARCH CLIENTS` 语句可以列出文本搜索客户端。

语法

2021 Vesoft Inc.

```
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;
```

最后更新: 2021年5月14日

2021 Vesoft Inc.

5.14.3 部署Raft listener

全文索引的数据是异步写入Elasticsearch集群的。流程是通过Storage服务的 Raft listener（简称listener）这个单独部署的进程，从Storage服务读取数据，然后将它们写入Elasticsearch集群。

前提条件

- 已经了解全文索引的使用限制。
- 已经部署Nebula Graph集群。
- 完成部署Elasticsearch集群。
- 准备一台或者多台额外的服务器，来部署Raft listener。

注意事项

- 请保证Nebula 各组件（Metad、Storaged、Graphd、listener）有相同的版本。
- 只能为一个图空间“一次性添加所有的 listener 机器”。尝试向已经存在有 listener 的图空间再添加新 listener 会失败。因此，需在一个命令语句里完整地添加全部的 listener。

部署流程

第一步：准备LISTENER的配置文件

用户必须在需要部署listener的机器上准备对应的配置文件。文件名称必须为 `nebula-storaged-listener.conf` 。用户可以参考提供的模板。注意去掉文件后缀 `.production` 。

大部分配置与Storage服务的配置文件相同，本文仅介绍差异部分。

名称	预设值	说明
`daemonize`	`true`	是否启动守护进程。
`pid_file`	`pids_listener/nebula-storaged.pid`	记录进程ID的文件。
`meta_server_addrs`	-	全部Meta服务的IP地址和端口。多个Meta服务用英文逗号（,）分隔。
`local_ip`	-	listener服务的本地IP地址。
`port`	-	listener服务的RPC守护进程监听端口。
`heartbeat_interval_secs`	`10`	Meta服务的心跳间隔。单位：秒（s）。
`listener_path`	`data/listener`	listener的WAL目录。只允许使用一个目录。
`data_path`	`data`	出于兼容性考虑，可以忽略此参数。填充一个默认值 `data` 。
`part_man_type`	`memory`	部件管理器类型，可选值为 `memory` 和 `meta` 。
`rocksdb_batch_size`	`4096`	批处理操作的默认保留字节。
`rocksdb_block_cache`	`4`	BlockBasedTable的默认块缓存大小。单位：兆字节（MB）。
`engine_type`	`rocksdb`	存储引擎类型，例如 `rocksdb` 、 `memory` 等。
`part_type`	`simple`	部件类型，例如 `simple` 、 `consensus` 等。

2021 Vesoft Inc.

> **Note**
>
> 在配置文件中请使用真实的（listener机器）IP地址替换 `127.0.0.1` 。

第二步：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` 是存放 listener 配置文件的路径。

第三步：添加 LISTENER 到 NEBULA GRAPH 集群

用命令行连接到Nebula Graph，然后执行 `USE <space>` 进入需要创建全文索引的图空间。然后执行如下命令添加 listener：

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

> **Warning**
>
> listener 必须使用真实的IP地址。
>
> 请在一个语句里完整地添加所有 listener。例如：

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

查看 listener

执行 `SHOW LISTENER` 语句可以列出所有的 listener。

示例

```
nebula> SHOW LISTENER;
+--------+-----------------+----------------------+----------+
| PartId | Type            | Host                 | Status   |
+--------+-----------------+----------------------+----------+
| 1      | "ELASTICSEARCH" | "[192.168.8.5:46780]" | "ONLINE" |
+--------+-----------------+----------------------+----------+
| 2      | "ELASTICSEARCH" | "[192.168.8.5:46780]" | "ONLINE" |
+--------+-----------------+----------------------+----------+
| 3      | "ELASTICSEARCH" | "[192.168.8.5:46780]" | "ONLINE" |
+--------+-----------------+----------------------+----------+
```

删除listener

执行 `REMOVE LISTENER ELASTICSEARCH` 语句可以删除图空间的所有listener。

示例

```
nebula> REMOVE LISTENER ELASTICSEARCH;
```

> **Danger**
>
> 删除 listener 后，将不能重新添加 listener，因此也无法继续向ES集群同步，文本索引数据将不完整。如果确实需要，只能重新创建 space。

下一步

部署全文索引和 listener 后，全文索引会在 Elasticsearch 集群中自动创建，用户可以开始进行全文搜索。详情请参见全文搜索。

2021 Vesoft Inc.

最后更新: 2021年5月26日

2021 Vesoft Inc.

5.14.4 全文搜索

全文搜索是基于全文索引对值为字符串类型的属性进行前缀搜索、通配符搜索、正则表达式搜索和模糊搜索。

在 `LOOKUP` 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和部署listener。

注意事项

使用全文索引前，请确认已经了解全文索引的使用限制。

自然语言全文搜索

自然语言搜索将搜索的字符串解释为自然人类语言中的短语。搜索不区分大小写。

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。
  - `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 basketballplayer (partition_num=3,replica_factor=1, vid_type=fixed_string(30));
nebula> SIGN IN TEXT SERVICE (127.0.0.1:9200);
nebula> USE basketballplayer;
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           |
+----------------+
```


2021 Vesoft Inc.

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

最后更新: 2021年5月14日

2021 Vesoft Inc.

5.15 子图和路径

5.15.1 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。

2021 Vesoft Inc.

2021 Vesoft Inc.

- 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";
+-----------------------
```

- 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。

最后更新: 2021年4月15日

2021 Vesoft Inc.

5.15.2 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") |
+--------------------------------------+
```


2021 Vesoft Inc.

```
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。

最后更新: 2021年4月1日

2021 Vesoft Inc.

5.16 查询调优

5.16.1 EXPLAIN和PROFILE

`EXPLAIN` 语句输出nGQL语句的执行计划，但不会执行nGQL语句。

`PROFILE` 语句执行nGQL语句，然后输出执行计划和执行概要。用户可以根据执行计划和执行概要优化查询性能。

执行计划

执行计划由Nebula Graph查询引擎中的执行计划器决定。

执行计划器将解析后的nGQL语句处理为 `action` 。 `action` 是最小的执行单元。典型的 `action` 包括获取指定点的所有邻居、获取边的属性、根据条件过滤点或边等。每个 `action` 都被分配给一个 `operator` 。

例如 `SHOW TAGS` 语句分为两个 `action` ， `operator` 为 `Start` 和 `ShowTags` 。更复杂的 `GO` 语句可能会被处理成10个以上的 `action` 。

语法

- `EXPLAIN`

```
EXPLAIN [format="row" | "dot"] <your_nGQL_statement>;
```

- `PROFILE`

```
PROFILE [format="row" | "dot"] <your_nGQL_statement>;
```

输出格式

`EXPLAIN` 或 `PROFILE` 语句的输出有两种格式： `row` （默认）和 `dot` 。用户可以使用 `format` 选项修改输出格式。

2021 Vesoft Inc.

row格式

`row` 格式将返回信息输出到一个表格中。

- `EXPLAIN`

```
nebula> EXPLAIN format="row" SHOW TAGS;
Execution succeeded (time spent 327/892 us)

Execution Plan

-----+----------+--------------+----------------+-------------------------------------------------------------------
| id | name     | dependencies | profiling data | operator info                                                     |
-----+----------+--------------+----------------+-------------------------------------------------------------------
|  1 | ShowTags | 0            |                | outputVar: [{"colNames":[],"name":"__ShowTags_1","type":"DATASET"}] |
|    |          |              |                | inputVar:                                                         |
-----+----------+--------------+----------------+-------------------------------------------------------------------
|  0 | Start    |              |                | outputVar: [{"colNames":[],"name":"__Start_0","type":"DATASET"}]    |
-----+----------+--------------+----------------+-------------------------------------------------------------------
```

- `PROFILE`

```
nebula> PROFILE format="row" SHOW TAGS;
+--------+
| Name   |
+--------+
| player |
+--------+
| team   |
+--------+
Got 2 rows (time spent 2038/2728 us)

Execution Plan

-----+----------+--------------+-------------------------------------------------+-------------------------------------------------------------------
| id | name     | dependencies | profiling data                                  | operator info                                                     |
-----+----------+--------------+-------------------------------------------------+-------------------------------------------------------------------
|  1 | ShowTags | 0            | ver: 0, rows: 1, execTime: 42us, totalTime: 1177us | outputVar: [{"colNames":[],"name":"__ShowTags_1","type":"DATASET"}] |
|    |          |              |                                                 | inputVar:                                                         |
-----+----------+--------------+-------------------------------------------------+-------------------------------------------------------------------
|  0 | Start    |              | ver: 0, rows: 0, execTime: 1us, totalTime: 57us | outputVar: [{"colNames":[],"name":"__Start_0","type":"DATASET"}]    |
-----+----------+--------------+-------------------------------------------------+-------------------------------------------------------------------
```

参数	说明
`id`	`operator` 的ID。
`name`	`operator` 的名称。
`dependencies`	当前 `operator` 所依赖的 `operator` 的ID。
`profiling data`	执行概要文件内容。 `ver` 表示 `operator` 的版本； `rows` 表示 `operator` 输出结果的行数； `execTime` 表示执行 `action` 的时间； `totalTime` 表示执行 `action` 的时间、系统调度时间、排队时间的总和。
`operator info`	`operator` 的详细信息。

dot格式

`dot` 格式将返回DOT语言的信息，然后用户可以使用Graphviz生成计划图。

> **Note**
>
> Graphviz是一款开源可视化图工具，可以绘制DOT语言脚本描述的图。Graphviz提供一个在线工具，可以预览DOT语言文件，并将它们导出为SVG或JSON等其他格式。详情请参见Graphviz Online。

```
nebula> EXPLAIN format="dot" SHOW TAGS;
Execution succeeded (time spent 161/665 us)
Execution Plan
--------------------------------------------------------------------------------------------------------------------------------------------------------------  -------------
  plan
--------------------------------------------------------------------------------------------------------------------------------------------------------------  -------------
  digraph exec_plan {
      rankdir=LR;
      "ShowTags_0"[label="ShowTags_0|outputVar: \[\{\"colNames\":\[\],\"name\":\"__ShowTags_0\",\"type\":\"DATASET\"\}\]\l|inputVar:\l",   shape=Mrecord];
      "Start_2"->"ShowTags_0";
      "Start_2"[label="Start_2|outputVar: \[\{\"colNames\":\[\],\"name\":\"__Start_2\",\"type\":\"DATASET\"\}\]\l|inputVar: \l",   shape=Mrecord];
```


2021 Vesoft Inc.

```
  }
--------------------
```

5.17 运维

5.17.1 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。

`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。

最后更新: 2021年5月26日

2021 Vesoft Inc.

5.17.2 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。

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，用户可以使用 `SHOW STATS` 语句列出统计结果。详情请参见SHOW STATS。

> **Note**
>
> 如果存储在Nebula Graph中的数据有变化，为了获取最新的统计结果，请重新执行 `SUBMIT JOB STATS` 。

示例

```
nebula> SUBMIT JOB STATS;
+------------+
| New Job Id |
+------------+
| 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 |
+----------------+---------------+------------+-------------------------+-------------------------+
```


2021 Vesoft Inc.

```
| 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。
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。

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。

2021 Vesoft Inc.

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。用户可以执行如下命令调试：

```
curl "http://{storaged-ip}:19779/admin?space={space_name}&op=compact"
```

最后更新: 2021年5月14日

2021 Vesoft Inc.

6. 安装部署

6.1 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。

6.1.1 阅读指南

如果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？
- 测试环境中运行Nebula Graph的要求是什么？
- 生产环境中运行Nebula Graph的要求是什么？
- 需要预留多少内存和硬盘空间给Nebula Graph集群？

6.1.2 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，建议使用内核版本为 `2.6.32` 及以上版本的Linux系统。

2021 Vesoft Inc.

软件要求

软件版本需要如下表所示，如果它们不符合要求，或者也不确定它们的版本，请按照安装编译所需软件中的步骤进行操作。

软件名称	版本	备注
glibc	2.12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。

2021 Vesoft Inc.

安装编译所需软件

本小节指导下载和安装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。

```
$ g++ --version
$ cmake --version
```

   如果版本正确，用户可以跳过本小节。如果不正确，请根据如下步骤安装：

   a. 克隆仓库 `nebula-common` 到主机。

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

   如需安装特定版本的Nebula Graph，使用 `--branch` 或 `-b` 选项指定相应的nebula-common分支。 例如，指定book，命令如下：

```
$ git clone --branch vbook https://github.com/vesoft-inc/nebula-common.git
```

   b. 进入目录 `nebula-common` 。

```
$ cd nebula-common
```

   c. 执行如下命令安装和启用GCC和CMake。

```
// 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，用户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
```

3. 执行脚本 `install-third-party.sh` 。

```
$ sudo ./third-party/install-third-party.sh
```


2021 Vesoft Inc.

6.1.3 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，建议在测试环境中使用内核版本为 `3.9` 及以上版本的Linux系统。

服务架构建议

进程	建议数量
metad（meta数据服务进程）	1
storaged（存储服务进程）	≥1
graphd（查询引擎服务进程）	≥1

例如单机测试环境，用户可以在机器上部署1个metad、1个storaged和1个graphd进程。

对于更常见的测试环境，例如三台机器构成的集群，用户可以按照如下方案部署Nebula Graph。

机器名称	metad进程数量	storaged进程数量	graphd进程数量
A	1	1	1
B	-	1	1
C	-	1	1

6.1.4 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，建议在生产环境中使用内核版本为 `3.9` 及以上版本的Linux系统。

用户可以通过调整一些内核参数来提高Nebula Graph性能，详情请参见内核配置。

2021 Vesoft Inc.

服务架构建议

不要跨机房部署集群

进程	数量
metad（meta数据服务进程）	3
storaged（存储服务进程）	≥3
graphd（查询引擎服务进程）	≥3

有且仅有3个metad进程，每个metad进程会自动创建并维护meta数据的一个副本。

storaged进程的数量不会影响图空间副本的数量。

用户可以在一台机器上部署多个进程，例如五台机器构成的集群，用户可以按照如下方案部署Nebula Graph。

机器名称	metad进程数量	storaged进程数量	graphd进程数量
A	1	1	1
B	1	1	1
C	1	1	1
D	-	1	1
E	-	1	1

6.1.5 Nebula Graph资源要求

用户可以预估一个3副本Nebula Graph集群所需的内存、硬盘空间和分区数量。

资源	单位	计算公式	说明
硬盘空间	Bytes	`点和边的总数` * `属性的平均字节大小` * 6 * 120%	-
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。块缓存大小请参见Memory usage in RocksDB。
分区数量	-	`集群硬盘数量` * `disk_partition_num_multiplier`	`disk_partition_num_multiplier` 是取值为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。

Note

用户可以在配置文件 `nebula-storaged.conf` 中添加 `--enable_partitioned_index_filter=true` 来降低bloom过滤器占用的内存大小，但是在某些随机寻道（random-seek）的情况下，可能会降低读取性能。

2021 Vesoft Inc.

6.1.6 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。

最后更新: 2021年5月26日

2021 Vesoft Inc.

6.2 编译安装Nebula Graph

6.2.1 使用源码安装Nebula Graph

使用源码安装Nebula Graph允许自定义编译和安装设置，并测试最新特性。

前提条件

- 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。
- 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：

     ```
     $ 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book 发布版本，请执行如下命令：

     ```
     $ git clone --branch vbook 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。

   > **Note**
   >
   > 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。

   - 如果在第1步下载的是最新版本的开发代码，请执行如下命令：

     ```
     $ cmake -DENABLE_BUILD_STORAGE=on -DENABLE_TESTING=OFF -DCMAKE_BUILD_TYPE=Release ..
     ```

   - 如果在第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和nebula-storage分支。例如安装book GA版本，请执行如下命令：

     ```
     $ cmake -DENABLE_BUILD_STORAGE=on -DENABLE_TESTING=OFF -DCMAKE_BUILD_TYPE=Release \
     -DNEBULA_COMMON_REPO_TAG=vbook -DNEBULA_STORAGE_REPO_TAG=vbook ..
     ```

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
   ```


2021 Vesoft Inc.

7. （可选）更新 `master` 分支的源码，因为它经常更新。
   a. 在目录 `nebula-graph/` 中，执行命令 `git pull upstream master` 更新源码。
   b. 在目录 `nebula-graph/modules/common/` 和 `nebula-graph/modules/storage/` 中，分别执行命令 `git pull upstream master` 。
   c. 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
- 连接Nebula Graph
- 基础操作语法

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book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，用户可以设置为 `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。

2021 Vesoft Inc.

CMAKE_C_COMPILER/CMAKE_CXX_COMPILER

通常情况下，CMake会自动查找并使用主机上的C/C++编译器，但是如果编译器没有安装在标准路径，或者想使用其他编译器，请执行如下命令指定目标编译器的安装路径：

```
$ cmake -DCMAKE_C_COMPILER=<path_to_gcc/bin/gcc> -DCMAKE_CXX_COMPILER=<path_to_gcc/bin/g++> ..
$ cmake -DCMAKE_C_COMPILER=<path_to_clang/bin/clang> -DCMAKE_CXX_COMPILER=<path_to_clang/bin/clang++> ..
```

ENABLE_CCACHE

`ENABLE_CCACHE` 默认值为 `ON`，表示使用ccache（compiler cache）工具加速编译。

如果想要禁用ccache，仅仅设置 `ENABLE_CCACHE=OFF` 是不行的，因为在某些平台上，ccache会代理当前编译器，因此还需要设置环境变量 `export CCACHE_DISABLE=true`，或者在文件 `~/.ccache/ccache.conf` 中添加 `disable=true`。更多信息请参见ccache official documentation。

NEBULA_THIRDPARTY_ROOT

`NEBULA_THIRDPARTY_ROOT` 指定第三方软件的安装路径，默认路径为 `/opt/vesoft/third-party`。

最后更新: 2021年5月26日

2021 Vesoft Inc.

6.2.2 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

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 `book` 安装包：

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

  下载适用于 `ubuntu 1804` 的 `book` 安装包：

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


2021 Vesoft Inc.

- 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页面，确认需要的版本，单击**Assets**。

2021 Vesoft Inc.

- 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。

2021 Vesoft Inc.

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>
```

> **Note**
>
> 如果不设置安装路径，默认安装路径为 `/usr/local/nebula/` 。

最后更新: 2021年5月10日

2021 Vesoft Inc.

6.3 部署Nebula Graph集群

Nebula Graph不提供官方的集群部署工具，需要手动部署，下文将介绍手动部署的简单流程。

6.3.1 部署方案

机器名称	IP地址	graphd进程数量	storaged进程数量	metad进程数量
A	192.168.10.111	1	1	1
B	192.168.10.112	1	1	1
C	192.168.10.113	1	1	1
D	192.168.10.114	1	1	-
E	192.168.10.115	1	1	-

6.3.2 前提条件

准备5台用于部署集群的机器。

6.3.3 手动部署流程

1.安装Nebula Graph

在集群的每一台服务器上都安装Nebula Graph，安装后暂不需要启动服务。安装方式请参见：

- 使用RPM或DEB安装包安装Nebula Graph
- 编译安装Nebula Graph

2.修改配置文件

修改每个服务器上的Nebula Graph配置文件。

Nebula Graph的所有配置文件均位于安装目录的 `etc` 目录内，包括 `nebula-graphd.conf` 、 `nebula-metad.conf` 和 `nebula-storaged.conf` ，用户可以只修改所需服务的配置文件。各个机器需要修改的配置文件如下。

机器名称	待修改配置文件
A	`nebula-graphd.conf` 、 `nebula-storaged.conf` 、 `nebula-metad.conf`
B	`nebula-graphd.conf` 、 `nebula-storaged.conf` 、 `nebula-metad.conf`
C	`nebula-graphd.conf` 、 `nebula-storaged.conf` 、 `nebula-metad.conf`
D	`nebula-graphd.conf` 、 `nebula-storaged.conf`
E	`nebula-graphd.conf` 、 `nebula-storaged.conf`

用户可以参考如下配置文件的内容，仅展示集群通信的部分设置，未展示的内容为默认设置，便于用户了解集群间各个服务器的关系。

2021 Vesoft Inc.

Note

主要修改的配置是 `meta_server_addrs`，所有配置文件都需要填写所有Meta服务的IP地址和端口，同时需要修改 `local_ip` 为机器本身的联网IP地址。配置参数的详细说明请参见：

- Meta服务配置
- Graph服务配置
- Storage服务配置

- 机器A配置
  - `nebula-graphd.conf`

```
########## networking ##########
# Comma separated Meta Server Addresses
--meta_server_addrs=192.168.10.111:9559,192.168.10.112:9559,192.168.10.113:9559
# Local IP used to identify the nebula-graphd process.
# Change it to an address other than loopback if the service is distributed or
# will be accessed remotely.
--local_ip=192.168.10.111
# Network device to listen on
--listen_netdev=any
# Port to listen on
--port=9669
```

  - `nebula-storaged.conf`

```
########## networking ##########
# Comma separated Meta server addresses
--meta_server_addrs=192.168.10.111:9559,192.168.10.112:9559,192.168.10.113:9559
# Local IP used to identify the nebula-storaged process.
# Change it to an address other than loopback if the service is distributed or
# will be accessed remotely.
--local_ip=192.168.10.111
# Storage daemon listening port
--port=9779
```

  - `nebula-metad.conf`

```
########## networking ##########
# Comma separated Meta Server addresses
--meta_server_addrs=192.168.10.111:9559,192.168.10.112:9559,192.168.10.113:9559
# Local IP used to identify the nebula-metad process.
# Change it to an address other than loopback if the service is distributed or
# will be accessed remotely.
--local_ip=192.168.10.111
# Meta daemon listening port
--port=9559
```


2021 Vesoft Inc.

- 机器B配置
  - `nebula-graphd.conf`

```
########## networking ##########
# Comma separated Meta Server Addresses
--meta_server_addrs=192.168.10.111:9559,192.168.10.112:9559,192.168.10.113:9559
# Local IP used to identify the nebula-graphd process.
# Change it to an address other than loopback if the service is distributed or
# will be accessed remotely.
--local_ip=192.168.10.112
# Network device to listen on
--listen_netdev=any
# Port to listen on
--port=9669
```

  - `nebula-storaged.conf`

```
########## networking ##########
# Comma separated Meta server addresses
--meta_server_addrs=192.168.10.111:9559,192.168.10.112:9559,192.168.10.113:9559
# Local IP used to identify the nebula-storaged process.
# Change it to an address other than loopback if the service is distributed or
# will be accessed remotely.
--local_ip=192.168.10.112
# Storage daemon listening port
--port=9779
```

  - `nebula-metad.conf`

```
########## networking ##########
# Comma separated Meta Server addresses
--meta_server_addrs=192.168.10.111:9559,192.168.10.112:9559,192.168.10.113:9559
# Local IP used to identify the nebula-metad process.
# Change it to an address other than loopback if the service is distributed or
# will be accessed remotely.
--local_ip=192.168.10.112
# Meta daemon listening port
--port=9559
```

- 机器C配置
  - `nebula-graphd.conf`

```
########## networking ##########
# Comma separated Meta Server Addresses
--meta_server_addrs=192.168.10.111:9559,192.168.10.112:9559,192.168.10.113:9559
# Local IP used to identify the nebula-graphd process.
# Change it to an address other than loopback if the service is distributed or
# will be accessed remotely.
--local_ip=192.168.10.113
# Network device to listen on
--listen_netdev=any
# Port to listen on
--port=9669
```

  - `nebula-storaged.conf`

```
########## networking ##########
# Comma separated Meta server addresses
--meta_server_addrs=192.168.10.111:9559,192.168.10.112:9559,192.168.10.113:9559
# Local IP used to identify the nebula-storaged process.
# Change it to an address other than loopback if the service is distributed or
# will be accessed remotely.
--local_ip=192.168.10.113
# Storage daemon listening port
--port=9779
```

  - `nebula-metad.conf`

```
########## networking ##########
# Comma separated Meta Server addresses
--meta_server_addrs=192.168.10.111:9559,192.168.10.112:9559,192.168.10.113:9559
# Local IP used to identify the nebula-metad process.
# Change it to an address other than loopback if the service is distributed or
# will be accessed remotely.
--local_ip=192.168.10.113
# Meta daemon listening port
--port=9559
```


2021 Vesoft Inc.

- 机器D配置
  - `nebula-graphd.conf`

```
########## networking ##########
# Comma separated Meta Server Addresses
--meta_server_addrs=192.168.10.111:9559,192.168.10.112:9559,192.168.10.113:9559
# Local IP used to identify the nebula-graphd process.
# Change it to an address other than loopback if the service is distributed or
# will be accessed remotely.
--local_ip=192.168.10.114
# Network device to listen on
--listen_netdev=any
# Port to listen on
--port=9669
```

  - `nebula-storaged.conf`

```
########## networking ##########
# Comma separated Meta server addresses
--meta_server_addrs=192.168.10.111:9559,192.168.10.112:9559,192.168.10.113:9559
# Local IP used to identify the nebula-storaged process.
# Change it to an address other than loopback if the service is distributed or
# will be accessed remotely.
--local_ip=192.168.10.114
# Storage daemon listening port
--port=9779
```

- 机器E配置
  - `nebula-graphd.conf`

```
########## networking ##########
# Comma separated Meta Server Addresses
--meta_server_addrs=192.168.10.111:9559,192.168.10.112:9559,192.168.10.113:9559
# Local IP used to identify the nebula-graphd process.
# Change it to an address other than loopback if the service is distributed or
# will be accessed remotely.
--local_ip=192.168.10.115
# Network device to listen on
--listen_netdev=any
# Port to listen on
--port=9669
```

  - `nebula-storaged.conf`

```
########## networking ##########
# Comma separated Meta server addresses
--meta_server_addrs=192.168.10.111:9559,192.168.10.112:9559,192.168.10.113:9559
# Local IP used to identify the nebula-storaged process.
# Change it to an address other than loopback if the service is distributed or
# will be accessed remotely.
--local_ip=192.168.10.115
# Storage daemon listening port
--port=9779
```

3.启动集群

依次启动各个服务器上的对应进程。说明如下。

机器名称	待启动的进程
A	graphd、storaged、metad
B	graphd、storaged、metad
C	graphd、storaged、metad
D	graphd、storaged
E	graphd、storaged

启动Nebula Graph进程的命令如下：

```
sudo /usr/local/nebula/scripts/nebula.service start <metad|graphd|storaged|all>
```


2021 Vesoft Inc.

> **Note**
>
> - graphd、storaged和metad都启动时，可以用all代替。
> - `/usr/local/nebula` 是Nebula Graph的默认安装路径，如果修改过安装路径，请使用实际路径。更多启停服务的内容，请参见管理Nebula Graph服务。

4.检查集群

安装原生CLI客户端Nebula Console，然后连接任何一个已启动graphd进程的机器，执行命令 `SHOW HOSTS` 检查集群状态。例如：

```
$ ./nebula-console --addr 192.168.10.111 --port 9669 -u root -p nebula

2021/05/25 01:41:19 [INFO] connection pool is initialized successfully
Welcome to Nebula Graph!

> SHOW HOSTS;
+-----------------+------+----------+--------------+----------------------+------------------------+
| Host            | Port | Status   | Leader count | Leader distribution  | Partition distribution |
+-----------------+------+----------+--------------+----------------------+------------------------+
| "192.168.10.111" | 9779 | "ONLINE" | 0            | "No valid partition" | "No valid partition"   |
+-----------------+------+----------+--------------+----------------------+------------------------+
| "192.168.10.112" | 9779 | "ONLINE" | 0            | "No valid partition" | "No valid partition"   |
+-----------------+------+----------+--------------+----------------------+------------------------+
| "192.168.10.113" | 9779 | "ONLINE" | 0            | "No valid partition" | "No valid partition"   |
+-----------------+------+----------+--------------+----------------------+------------------------+
| "192.168.10.114" | 9779 | "ONLINE" | 0            | "No valid partition" | "No valid partition"   |
+-----------------+------+----------+--------------+----------------------+------------------------+
| "192.168.10.115" | 9779 | "ONLINE" | 0            | "No valid partition" | "No valid partition"   |
+-----------------+------+----------+--------------+----------------------+------------------------+
| "Total"         |      |          | 0            |                      |                        |
+-----------------+------+----------+--------------+----------------------+------------------------+
```

最后更新: 2021年5月26日

2021 Vesoft Inc.

6.4 卸载Nebula Graph

本文介绍如何卸载Nebula Graph。

> **Caution**
>
> 如果需要重新部署Nebula Graph，请务必完全卸载后再重新部署，否则可能会出现问题，包括Meta不一致等。

6.4.1 前提条件

停止Nebula Graph服务。详情参见管理Nebula Graph服务。

6.4.2 步骤1：删除数据和元数据文件

如果在配置文件内修改了数据文件的路径，可能会导致安装路径和数据文件保存路径不一致，因此需要查看配置文件，确认数据文件保存路径，然后手动删除数据文件目录。

> **Note**
>
> 如果是集群架构，需要删除所有Storage和Meta服务节点的数据文件。

1. 检查Storage服务的disk配置。例如：

```
########## Disk ##########
# Root data path. Split by comma. e.g. --data_path=/disk1/path1/,/disk2/path2/
# One path per Rocksdb instance.
--data_path=/nebula/data/storage
```

2. 检查metad服务的配置文件，找到对应元数据目录。
3. 删除以上数据和元数据目录。

6.4.3 步骤2：卸载安装目录

> **Note**
>
> 删除整个安装目录，包括 `cluster.id` 文件。

安装路径为参数 `--prefix` 指定的路径。默认路径为 `/usr/local/nebula` 。

卸载编译安装的Nebula Graph

找到Nebula Graph的安装目录，删除整个安装目录。

卸载RPM包安装的Nebula Graph

1. 使用如下命令查看Nebula Graph版本。

```
$ rpm -qa | grep "nebula"
```

返回类似如下结果。

```
nebula-graph-book-1.x86_64
```


2021 Vesoft Inc.

2. 使用如下命令卸载Nebula Graph。

```
sudo rpm -e <nebula_version>
```

例如：

```
sudo rpm -e nebula-graph-book-1.x86_64
```

3. 删除安装目录。

卸载DEB包安装的Nebula Graph

1. 使用如下命令查看Nebula Graph版本。

```
$ dpkg -l | grep "nebula"
```

返回类似如下结果。

```
ii  nebula-graph  book  amd64    Nebula Package built using CMake
```

2. 使用如下命令卸载Nebula Graph。

```
sudo dpkg -r <nebula_version>
```

例如：

```
sudo dpkg -r nebula-graph
```

3. 删除安装目录。

卸载Docker Compose部署的Nebula Graph

1. 在目录 `nebula-docker-compose` 内执行如下命令停止Nebula Graph服务。

```
docker-compose down -v
```

2. 删除目录 `nebula-docker-compose` 。

最后更新: 2021年5月26日

2021 Vesoft Inc.

7. 配置和日志

7.1 配置

7.1.1 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。用户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
```

> **Note**
>
> 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，用户也可以使用 `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
```

> **Note**
>
> 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。

最后更新: 2021年5月14日

2021 Vesoft Inc.

7.1.2 Meta服务配置

Meta服务提供了两份初始配置文件 `nebula-metad.conf.default` 和 `nebula-metad.conf.production`，方便在不同场景中使用。文件的默认路径为 `/usr/local/nebula/etc/` 。

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。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。

> **Caution**
>
> 如果修改了配置文件，必须在配置文件开头添加 `--local_config=true` ，并重启服务。否则会从缓存中读取旧的配置。

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。
`timezone_name`	-	指定Nebula Graph的时区。初始配置文件中未设置该参数，如需使用请手动添加。系统默认值为 `UTC+00:00:00` 。格式请参见Specifying the Time Zone with TZ。例如，东八区的设置方式为 `--timezone_name=CST-8` 。

> **Note**
>
> - 在插入时间类型的属性值时，Nebula Graph会根据 `timezone_name` 设置的时区将该时间值转换成相应的UTC时间，因此在查询中返回的时间类型属性值为UTC时间。
> - `timezone_name` 参数只用于转换Nebula Graph中存储的数据，Nebula Graph进程中其它时区相关数据，例如日志打印的时间等，仍然使用主机系统默认的时区。

2021 Vesoft Inc.

logging配置

名称	预设值	说明
`log_dir`	`logs`	存放Meta服务日志的目录，建议和数据保存在不同硬盘。
`minloglevel`	`0`	最小日志级别，即不会记录低于这个级别的日志。可选值为 `0` （INFO）、 `1` （WARNING）、 `2` （ERROR）、 `3` （FATAL）。建议在调试时设置为 `0` ，生产环境中设置为 `1` 。如果设置为 `4` ，Nebula Graph不会记录任何日志。
`v`	`0`	日志详细级别，值越大，日志记录越详细。可选值为 `0` 、 `1` 、 `2` 、 `3` 。
`logbufsecs`	`0`	缓冲日志的最大时间，超时后输出到日志文件。 `0` 表示实时输出。单位：秒。
`redirect_stdout`	`true`	是否将标准输出和标准错误重定向到单独的输出文件。
`stdout_log_file`	`metad-stdout.log`	标准输出日志文件名称。
`stderr_log_file`	`metad-stderr.log`	标准错误日志文件名称。
`stderrthreshold`	`2`	要复制到标准错误中的最小日志级别（ `minloglevel` ）。

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。

Caution

必须在配置文件中使用真实的IP地址。否则某些情况下 `127.0.0.1/0.0.0.0` 无法正确解析。

storage配置

名称	预设值	说明
`data_path`	`data/meta`	meta数据存储路径。

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。

2021 Vesoft Inc.

rocksdb options配置

名称	预设值	说明
`rocksdb_wal_sync`	`true`	是否同步RocksDB的WAL日志。

最后更新: 2021年5月26日

2021 Vesoft Inc.

7.1.3 Graph服务配置

Graph服务提供了两份初始配置文件 `nebula-graphd.conf.default` 和 `nebula-graphd.conf.production`，方便在不同场景中使用。文件的默认路径为 `/usr/local/nebula/etc/` 。

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。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。

> **Caution**
>
> 如果修改了配置文件，必须在配置文件开头添加 `--local_config=true` ，并重启服务。否则会从缓存中读取旧的配置。

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。
`timezone_name`	-	指定Nebula Graph的时区。初始配置文件中未设置该参数，如需使用请手动添加。系统默认值为 `UTC+00:00:00` 。格式请参见Specifying the Time Zone with TZ。例如，东八区的设置方式为 `--timezone_name=CST-8` 。

> **Note**
>
> - 在插入时间类型的属性值时，Nebula Graph会根据 `timezone_name` 设置的时区将该时间值转换成相应的UTC时间，因此在查询中返回的时间类型属性值为UTC时间。
> - `timezone_name` 参数只用于转换Nebula Graph中存储的数据，Nebula Graph进程中其它时区相关数据，例如日志打印的时间等，仍然使用主机系统默认的时区。

2021 Vesoft Inc.

logging配置

名称	预设值	说明
`log_dir`	`logs`	存放Graph服务日志的目录，建议和数据保存在不同硬盘。
`minloglevel`	`0`	最小日志级别，即不会记录低于这个级别的日志。可选值为 `0` （INFO）、 `1` （WARNING）、 `2` （ERROR）、 `3` （FATAL）。建议在调试时设置为 `0` ，生产环境中设置为 `1` 。如果设置为 `4` ，Nebula Graph不会记录任何日志。
`v`	`0`	日志详细级别，值越大，日志记录越详细。可选值为 `0` 、 `1` 、 `2` 、 `3` 。
`logbufsecs`	`0`	缓冲日志的最大时间，超时后输出到日志文件。 `0` 表示实时输出。单位：秒。
`redirect_stdout`	`true`	是否将标准输出和标准错误重定向到单独的输出文件。
`stdout_log_file`	`graphd-stdout.log`	标准输出日志文件名称。
`stderr_log_file`	`graphd-stderr.log`	标准错误日志文件名称。
`stderrthreshold`	`2`	要复制到标准错误中的最小日志级别（ `minloglevel` ）。

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。

2021 Vesoft Inc.

networking配置

名称	预设值	说明
`meta_server_addrs`	`127.0.0.1:9559`	全部Meta服务的IP地址和端口。多个Meta服务用英文逗号（,）分隔。
`local_ip`	`127.0.0.1`	Graph服务的本地IP地址。本地IP地址用于识别nebula-graphd进程，如果是分布式集群或需要远程访问，请修改为对应地址。
`listen_netdev`	`any`	监听的网络设备。
`port`	`9669`	Graph服务的RPC守护进程监听端口。
`reuse_port`	`false`	是否启用SO_REUSEPORT。
`listen_backlog`	`1024`	socket监听的连接队列最大长度，调整本参数需要同时调整 `net.core.somaxconn` 。
`client_idle_timeout_secs`	`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。

> **Caution**
>
> 必须在配置文件中使用真实的IP地址。否则某些情况下 `127.0.0.1/0.0.0.0` 无法正确解析。

charset and collate配置

名称	预设值	说明
`default_charset`	`utf8`	创建图空间时的默认字符集。
`default_collate`	`utf8_bin`	创建图空间时的默认排序规则。

authorization配置

名称	预设值	说明
`enable_authorize`	`false`	用户登录时是否进行身份验证。身份验证详情请参见身份验证。
`auth_type`	`password`	用户登录的身份验证方式。取值为 `password` 、 `ldap` 、 `cloud` 。

最后更新: 2021年5月26日

2021 Vesoft Inc.

7.1.4 Storage服务配置

Storage服务提供了两份初始配置文件 `nebula-storaged.conf.default` 和 `nebula-storaged.conf.production`，方便在不同场景中使用。文件的默认路径为 `/usr/local/nebula/etc/`。

> **Note**
>
> Raft Listener的配置和Storage服务配置不同，详情请参见部署Raft 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。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。

> **Caution**
>
> 如果修改了配置文件，必须在配置文件开头添加 `--local_config=true`，并重启服务。否则会从缓存中读取旧的配置。

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。
`timezone_name`	-	指定Nebula Graph的时区。初始配置文件中未设置该参数，如需使用请手动添加。系统默认值为 `UTC+00:00:00`。格式请参见Specifying the Time Zone with TZ。例如，东八区的设置方式为 `--timezone_name=CST-8`。

> **Note**
>
> - 在插入时间类型的属性值时，Nebula Graph会根据 `timezone_name` 设置的时区将该时间值转换成相应的UTC时间，因此在查询中返回的时间类型属性值为UTC时间。
> - `timezone_name` 参数只用于转换Nebula Graph中存储的数据，Nebula Graph进程中其它时区相关数据，例如日志打印的时间等，仍然使用主机系统默认的时区。

2021 Vesoft Inc.

logging配置

名称	预设值	说明
`log_dir`	`logs`	存放Storage服务日志的目录，建议和数据保存在不同硬盘。
`minloglevel`	`0`	最小日志级别，即不会记录低于这个级别的日志。可选值为 `0` （INFO）、 `1` （WARNING）、 `2` （ERROR）、 `3` （FATAL）。建议在调试时设置为 `0` ，生产环境中设置为 `1` 。如果设置为 `4` ，Nebula Graph不会记录任何日志。
`v`	`0`	日志详细级别，值越大，日志记录越详细。可选值为 `0` 、 `1` 、 `2` 、 `3` 。
`logbufsecs`	`0`	缓冲日志的最大时间，超时后输出到日志文件。 `0` 表示实时输出。单位：秒。
`redirect_stdout`	`true`	是否将标准输出和标准错误重定向到单独的输出文件。
`stdout_log_file`	`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。

Caution

必须在配置文件中使用真实的IP地址。否则某些情况下 `127.0.0.1/0.0.0.0` 无法正确解析。

raft配置

名称	预设值	说明
`raft_heartbeat_interval_secs`	`30`	Raft选举超时时间。单位：秒。
`raft_rpc_timeout_ms`	`500`	Raft客户端的远程过程调用（RPC）超时时间。单位：毫秒。
`wal_ttl`	`14400`	Raft WAL的生存时间。单位：秒。

2021 Vesoft Inc.

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。

2021 Vesoft Inc.

`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。

最后更新: 2021年5月26日

2021 Vesoft Inc.

7.1.5 Linux 内核配置

本文介绍与Nebula Graph相关的Linux内核配置，并介绍如何修改配置。

资源控制

ULIMIT注意事项

命令 `ulimit` 用于为当前shell会话设置资源阈值，注意事项如下：

- `ulimit` 所做的更改仅对当前会话或子进程生效。
- 资源的阈值（软阈值）不能超过硬阈值。
- 普通用户不能使用命令调整硬阈值，即使使用 `sudo` 也不能调整。
- 修改系统级别或调整硬性阈值，请编辑文件 `/etc/security/limits.conf` 。这种方式需要重新登录才生效。

ULIMIT -C

`ulimit -c` 用于限制core文件的大小，建议设置为 `unlimited` ，命令如下：

```
ulimit -c unlimited
```

ULIMIT -N

`ulimit -n` 用于限制打开文件的数量，建议设置为超过10万，例如：

```
ulimit -n 130000
```

内存

VM.SWAPPINESS

`vm.swappiness` 是触发虚拟内存（swap）的空闲内存百分比。值越大，使用swap的可能性就越大，建议设置为0，表示首先删除页缓存。需要注意的是，0表示**尽量**不使用swap。

VM.MIN_FREE_KBYTES

`vm.min_free_kbytes` 用于设置Linux虚拟机保留的最小空闲千字节数。如果系统内存足够，建议设置较大值。例如物理内存为128 GB，可以将 `vm.min_free_kbytes` 设置为5 GB。如果值太小，会导致系统无法申请足够的连续物理内存。

VM.MAX_MAP_COUNT

`vm.max_map_count` 用于限制单个进程的VMA（虚拟内存区域）数量。默认值为 `65530` ，对于绝大多数应用程序来说已经足够。如果应用程序因为内存消耗过大而报错，请增大本参数的值。

VM.DIRTY_*

`vm.dirty_*` 是一系列控制系统脏数据缓存的参数。对于写密集型场景，用户可以根据需要进行调整（吞吐量优先或延迟优先），建议使用系统默认值。

TRANSPARENT HUGE PAGE

为了降低延迟，用户必须关闭THP（transparent huge page）。命令如下：

```
root# echo never > /sys/kernel/mm/transparent_hugepage/enabled
root# echo never > /sys/kernel/mm/transparent_hugepage/defrag
root# swapoff -a && swapon -a
```

网络

NET.IPV4.TCP_SLOW_START_AFTER_IDLE

`net.ipv4.tcp_slow_start_after_idle` 默认值为1，会导致闲置一段时间后拥塞窗口超时，建议设置为 `0` ，尤其适合大带宽高延迟场景。

2021 Vesoft Inc.

NET.CORE.SOMAXCONN

`net.core.somaxconn` 用于限制socket监听的连接队列数量。默认值为 `128` 。对于有大量突发连接的场景，建议设置为不低于 `1024` 。

NET.IPV4.TCP_MAX_SYN_BACKLOG

`net.ipv4.tcp_max_syn_backlog` 用于限制处于SYN_RECV（半连接）状态的TCP连接数量。默认值为 `128` 。对于有大量突发连接的场景，建议设置为不低于 `1024` 。

NET.CORE.NETDEV_MAX_BACKLOG

`net.core.netdev_max_backlog` 用于限制队列中数据包的数量。默认值为 `1000` ，建议设置为 `10000` 以上，尤其是万兆网卡。

NET.IPV4.TCP_KEEPALIVE_*

`net.ipv4.tcp_keepalive_*` 是一系列保持TCP连接存活的参数。对于使用四层透明负载均衡的应用程序，如果空闲连接异常断开，请增大 `tcp_keepalive_time` 和 `tcp_keepalive_intvl` 的值。

NET.IPV4.TCP_WMEM/RMEM

TCP套接字发送/接收缓冲池的最小、最大、默认空间。对于大连接，建议设置为 `带宽(GB)*往返时延(ms)` 。

SCHEDULER

对于SSD设备，建议将 `scheduler` 设置为 `noop` 或者 `none` ，路径为 `/sys/block/DEV_NAME/queue/scheduler` 。

其他参数

KERNEL.CORE_PATTERN

建议设置为 `core` ，并且将 `kernel.core_uses_pid` 设置为 `1` 。

修改参数

SYSCTL命令

- `sysctl <conf_name>`

  查看当前参数值。 - `sysctl -w <conf_name>=<value>`

  临时修改参数值，立即生效，重启后恢复原值。 - `sysctl -p [<file_path>]`

  从指定配置文件里加载Linux系统参数，默认从 `/etc/sysctl.conf` 加载。

PRLIMIT

命令 `prlimit` 可以获取和设置进程资源的限制，结合 `sudo` 可以修改硬阈值，例如， `prlimit --nofile=140000 --pid=$$` 调整当前进程允许的打开文件的最大数量为 `140000` ，立即生效，此命令仅支持RedHat 7u或更高版本。

最后更新: 2021年5月14日

2021 Vesoft Inc.

7.2 日志

7.2.1 日志配置

Nebula Graph使用glog打印日志，使用gflags控制日志级别，并在运行时通过HTTP接口动态修改日志级别，方便跟踪问题。

日志目录

日志的默认目录为 `/usr/local/nebula/logs/` 。

如果在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。
`value`	日志级别，可选值请参见配置说明。
`ws_ip`	HTTP服务的IP地址，可以在配置文件中查看。默认值为 `127.0.0.1` 。
`ws_port`	HTTP服务的端口，可以在配置文件中查看。默认值分别为 `19559` （Meta）、 `19669` （Graph） `19779` （Storage）。

2021 Vesoft Inc.

示例如下：

```
$ curl -X PUT -H "Content-Type: application/json" -d '{"minloglevel":0,"v":3}' "127.0.0.1:19779/flags" # storaged
$ curl -X PUT -H "Content-Type: application/json" -d '{"minloglevel":0,"v":3}' "127.0.0.1:19669/flags" # graphd
$ curl -X PUT -H "Content-Type: application/json" -d '{"minloglevel":0,"v":3}' "127.0.0.1:19559/flags" # metad
```

如果在Nebula Graph运行时修改了日志级别，重启服务后会恢复为配置文件中设置的级别，如果需要永久修改，请修改配置文件。

RocksDB 日志

RocksDB 的日志通常在 `/usr/local/nebula/data/storage/nebula/$id/data/LOG`，其中 `$id` 为实例号。该日志通常用于调试 RocksDB 参数。

最后更新: 2021年5月18日

2021 Vesoft Inc.

8. 监控

8.1 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。

8.1.1 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` 。不同的Nebula Graph服务支持查询的监控指标也不同。指标结构的说明如下。

类别	示例	说明
指标名称	`num_queries`	简单描述指标的含义。
统计类型	`sum`	指标统计的方法。当前支持SUM、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。

8.1.2 通过HTTP端口查询监控指标

语法

```
curl -G "http://<ip>:<port>/stats?stats=<metric_name_list> [&format=json]"
```

选项	说明
`ip`	服务器的IP地址，可以在安装目录内查看配置文件获取。
`port`	服务器的HTTP端口，可以在安装目录内查看配置文件获取。默认情况下，Meta服务端口为19559，Graph服务端口为19669，Storage服务端口为19779。
`metric_name_list`	监控指标名称，多个监控指标用英文逗号（,）隔开。
`&format=json`	将结果以JSON格式返回。

> **Note**
>
> 如果Nebula Graph服务部署在容器中，需要执行 `docker-compose ps` 命令查看映射到容器外部的端口，然后通过该端口查询。

示例

- 查询单个监控指标

  查询Graph服务中，最近10分钟的请求总数。

2021 Vesoft Inc.

```
$ 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
```

最后更新: 2021年5月20日

2021 Vesoft Inc.

9. 数据安全

9.1 验证和授权

9.1.1 身份验证

身份验证用于将会话映射到特定用户，从而实现访问控制。

当客户端连接到Nebula Graph时，Nebula Graph会创建一个会话，会话中存储连接的各种信息，如果开启了身份验证，就会将会话映射到对应的用户。

> **Note**
>
> 默认情况下，身份验证功能是关闭的，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。

> **Note**
>
> 开启身份验证后，默认的God角色账号为 `root` ，密码为 `nebula` 。角色详情请参见内置角色权限。

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。

最后更新: 2021年5月25日

2021 Vesoft Inc.

9.1.2 用户管理

用户管理是Nebula Graph访问控制中不可或缺的组成部分，本文将介绍用户管理的相关语法。

开启身份验证后，用户需要使用已创建的用户才能连接Nebula Graph，而且连接后可以进行的操作也取决于该用户拥有的角色权限。

> **Note**
>
> - 默认情况下，身份验证功能是关闭的，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。

- 语法

```
GRANT ROLE <role_type> ON <space_name> TO <user_name>;
```

- 示例

```
nebula> GRANT ROLE USER ON basketballplayer 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。

- 语法

```
REVOKE ROLE <role_type> ON <space_name> FROM <user_name>;
```

- 示例

```
nebula> REVOKE ROLE USER ON basketballplayer FROM user1;
```


2021 Vesoft Inc.

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。

> **Note**
>
> 删除用户不会自动断开该用户当前会话，而且权限仍在当前会话中生效。

- 语法

```
DROP USER [IF EXISTS] <user_name>;
```

- 示例

```
nebula> DROP USER user1;
```

查看用户列表（SHOW USERS）

执行 SHOW USERS 语句可以查看用户列表。当前仅**God**角色用户能够执行 SHOW USERS 语句。

- 语法

```
SHOW USERS;
```

- 示例

```
nebula> SHOW USERS;
+-----------+
| Account   |
+-----------+
| "test1"   |
+-----------+
| "test2"   |
+-----------+
| "test3"   |
+-----------+
```

最后更新: 2021年5月14日

2021 Vesoft Inc.

9.1.3 内置角色权限

所谓角色，就是一组相关权限的集合。用户可以把角色分配给创建的用户，从而实现访问控制。

内置角色

Nebula Graph内置了多种角色，说明如下：

- God
  - 初始最高权限角色，拥有**所有操作**的权限。类似于Linux中的 `root` 和Windows中的 `administrator` 。
  - Meta服务初始化时，会自动创建God角色用户 `root` ，密码为 `nebula` 。

    > **注意**
    >
    > 请及时修改 `root` 用户的密码，保证数据安全。

  - 一个集群只能有一个God角色用户，该用户可以管理集群内所有图空间。
  - 不支持手动授权God角色，只能使用默认God角色用户 `root` 。
- Admin
  - 对权限内的图空间拥有Schema和data的**读写**权限。
  - 可以将权限内的图空间授权给其他用户。

    > **注意**
    >
    > 只能授权低于ADMIN级别的角色给其他用户。

- DBA
  - 对权限内的图空间拥有Schema和data的**读写**权限。
  - 无法将权限内的图空间授权给其他用户。
- User
  - 对权限内的图空间拥有Schema的**只读**权限。
  - 对权限内的图空间拥有data的**读写**权限。
- Guest
  - 对权限内的图空间拥有Schema和data的**只读**权限。

> **Note**
>
> - 不支持自行创建角色，只能使用默认的内置角色。
> - 一个用户在一个图空间内只能拥有一个角色权限。授权用户请参见用户管理。

2021 Vesoft Inc.

角色权限

各角色的执行权限如下。

权限	God	Admin	DBA	User	Guest	相关语句
Read space	Y	Y	Y	Y	Y	`USE`、`DESCRIBE SPACE`
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。

最后更新: 2021年5月24日

2021 Vesoft Inc.

9.2 管理快照

Nebula Graph提供快照（snapshot）功能，用于保存集群当前时间点的数据状态，当出现数据丢失或误操作时，可以通过快照恢复数据。

9.2.1 前提条件

Nebula Graph的身份认证功能默认是关闭的，此时任何用户都能使用快照功能。

如果身份认证开启，仅God角色用户可以使用快照功能。关于角色说明，请参见内置角色权限。

9.2.2 注意事项

- 系统结构发生变化后，建议立刻创建快照，例如在 `add host`、 `drop host`、 `create space`、 `drop space`、 `balance` 等操作之后。
- 不支持自动回收创建失败的快照垃圾文件，需要手动删除。
- 不支持指定快照保存路径，默认路径为 `/usr/local/nebula/data`。

9.2.3 快照路径

Nebula Graph创建的快照以目录的形式存储，例如 `SNAPSHOT_2021_03_09_08_43_12`，后缀 `2021_03_09_08_43_12` 根据创建时间（UTC）自动生成。

创建快照时，快照目录会自动在leader Meta服务器和所有Storage服务器的目录 `checkpoints` 内创建。

为了快速定位快照所在路径，可以使用Linux命令 `find`。例如：

```
$ find |grep 'SNAPSHOT_2021_03_09_08_43_12'
./data/meta2/nebula/0/checkpoints/SNAPSHOT_2021_03_09_08_43_12
./data/meta2/nebula/0/checkpoints/SNAPSHOT_2021_03_09_08_43_12/data
./data/meta2/nebula/0/checkpoints/SNAPSHOT_2021_03_09_08_43_12/data/000081.sst
...
```

9.2.4 创建快照

命令 `CREATE SNAPSHOT` 可以创建集群当前时间点的快照。只支持创建所有图空间的快照，不支持创建指定图空间的快照。

> **Note**
>
> 如果快照创建失败，请删除快照重新创建。

```
nebula> CREATE SNAPSHOT;
```

9.2.5 查看快照

命令 `SHOW SNAPSHOTS` 可以查看集群中的所有快照。

```
nebula> SHOW SNAPSHOTS;
+-------------------------------+---------+-----------------+
| Name                          | Status  | Hosts           |
+-------------------------------+---------+-----------------+
| "SNAPSHOT_2021_03_09_08_43_12" | "VALID" | "127.0.0.1:9779" |
+-------------------------------+---------+-----------------+
| "SNAPSHOT_2021_03_09_09_10_52" | "VALID" | "127.0.0.1:9779" |
+-------------------------------+---------+-----------------+
```


2021 Vesoft Inc.

参数说明如下：

参数	说明
`Name`	快照名称，前缀为 `SNAPSHOT`，表示该文件为快照文件，后缀为快照创建的时间点（UTC时间）。
`Status`	快照状态。 `VALID` 表示快照有效， `INVALID` 表示快照无效。
`Hosts`	创建快照时所有Storage服务器的IP地址和端口。

9.2.6 删除快照

命令 `DROP SNAPSHOT` 可以删除指定的快照，语法为：

```
DROP SNAPSHOT <snapshot_name>;
```

示例如下：

```
nebula> DROP SNAPSHOT SNAPSHOT_2021_03_09_08_43_12;
nebula> SHOW SNAPSHOTS;
+-------------------------------+---------+------------------+
| Name                          | Status  | Hosts            |
+-------------------------------+---------+------------------+
| "SNAPSHOT_2021_03_09_09_10_52" | "VALID" | "127.0.0.1:9779" |
+-------------------------------+---------+------------------+
```

9.2.7 恢复快照

当前未提供恢复快照命令，需要手动拷贝快照文件到对应的文件夹内，也可以通过shell脚本进行操作。实现逻辑如下：

1. 创建快照后，会在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，用户可以找到合适的时间点快照，将内部的文件夹 `data` 和 `wal` 分别拷贝到各自的上级目录（和 `checkpoints` 平级），覆盖之前的 `data` 和 `wal` ，然后重启集群即可。

最后更新: 2021年5月26日

2021 Vesoft Inc.

10. 调整服务

10.1 Compaction

本文介绍Compaction的相关信息。

Nebula Graph中，`Compaction`是最重要的后台操作，对性能有极其重要的影响。

`Compaction`操作会读取硬盘上的数据，然后重组数据结构和索引，然后再写回硬盘，可以成倍提升读取性能。将大量数据写入Nebula Graph后，为了提高读取性能，需要手动触发`Compaction`操作（全量`Compaction`）。

> **Note**
>
> `Compaction`操作会长时间占用硬盘的IO，建议在业务低峰期（例如凌晨）执行该操作。

Nebula Graph有两种类型的`Compaction`操作：自动`Compaction`和全量`Compaction`。

10.1.1 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，用户可以关闭自动`Compaction`。

关闭自动Compaction

> **Danger**
>
> 命令`UPDATE CONFIGS`会将未设置的参数恢复为默认值，因此修改前需要使用`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"} |
+-----------+-------------------------------------+-------+-----------
+-----------------------------------------------------------------------------------------------------------------------+
...
```


2021 Vesoft Inc.

10.1.2 全量Compaction

全量 `Compaction` 可以对图空间进行大规模后台操作，例如合并文件、删除TTL过期数据等，该操作需要手动发起。使用如下语句执行全量 `Compaction` 操作：

> **Note**
>
> 建议在业务低峰期（例如凌晨）执行该操作，避免大量占用硬盘IO影响业务。

```
nebula> USE <your_graph_space>;
nebula> SUBMIT JOB COMPACT;
```

上述命令会返回作业的ID，用户可以使用如下命令查看 `Compaction` 状态：

```
nebula> SHOW JOB <job_id>;
```

10.1.3 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` 。
- 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)
  ```

10.1.4 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？

如果已经设置读写速率限制，例如 `rate_limit` 限制为20MB/S时，用户可以通过 `硬盘使用量/rate_limit` 预估需要耗费的时间。如果没有设置读写速率限制，根据经验，速率大约为50MB/S。

可以动态调整rate_limit吗？

不可以。

全量Compaction操作开始后可以停止吗？

不可以停止，必须等待操作完成。这是RocksDB的限制。

最后更新: 2021年5月14日

2021 Vesoft Inc.

10.2 Storage负载均衡

用户可以使用 `BALANCE` 语句平衡分片和Raft leader的分布，或者删除冗余的Storage服务器。

10.2.1 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。

10.2.2 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。

> **Danger**
>
> 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。

2021 Vesoft Inc.

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` 检查分片的分布。

```
nebual> SHOW HOSTS;
+-------------+------+----------+--------------+----------------------------------+------------------------+
| Host        | Port | Status   | Leader count | Leader distribution              | Partition distribution |
+-------------+------+----------+--------------+----------------------------------+------------------------+
| "storaged0" | 9779 | "ONLINE" | 4            | "basketballplayer:4"             | "basketballplayer:15"  |
+-------------+------+----------+--------------+----------------------------------+------------------------+
| "storaged1" | 9779 | "ONLINE" | 8            | "basketballplayer:8"             | "basketballplayer:15"  |
+-------------+------+----------+--------------+----------------------------------+------------------------+
| "storaged2" | 9779 | "ONLINE" | 3            | "basketballplayer:3"             | "basketballplayer:15"  |
+-------------+------+----------+--------------+----------------------------------+------------------------+
| "storaged3" | 9779 | "ONLINE" | 0            | "No valid partition"             | "No valid partition"   |
+-------------+------+----------+--------------+----------------------------------+------------------------+
| "storaged4" | 9779 | "ONLINE" | 0            | "No valid partition"             | "No valid partition"   |
+-------------+------+----------+--------------+----------------------------------+------------------------+
| "Total"     |      |          | 15           | "basketballplayer:15"            | "basketballplayer: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。

> **Note**
>
> `BALANCE DATA` 不会均衡leader的分布。均衡leader请参见均衡leader分布。

```
nebula> SHOW HOSTS;
+-------------+------+----------+--------------+----------------------------------+------------------------+
| Host        | Port | Status   | Leader count | Leader distribution              | Partition distribution |
+-------------+------+----------+--------------+----------------------------------+------------------------+
| "storaged0" | 9779 | "ONLINE" | 4            | "basketballplayer:4"             | "basketballplayer:9"   |
+-------------+------+----------+--------------+----------------------------------+------------------------+
| "storaged1" | 9779 | "ONLINE" | 8            | "basketballplayer:8"             | "basketballplayer:9"   |
+-------------+------+----------+--------------+----------------------------------+------------------------+
| "storaged2" | 9779 | "ONLINE" | 3            | "basketballplayer:3"             | "basketballplayer:9"   |
+-------------+------+----------+--------------+----------------------------------+------------------------+
| "storaged3" | 9779 | "ONLINE" | 0            | "No valid partition"             | "basketballplayer:9"   |
+-------------+------+----------+--------------+----------------------------------+------------------------+
| "storaged4" | 9779 | "ONLINE" | 0            | "No valid partition"             | "basketballplayer:9"   |
+-------------+------+----------+--------------+----------------------------------+------------------------+
| "Total"     |      |          | 15           | "basketballplayer:15"            | "basketballplayer: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社区寻求帮助。

2021 Vesoft Inc.

10.2.3 停止负载均衡任务

停止负载均衡任务，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。用户可以执行命令 `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，用户可以再次执行命令 `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。

10.2.4 重置负载均衡任务

如果停止负载均衡任务后重新执行仍然失败，可以尝试用命令 `BALANCE DATA RESET PLAN` 重置负载均衡任务，该操作会清空旧的任务。之后再使用 `BALANCE DATA` 命令，会新建负载均衡任务，而不是执行旧的任务。

10.2.5 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	9779
storage4	192.168.0.9	9779

请执行如下命令：

```
BALANCE DATA REMOVE 192.168.0.8:9779,192.168.0.9:9779;
```

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。

> **Note**
>
> 已下线节点状态会显示为OFFLINE.

10.2.6 均衡leader分布

`BALANCE DATA` 只能均衡分片分布，不能均衡Raft leader分布。用户可以使用命令 `BALANCE LEADER` 均衡leader分布。

示例

```
nebula> BALANCE LEADER;
```

用户可以执行 `SHOW HOSTS` 检查结果。

```
nebula> SHOW HOSTS;
+-------------+------+----------+--------------+----------------------------------+------------------------+
| Host        | Port | Status   | Leader count | Leader distribution              | Partition distribution |
+-------------+------+----------+--------------+----------------------------------+------------------------+
| "storaged0" | 9779 | "ONLINE" | 3            | "basketballplayer:3"             | "basketballplayer:9"   |
```


2021 Vesoft Inc.

```
+-------------+------+----------+--------------+----------------------------------+------------------------+
| "storaged1" | 9779 | "ONLINE" | 3            | "basketballplayer:3"             | "basketballplayer:9"   |
+-------------+------+----------+--------------+----------------------------------+------------------------+
| "storaged2" | 9779 | "ONLINE" | 3            | "basketballplayer:3"             | "basketballplayer:9"   |
+-------------+------+----------+--------------+----------------------------------+------------------------+
| "storaged3" | 9779 | "ONLINE" | 3            | "basketballplayer:3"             | "basketballplayer:9"   |
+-------------+------+----------+--------------+----------------------------------+------------------------+
| "storaged4" | 9779 | "ONLINE" | 3            | "basketballplayer:3"             | "basketballplayer:9"   |
+-------------+------+----------+--------------+----------------------------------+------------------------+
| "Total"     |      |          | 15           | "basketballplayer:15"            | "basketballplayer:45"  |
+-------------+------+----------+--------------+----------------------------------+------------------------+
```

最后更新: 2021年5月26日

2021 Vesoft Inc.

11. Nebula Graph Studio

11.1 认识 Nebula Graph Studio

11.1.1 什么是 Nebula Graph Studio

Nebula Graph Studio（简称 Studio）是一款可以通过 Web 访问的图数据库可视化工具，搭配 Nebula Graph 内核使用，提供构图、数据导入、编写 nGQL 查询、图探索等一站式服务。即使没有图数据库操作经验，用户也可以快速成为图专家。

发行版本

Studio 目前有三个发行版本：

- Docker 版本：用户可以使用 Docker 服务部署 Studio，并连接到 Nebula Graph 数据库。详细信息参考 部署 Studio。
- RPM 版本：用户可以使用 RPM 服务部署 Studio，并连接到 Nebula Graph 数据库。详细信息参考 部署 Studio。
- 云服务版本：用户可以在 Nebula Graph Cloud Service 上创建 Nebula Graph 数据库实例，并一键直连云服务版 Studio。详细信息参考 Nebula Graph Cloud Service 用户手册。

三个发行版本功能基本相同。但是，因为部署方式不同，会有不同的使用限制。详细信息，参考 使用限制。

产品功能

Studio 具备以下功能：

- GUI 设计，方便管理 Nebula Graph 图数据：
  - 使用 **Schema** 管理功能，用户可以使用图形界面完成 Schema（模式）创建，快速上手 Nebula Graph。
  - 使用 **控制台** 功能，用户可以使用 nGQL 语句创建 Schema，并对数据执行增删改查操作。
  - 使用 **导入** 功能，通过简单的配置，用户即能批量导入点和边数据，并能实时查看数据导入日志。
- 图探索，支持可视化展示图数据，使更容易发现数据之间的关联性，提高数据分析和解读的效率。

适用场景

如果有以下任一需求，都可以使用 Studio：

- 有一份数据集，想进行可视化图探索或者数据分析。用户可以使用 Docker Compose 或者 Nebula Graph Cloud Service 部署 Nebula Graph，再使用 Studio 完成可视化操作。
- 已经安装部署了 Nebula Graph 数据库，并且已经导入数据集，想使用 GUI 工具执行 nGQL 语句查询、可视化图探索或者数据分析。
- 刚开始学习 nGQL（Nebula Graph Query Language），但是不习惯用命令行工具，更希望使用 GUI 工具查看语句输出的结果。

身份验证

对于云服务版 Studio，只有 Nebula Graph 实例的创建者以及被授予操作权限的 Nebula Graph Cloud Service 用户可以登录 Studio。详细信息参考 Nebula Graph Cloud Service 用户手册。

对于 Docker 版和 RPM Studio，因为 Nebula Graph 默认不启用身份验证，所以，一般情况下用户可以使用默认账号和密码（`user` 和 `password`）登录 Studio。当 Nebula Graph 启用了身份验证后，用户只能使用指定的账号和密码登录 Studio。关于 Nebula Graph 的身份验证功能，参考 Nebula Graph 用户手册。

最后更新: 2021年5月19日

2021 Vesoft Inc.

11.1.2 名词解释

本文提供了在使用 Studio 时可能需要知道的名词解释。

- Nebula Graph Studio：在本手册中简称为 Studio，是一款可以通过 Web 访问的图数据库可视化工具，搭配 Nebula Graph DBMS 使用，提供构图、数据导入、编写 nGQL 查询、图探索等一站式服务。
- Nebula Graph：一款开源图数据库管理系统（Graph Database Management System），擅长处理千亿个点和万亿条边的超大规模数据集。
- Nebula Graph Cloud Service：Nebula Graph 图数据库的云服务平台（Database-as-a-Service，DBaaS），按需付费，支持一键部署 Nebula Graph，集成了 Nebula Graph Studio，并内置资源监控工具。详细信息，参考 Nebula Graph Cloud Service 用户手册。

最后更新: 2021年5月20日

2021 Vesoft Inc.

11.1.3 使用限制

本文描述了在使用 Studio 时可能会受到的限制。

Nebula Graph 版本支持

目前 Studio v1.x 仅支持 Nebula Graph v1.x，Studio v2.x 仅支持 Nebula Graph v2.x。

系统架构

Docker 版和 RPM 版 Studio 目前仅支持 x86_64 架构。

数据上传

使用云服务版 Studio 上传数据有以下限制：

- 目前仅支持上传没有表头行的 CSV 文件，而且仅支持英文逗号（,）作为分隔符。
- 单个文件不得超过 100 MB。
- 单个实例上传文件总量不得超过 1 GB。
- 单个文件仅能保存 1 天。

使用 Docker 版和 RPM 版 Studio 上传数据也仅支持上传无表头的 CSV 文件，但是，单个文件大小及保存时间不受限制，而且，数据总量以本地存储容量为准。

数据备份

目前仅支持在 **控制台** 上以 CSV 格式导出查询结果，不支持其他数据备份方式。

nGQL 支持

使用 Docker 版 Studio 时，除以下内容外，用户可以在 **控制台** 上执行所有 nGQL 语句：

- `USE <space_name>`：只能在 **Space** 下拉列表中选择图空间，不能运行这个语句选择图空间。
- **控制台** 上使用 nGQL 语句时，用户可以直接回车换行，不能使用换行符。

使用云服务版 Studio 时，除以上限制外，用户也不能在 **控制台** 上执行用户管理和角色管理相关的语句，包括：

- `CREATE USER`
- `ALTER USER`
- `CHANGE PASSWORD`
- `DROP USER`
- `GRANT ROLE`
- `REVOKE ROLE`

关于语句的详细信息，参考 Nebula Graph 用户手册。

浏览器支持

建议使用最新版本的 Chrome 访问 Studio。

最后更新: 2021年5月24日

2021 Vesoft Inc.

11.1.4 版本更新

Studio 处于持续开发状态中。用户可以通过 Studio 发布版本更新 查看最新发布的功能。

云服务版 Studio

对于云服务版 Studio，以 Nebula Graph Cloud Service 上实际部署的版本为准，用户不能自行更新 Studio 版本。当前公测环境里的 Studio 版本为 v1.1.1-beta。

Docker 版和 RPM 版 Studio

对于 Docker 版和 RPM 版 Studio，建议每次都在 `nebula-web-docker` 目录下运行以下命令启动 Studio：

```
docker-compose pull && docker-compose up -d
```

成功连接 Docker 版 Studio 后，用户可以在页面右上角点击版本号，再点击 **新发布**，前往查看 Studio 的版本更新记录。

最后更新: 2021年5月25日

2021 Vesoft Inc.

11.1.5 快捷键

本文列出了 Studio 支持的快捷键。

要完成的任务	操作
在 **控制台** 页面运行 nGQL 语句	按 Shift + Enter 键
在 **图探索** 页面选中多个点	按 Shift 键 + 鼠标单击
在 **图探索** 页面缩小图	按 Shift + '-' 键
在 **图探索** 页面放大图	按 Shift + '+' 键
在 **图探索** 页面显示图	按 Shift + 'l' 键
在 **图探索** 页面撤销操作	按 Shift + 'z' 键
在 **图探索** 页面删除图	选中后按 Shift + 'del' 键
在 **图探索** 页面对某个点快速拓展	鼠标双击 或 按 Shift + Enter 键

最后更新: 2021年5月24日

2021 Vesoft Inc.

11.2 安装与登录

11.2.1 部署 Studio

Nebula Graph Studio（以下简称 Studio）支持云端或本地部署。云服务版 Studio 只能在 Nebula Graph Cloud Service 上使用。当在 Nebula Graph Cloud Service 上创建 Nebula Graph 实例时即自动完成云服务版本 Studio 的部署，一键直连即可使用，不需要自己部署。详细信息参考《Nebula Graph Cloud Service 用户手册》。本文介绍如何在本地通过 Docker 和 RPM 部署 Studio。

Docker 部署 Studio

前提条件

在部署 Docker 版 Studio 之前，用户需要确认以下信息：

- Nebula Graph 服务已经部署并启动。详细信息，参考Nebula Graph安装部署。

> **Note**
>
> 用户可以使用Docker Compose或RPM方式部署并启动 Nebula Graph 服务。如果刚开始使用 Nebula Graph，建议使用 Docker Compose 部署 Nebula Graph。详细信息参考 使用 Docker Compose 部署 Nebula Graph。

- 在即将运行 Docker 版 Studio 的机器上安装并启动 Docker Compose。详细信息参考 Docker Compose 文档。
- 确保在安装开始前，以下端口处于未被使用状态。

端口号	说明
7001	Studio提供的web服务
8080	Nebula-http-gateway，Client的HTTP服务
5699	Nebula importer文件导入工具，数据导入服务

- （可选）在中国大陆从 Docker Hub 拉取 Docker 镜像的速度可能比较慢，用户可以使用 `registry-mirrors` 参数配置加速镜像。例如，如果要使用 Docker 中国区官方镜像、网易镜像和中国科技大学的镜像，则按以下格式配置 `registry-mirrors` 参数：

```
{
"registry-mirrors": [
  "https://registry.docker-cn.com",
  "http://hub-mirror.c.163.com",
  "https://docker.mirrors.ustc.edu.cn"
  ]
}
```

配置文件的路径和方法因操作系统和/或 Docker Desktop 版本而异。详细信息参考 Docker Daemon 配置文档。

2021 Vesoft Inc.

操作步骤

在命令行工具中按以下步骤依次运行命令，部署并启动 Docker 版 Studio，这里我们用 Nebula Graph 版本为 2.x 的进行演示：

1. 下载 Studio 的部署配置文件。

安装包	Nebula Graph版本
nebula-graph-studio-v1.gz	1.x
nebula-graph-studio-v2.gz	2.x

2. 解压后进入 `nebula-graph-studio-v2` 目录。

```
cd nebula-graph-studio-v2
```

3. 拉取 Studio 的 Docker 镜像。

```
docker-compose pull
```

4. 构建并启动 Studio 服务。其中，`-d` 表示在后台运行服务容器。

```
docker-compose up -d
```

当屏幕返回以下信息时，表示 Docker 版 Studio 已经成功启动。

```
Creating docker_importer_1 ... done
Creating docker_client_1   ... done
Creating docker_web_1      ... done
Creating docker_nginx_1    ... done
```

5. 启动成功后，在浏览器地址栏输入 `http://ip address:7001`。

> **Note**
>
> 在运行 Docker 版 Studio 的机器上，用户可以运行 `ifconfig` 或者 `ipconfig` 获取本机 IP 地址。如果使用这台机器访问 Studio，可以在浏览器地址栏里输入 `http://localhost:7001`。

如果在浏览器窗口中能看到以下登录界面，表示已经成功部署并启动 Studio。

2021 Vesoft Inc.

RPM 部署Studio

前提条件

在部署 RPM 版 Studio 之前，用户需要确认以下信息：

- Nebula Graph 服务已经部署并启动。详细信息，参考Nebula Graph 安装部署。

> **Note**
>
> 用户可以使用Docker Compose或RPM方式部署并启动 Nebula Graph 服务。如果刚开始使用 Nebula Graph，建议使用 Docker Compose 部署 Nebula Graph。详细信息参考 使用 Docker Compose 部署 Nebula Graph。

- 使用的 Linux 发行版为 CentOS ，安装有 losf 和版本为 v10.16.0 + 以上的 Node.js。

> **Note**
>
> `node` 及 `npm` 命令需要安装在 `/usr/bin/` 目录下，以防出现 RPM 安装时 node 命令找不到的情况。 如果依赖安装于用户个人目录下，如 `/usr/local` ，用户可以使用以下命令建立软连接：
>
> ```
> $ sudo ln -s /usr/local/bin/node /usr/bin/node
> $ sudo ln -s /usr/local/bin/npm /usr/bin/npm
> ```

- 确保在安装开始前，以下端口处于未被使用状态。

端口号	说明
7001	Studio提供web服务使用。
8080	Nebula HTTP Gateway Client进行HTTP通信使用。
5699	Nebula Importer导入数据导入使用。

安装

1. 根据需要选择并下载RPM包，建议选择最新版本。常用下载链接如下：

安装包	检验和	Nebula版本
nebula-graph-studio-2.2.1-4.x86_64.rpm	nebula-graph-studio-2.2.1-4.x86_64.rpm.sha256	2.0.1
nebula-graph-studio-2.2.0-1.x86_64.rpm	nebula-graph-studio-2.2.0-1.x86_64.rpm.sha256	2.0.1
nebula-graph-studio-2.1.9-1.x86_64.rpm	-	2.0 GA
nebula-graph-studio-1.2.7-2.x86_64.rpm	nebula-graph-studio-1.2.7-2.x86_64.rpm.sha256	1.x

2. 使用 `sudo rpm -i <rpm>` 命令安装RPM包。

   例如，安装Studio book 版本需要运行以下命令：

```
$ sudo rpm -i nebula-graph-studio-book-1.x86_64.rpm
```

卸载

用户可以使用以下的命令卸载 Studio。

```
$ sudo rpm -e nebula-graph-studio-book-1.x86_64.rpm
```


2021 Vesoft Inc.

异常处理

如果在安装过程中自动启动失败或是需要手动启动或停止服务，请使用以下命令

- 手动启动服务

```
$ bash /usr/local/nebula-graph-studio/scripts/start.sh
```

- 手动停止服务

```
$ bash /usr/local/nebula-graph-studio/scripts/stop.sh
```

如果启动服务时遇到报错报错 ERROR: bind EADDRINUSE 0.0.0.0:7001，用户可以通过以下命令查看端口7001是否被占用。

```
$ losf -i:7001
```

如果端口被占用，且无法结束该端口上进程，用户可以通过以下命令修改Studio服务启动端口，并重新启动服务。

```
//修改studio服务启动端口
$ vi config/config.default.js

//修改
...
    config.cluster = {
        listen: {
            port: 7001, // 修改这个端口号，改成任意一个当前可用的即可
            hostname: '0.0.0.0',
        },
    };
...

//重新启动npm
$ npm run start
```

后续操作

进入 Studio 登录界面后，用户需要连接 Nebula Graph。详细信息，参考连接数据库。

最后更新: 2021年5月27日

2021 Vesoft Inc.

11.2.2 连接数据库

在 Nebula Graph Cloud Service 上，创建 Nebula Graph 实例后，用户可以一键直连云服务版 Studio。详细信息参考 Nebula Graph Cloud Service 用户手册。但是，对于 Docker 版和 RPM 版 Studio，在成功启动 Studio 后，用户需要配置连接 Nebula Graph。本文主要描述 Docker 版和 RPM 版 Studio 如何连接 Nebula Graph 数据库。

前提条件

在连接 Nebula Graph 数据库前，用户需要确认以下信息：

- Docker 版或 RPM 版 Studio 已经启动。详细信息参考 部署 Studio。
- Nebula Graph 的 Graph 服务本机 IP 地址以及服务所用端口。默认端口为 `9669`。
- Nebula Graph 数据库登录账号信息，包括用户名和密码。

> **Note**
>
> 如果 Nebula Graph 已经启用了身份验证，并且已经创建了不同角色的用户，用户只能使用被分配到的账号和密码登录数据库。如果未启用身份验证，用户可以使用默认用户名（`user`）和默认密码（`password`）登录数据库。

2021 Vesoft Inc.

操作步骤

按以下步骤连接 Nebula Graph 数据库：

2021 Vesoft Inc.

1. 在 Studio 的 **配置数据库** 页面上，输入以下信息：
   - **Host**：填写 Nebula Graph 的 Graph 服务本机 IP 地址及端口。格式为 `ip:port`。如果端口未修改，则使用默认端口 `9669`。

     > **Note**
     >
     > 即使 Nebula Graph 数据库与 Studio 部署在同一台机器上，用户也必须在 **Host** 字段填写这台机器的本机 IP 地址，而不是 `127.0.0.1` 或者 `localhost`。

   - **用户名** 和 **密码**：根据 Nebula Graph 的身份验证设置填写登录账号和密码。
     - 如果未启用身份验证，可以填写默认用户名 `user` 和默认密码 `password`。
     - 如果已启用身份验证，但是未创建账号信息，用户只能以 GOD 角色登录，必须填写 `root` 及对应的密码 `nebula`。
     - 如果已启用身份验证，同时又创建了不同的用户并分配了角色，不同角色的用户使用自己的账号和密码登录。

2. 完成设置后，点击 **连接** 按钮。
   如果能看到如下图所示的界面，表示已经成功连接到 Nebula Graph 数据库。

一次连接会话持续 30 分钟。如果超过 30 分钟没有操作，会话即断开，用户需要重新登录数据库。

2021 Vesoft Inc.

后续操作

成功连接 Nebula Graph 数据库后，根据账号的权限，用户可以选择执行以下操作：

- 如果已拥有 GOD 或者 ADMIN 权限的账号登录，可以使用 **控制台** 或者 **Schema** 页面管理 Schema。
- 如果已拥有 GOD、ADMIN、DBA 或者 USER 权限的账号登录，可以 批量导入数据 或者在 **控制台** 页面上运行 nGQL 语句插入数据。
- 如果已拥有 GOD、ADMIN、DBA、USER 或者 GUEST 权限的账号登录，可以在 **控制台** 页面上运行 nGQL 语句读取数据或者在 **图探索** 页面上进行图探索或数据分析。

最后更新: 2021年5月19日

2021 Vesoft Inc.

11.2.3 清除连接

使用云服务版 Studio 时，用户不能清除连接。

使用 Docker 版 Studio 时，如果需要重新连接 Nebula Graph 数据库，可以清除当前连接后再重新配置数据库。

当 Docker 版 Studio 还连接在某个 Nebula Graph 数据库时，在工具栏中，选择 **设置 > 清除连接**。之后，如果浏览器上显示 **配置数据库** 页面，表示 Studio 已经成功断开了与 Nebula Graph 数据库的连接。

最后更新: 2021年5月14日

2021 Vesoft Inc.

11.3 快速开始

11.3.1 规划 Schema

在使用 Studio 之前，用户需要先根据 Nebula Graph 数据库的要求规划 Schema（模式）。

Schema 至少要包含以下要素：

- 标签（Tag，即点类型），以及每种标签的属性。
- 边类型（Edge Type），以及每种边类型的属性。

用户可以下载Nebula Graph示例数据集basketballplayer，本文将通过该实例说明如何规划 Schema。

下表列出了 Schema 要素。

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

下图说明示例中 **player** 类点与 **team** 类点之间如何发生关系（**serve/follow**）。

最后更新: 2021年5月17日

2021 Vesoft Inc.

11.3.2 创建 Schema

在 Nebula Graph 中，用户必须先有 Schema，再向其中写入点数据和边数据。本文描述如何使用 Nebula Graph 的 **控制台** 或 **Schema** 功能创建 Schema。

> **Note**
>
> 用户也可以使用 nebula-console 创建 Schema。详细信息，参考 [使用 Docker Compose 部署 Nebula Graph] 和 Nebula Graph 快速开始。

前提条件

在 Studio 上创建 Schema 之前，用户需要确认以下信息：

- Studio 已经连接到 Nebula Graph 数据库。
- 账号拥有 GOD、ADMIN 或 DBA 权限。详细信息，参考 Nebula Graph 内置角色。
- 已经规划好了 Schema 的要素。
- 已经创建了图空间。

> **Note**
>
> 本示例假设已经创建了图空间。如果账号拥有 GOD 权限，也可以在 **控制台** 或 **Schema** 上创建一个图空间。

使用 Schema 管理功能创建 Schema

按以下步骤使用 **Schema** 创建 Schema：

1. 创建标签。详细信息，参考 操作标签。
2. 创建边类型。详细信息，参考 操作边类型。

2021 Vesoft Inc.

使用控制台创建 Schema

按以下步骤使用 **控制台** 创建 Schema：

1. 在工具栏里，点击 **控制台** 页签。
2. 在 **当前Space** 中选择一个图空间。在本示例中，选择 **basketballplayer**。

3. 在命令行中，依次输入以下语句，并点击 图标。

```
-- 创建标签 player，带有 2 个属性
CREATE TAG player(name string, age int);

-- 创建标签 team，带有 1 个属性
CREATE TAG team(name string);

-- 创建边类型 follow，带有 1 个属性
CREATE EDGE follow(degree int);

-- 创建边类型 serve，带有 2 个属性
CREATE EDGE serve(start_year int, end_year int);
```

至此，用户已经完成了 Schema 创建。用户可以运行以下语句查看标签与边类型的定义是否正确、完整。

```
-- 列出当前图空间中所有标签
SHOW TAGS;

-- 列出当前图空间中所有边类型
SHOW EDGES;

-- 查看每种标签和边类型的结构是否正确
DESCRIBE TAG player;
DESCRIBE TAG team;
DESCRIBE EDGE follow;
DESCRIBE EDGE serve;
```

后续操作

创建 Schema 后，用户可以开始 导入数据。

最后更新: 2021年5月19日

2021 Vesoft Inc.

11.3.3 导入数据

准备好 CSV 文件，创建了 Schema 后，用户可以使用 **导入** 功能将所有点和边数据上传到 Studio，用于数据查询、图探索和数据分析。

前提条件

导入数据之前，需要确认以下信息：

- Studio 已经连接到 Nebula Graph 数据库。
- Nebula Graph 数据库里已经创建了 Schema。
- CSV 文件符合 Schema 要求。
- 账号拥有 GOD、ADMIN、DBA 或者 USER 的权限，能往图空间中写入数据。

2021 Vesoft Inc.

操作步骤

按以下步骤导入数据：

2021 Vesoft Inc.

1. 在工具栏里，点击 **导入** 页签。
2. 在 **选择Space** 页面，选择一个图空间，再点击 **下一步** 按钮。
3. 在 **上传文件** 页面，点击 **上传文件** 按钮，并选择需要的 CSV 文件。本示例中，选择 `edge_serve.csv`、`edge_follow.csv`、`vertex_player.csv` 和 `vertex_team.csv` 文件。

> **Note**
>
> 一次可以选择多个 CSV 文件，本文使用的 CSV 文件可以在规划 Schema 中下载。

4. 在文件列表的 **操作** 列，点击 **预览** 或 **删除**，保证文件信息正确，之后，再点击 **下一步** 按钮。
5. 在 **关联点** 页面，点击 **+ 绑定数据源** 按钮，在对话框中选择点数据文件，并点击 **确认** 按钮。如本示例中的 `vertex_player.csv` 或 `vertex_team.csv` 文件。
6. 在 **数据源 X** 页签下，点击 **+ Tag** 按钮。
7. 在 **vertexId** 部分，完成以下操作：
   a. 在 **对应列标** 列，点击 **选择**。

   b. 在弹出对话框中，选择数据列。在本示例中，`user.csv` 中仅有一列数据用于生成代表用户的 VID，`course.csv` 中选择表示 `courseName` 信息的 **Column 1** 用于生成代表课程的 VID。

> **Note**
>
> 在同一个图空间中，VID 始终唯一，不可重复。关于 VID 的信息，参考 Nebula Graph 的点ID "点击进入 Nebula Graph 用户手册")。

8. 在 **TAG 1** 部分，完成以下操作：
   a. 在 **TAG** 下拉列表中，选择数据源对应的标签名称。在本示例中，`vertex_player.csv` 文件对应选择 **player**；`vertex_team.csv` 文件对应选择 **team**。
   b. 在显示的属性列表中，点击 **选择**，为标签属性绑定源数据。在本示例中，`player` 标签的 `name` 属性对应 `vertex_player.csv` 文件中的 **Column 2** 列，**类型** 为 **string**，`age` 属性对应文件中的 **Column 1** 列，**类型** 为 **int**；`team` 标签的 `name` 属性对应 `vertex_team.csv` 文件中的 **Column 1** 列，**类型** 为 **string**。

2021 Vesoft Inc.

9. （可选）如果有多个标签数据文件，重复步骤 5 到步骤 8。
10. 完成配置后，点击 **下一步**。
    界面提示 **配置验证成功**，表示标签数据源绑定成功。
11. 在 **关联边** 页面，点击 **+ 绑定数据源** 按钮，在对话框中选择边数据文件，并点击 **确认** 按钮。如本示例中的 `edge_follow.csv` 文件。
12. 在 **Edge X** 页签的 **类型** 下拉列表中，选择边类型名称。本示例中，选择 **follow**。
13. 根据边类型的属性，从 `edge_follow.csv` 文件中选择相应的数据列。其中，**srcId** 和 **dstId** 分别表示边的起点与终点，所选择的数据及处理方式必须与相应的 VID 保持一致。本示例中，**srcId** 对应的是表示起点球员的 VID，**dstId** 对应的是表示终点球员的 VID。**rank** 为选填项，可以忽略。

14. 完成设置后，点击 **下一步** 按钮。
15. 在 **导入** 页面，点击 **导入** 按钮开始导入数据。 在 **log** 页面上可以看到数据导入进度。导入所需时间因数据量而异。导入过程中可以点击 **终止导入** 停止数据导入。当 **log** 页面显示如图所示信息时，表示数据导入完成。

```
2021/05/06 03:05:13 [INFO] reader.go:180: Total lines of file(/usr/local/nebula-graph-studio/tmp/upload/vertex_team.csv) is: 30, error lines: 0
2021/05/06 03:05:13 [INFO] reader.go:64: Start to read file(2): /usr/local/nebula-graph-studio/tmp/upload/edge_serve.csv, schema: <
:SRC_VID(string),:DST_VID(string),serve.start_year:int,serve.end_year:int >
2021/05/06 03:05:13 [INFO] reader.go:64: Start to read file(3): /usr/local/nebula-graph-studio/tmp/upload/edge_follow.csv, schema: <
:SRC_VID(string),:DST_VID(string),follow.degree:int >
2021/05/06 03:05:13 [INFO] reader.go:180: Total lines of file(/usr/local/nebula-graph-studio/tmp/upload/edge_follow.csv) is: 81, error lines: 0
2021/05/06 03:05:13 [INFO] reader.go:180: Total lines of file(/usr/local/nebula-graph-studio/tmp/upload/edge_serve.csv) is: 152, error lines: 0
2021/05/06 03:05:13 [INFO] statsmgr.go:61: Done(/usr/local/nebula-graph-studio/tmp/upload/vertex_player.csv): Time(0.03s), Finished(51), Failed(0), Latency
AVG(6857us), Batches Req AVG(8667us), Rows AVG(1767.23/s)
2021/05/06 03:05:13 [INFO] statsmgr.go:61: Done(/usr/local/nebula-graph-studio/tmp/upload/vertex_team.csv): Time(0.04s), Finished(81), Failed(0), Latency
AVG(5696us), Batches Req AVG(7650us), Rows AVG(2274.78/s)
```

后续操作

完成数据导入后，用户可以开始 图探索。

最后更新: 2021年5月26日

2021 Vesoft Inc.

11.3.4 查询图数据

导入数据后，用户可以开始使用 **控制台** 或者 **图探索** 查询图数据。

以查询代表“History of Chinese Women Through Time”课程的点的属性为例：

- 在 **控制台** 页面：运行 `FETCH PROP ON serve "player100" -> "team204";`，数据库会返回 rank 为0的边。返回结果后，点击 **查看子图** 按钮，将点数据查询结果导入 **图探索** 进行可视化显示。

- 在 **图探索** 页面：点击 **开始探索** 按钮，在 **指定VID** 对话框中，输入 **"player101"**，**图探索** 画板里会显示这个点，将鼠标移到点上，用户能看到这个点所有属性信息，如下图所示。

最后更新: 2021年5月14日

2021 Vesoft Inc.

11.4 操作指南

11.4.1 管理 Schema

操作图空间

Studio 连接到 Nebula Graph 数据库后，用户可以创建或删除图空间。用户可以使用 **控制台** 或者 **Schema** 操作图空间。本文仅说明如何使用 **Schema** 操作图空间。

支持版本

Studio vbook 及以后版本。请更新版本，详细操作参考 版本更新。

前提条件

操作图空间之前，用户需要确保以下信息：

- Studio 已经连接到 Nebula Graph 数据库。
- 当前登录的账号拥有创建或删除图空间的权限，即：
  - 如果 Nebula Graph 未开启身份验证，用户以默认用户名 `user` 账号和默认密码 `password` 登录。
  - 如果 Nebula Graph 已开启身份验证，用户以 `root` 账号及其密码登录。

创建图空间

按以下步骤使用 **Schema** 创建图空间：

1. 在工具栏里，点击 **Schema** 页签。
2. 在图空间列表上方，点击 **+ 创建** 按钮。
3. 在 **创建** 页面，完成以下配置：
   a. **名称**：指定图空间名称，本示例中设置为 `basketballplayer`。不可与已有的图空间名称重复。不可使用关键字或保留关键字做标识符，参考关键字。
   b. **选填参数**：分别设置 `partition_num`、`replica_factor`、`charset`、`collate` 和 `vid type` 的值。在本示例中，四个参数分别设置为 `10`、`1`、`utf8`、`utf8_bin` 和 `FIXED_STRING(32)`。详细信息，参考 `CREATE SPACE` 语法。

   在 **对应的nGQL语句** 面板上，用户能看到上述设置对应的 nGQL 语句。如下所示：

   ```
   CREATE SPACE basketballplayer (partition_num = 10, replica_factor = 1, charset = utf8, collate = utf8_bin, vid_type = FIXED_STRING(32))
   ```

4. 配置确认无误后，点击 **创建** 按钮。 如果页面回到 **图空间列表**，而且列表中显示刚创建的图空间信息，表示图空间创建成功。

2021 Vesoft Inc.

删除图空间

> **Danger**
>
> 删除图空间会删除其中的所有数据，已删除的数据如未备份无法恢复。

按以下步骤使用 **Schema** 删除图空间：

1. 在工具栏里，点击 **Schema** 页签。
2. 在图空间列表里，找到需要删除的图空间，并在 **操作** 列中，点击 图标。

3. 在弹出的对话框中点击 **确认**。 删除成功后，页面回到 **图空间列表**。

2021 Vesoft Inc.

后续操作

图空间创建成功后，用户可以开始创建或修改 Schema，包括：

- 操作标签
- 操作边类型
- 操作索引

最后更新: 2021年5月18日

2021 Vesoft Inc.

操作标签（点类型）

在 Nebula Graph 数据库中创建图空间后，用户需要创建标签（点类型）。用户可以选择使用 **控制台** 或者 **Schema** 操作标签。本文仅说明如何使用 **Schema** 操作标签。

支持版本

Studio vbook 及以后版本。请更新版本，详细操作参考 版本更新。

前提条件

在 Studio 上操作标签之前，用户必须确认以下信息：

- Studio 已经连接到 Nebula Graph 数据库。
- 图空间已经创建。
- 当前登录的账号拥有 GOD、ADMIN 或者 DBA 的权限。

创建标签

按以下步骤使用 **Schema** 创建标签：

1. 在工具栏中，点击 **Schema** 页签。
2. 在 **图空间列表** 中，找到图空间，点击图空间名称或者在 **操作** 列中点击 图标。
3. 在 **当前图空间** 里确认图空间名称。用户也可以通过选择图空间名称切换图空间。
4. 点击 **标签** 页签，并点击 **创建** 按钮。
5. 在 **创建** 页面上，完成以下设置：

   a. **名称**：按提示信息输入合规的标签名称。本示例中，输入 `player` 和 `team`。

   b. （可选）如果标签需要属性，在 **定义属性** 模块左上角，点击勾选框，并在展开的列表中，完成以下操作：
   - 输入属性名称、数据类型和默认值。 - 如果一个标签有多个属性，可以点击 **添加属性** 按钮，并定义属性。 - 如果要删除某个属性，在该属性所在行，点击 图标。

   c. （可选）标签未设置索引时，用户可以设置 TTL：在 **设置TTL** 模块左上角，点击勾选框，并在展开的列表中设置 `TTL_COL` 和 `TTL_DURATION` 参数信息。关于这两个参数的详细信息，参考 TTL 配置。
6. 完成设置后，在 **对应的 nGQL**面板，用户能看到与上述配置等价的 nGQL 语句。
7. 确认无误后，点击 **+ 创建** 按钮。 如果标签创建成功，**定义属性**面板会显示这个标签的属性列表。

2021 Vesoft Inc.

修改标签

按以下步骤使用 **Schema** 修改标签：

1. 在工具栏中，点击 **Schema** 页签。
2. 在 **图空间列表** 中，找到图空间，点击图空间名称或者在 **操作** 列中点击 图标。
3. 在 **当前图空间** 里确认图空间名称。用户也可以通过选择图空间名称切换图空间。
4. 点击 **标签** 页签，找到需要修改的标签，并在 **操作** 列中，点击 图标。
5. 在 **编辑** 页面，用户可以选择以下操作：
   - 如果要修改属性：在 **定义属性** 面板上，找到需要修改的属性，在右侧点击 **编辑**，再修改属性的数据类型和默认值。之后，点击 **确认** 或者 **取消** 完成修改。
   - 如果要删除属性：在 **定义属性** 面板上，找到需要删除的属性，在右侧点击 **删除**，经确认后，删除属性。
   - 如果要添加属性：在 **定义属性**面板上，点击 **添加属性** 按钮，添加属性信息。详细操作，参考 创建标签面板。
   - 如果配置了 TTL，要修改 TTL 信息：在 **设置TTL** 面板上，修改 `TTL_COL` 和 `TTL_DURATION` 配置。
   - 如果要删除已经配置的 TTL 信息：在 **设置TTL** 面板的左上角，点击勾选框，取消选择。
   - 如果要配置 TTL 信息：在 **使用TTL** 面板的右上角，点击勾选框，开始设置 TTL 信息。
6. 完成设置后，在 **对应的 nGQL** 面板上，用户能看到修改后的 nGQL 语句。

删除标签

> **Danger**
>
> 删除标签前先确认影响，已删除的数据如未备份无法恢复。

按以下步骤使用 **Schema** 删除标签：

1. 在工具栏中，点击 **Schema** 页签。
2. 在 **图空间列表** 中，找到图空间，点击图空间名称或者在 **操作** 列中点击 图标。
3. 在 **当前图空间** 里确认图空间名称。用户也可以通过选择图空间名称切换图空间。
4. 点击 **标签** 页签，找到需要修改的标签，并在 **操作** 列中，点击 图标。
5. 在弹出的对话框中点击 **确认**。

后续操作

标签创建成功后，用户可以在 **控制台** 上逐条插入点数据，或者使用 **导入** 功能批量插入点数据。

最后更新: 2021年5月17日

2021 Vesoft Inc.

操作边类型

在 Nebula Graph 数据库中创建图空间后，用户可能需要创建边类型。用户可以选择使用 **控制台** 或者 **Schema** 操作边类型。本文仅说明如何使用 **Schema** 操作边类型。

支持版本

Studio vbook 及以后版本。请更新版本，详细操作参考 版本更新。

前提条件

在 Studio 上操作边类型之前，用户必须确认以下信息：

- Studio 已经连接到 Nebula Graph 数据库。
- 图空间已经创建。
- 当前登录的账号拥有 GOD、ADMIN 或者 DBA 的权限。

2021 Vesoft Inc.

创建边类型

按以下步骤使用 **Schema** 创建边类型：

1. 在工具栏中，点击 **Schema** 页签。
2. 在 **图空间列表** 中，找到图空间，点击图空间名称或者在 **操作** 列中点击 图标。
3. 在 **当前图空间** 里确认图空间名称。用户也可以通过选择图空间名称切换图空间。
4. 点击 **边类型** 页签，并点击 **创建** 按钮。
5. 在 **创建** 页面上，完成以下设置：
   a. **名称**：按提示信息输入合规的边类型名称。本示例中，输入 `serve`。
   b. （可选）如果边类型需要属性，在 **定义属性**面板的左上角，点击勾选框，并在展开的列表中，完成以下操作：
      - 输入属性名称、数据类型和默认值。
      - 如果一个边类型有多个属性，可以点击 **添加属性** 按钮，并定义属性。
      - 如果要删除某个属性，在该属性所在行，点击 图标。
   c. （可选）边类型未设置索引时，用户可以设置 TTL：在 **设置TTL**面板的左上角，点击勾选框，并在展开的列表中设置 `TTL_COL` 和 `TTL_DURATION` 参数信息。关于这两个参数的详细信息，参考 TTL 配置。
6. 完成设置后，在 **对应的nGQL语句** 面板上，用户能看到与上述配置等价的 nGQL 语句。

7. 确认无误后，点击 **+ 创建** 按钮。 如果边类型创建成功，**定义属性**面板会显示这个边类型的属性列表。

2021 Vesoft Inc.

修改边类型

按以下步骤使用 **Schema** 修改边类型：

1. 在工具栏中，点击 **Schema** 页签。
2. 在 **图空间列表** 中，找到图空间，点击图空间名称或者在 **操作** 列中点击 图标。
3. 在 **当前图空间** 里确认图空间名称。用户也可以通过选择图空间名称完成图空间切换。
4. 点击 **边类型** 页签，找到需要修改的边类型，并在 **操作** 列中，点击 图标。
5. 在 **编辑** 页面，用户可以选择以下操作：
   - 如果要修改属性：在 **定义属性** 面板上，找到需要修改的属性，在右侧点击 **编辑**，再修改属性的数据类型或者默认值。修改完成后，点击 **确认** 或 **取消**。
   - 如果要删除属性：在 **定义属性** 面板上，找到需要删除的属性，在右侧点击 **删除**，经确认后，删除属性。
   - 如果要添加属性：在 **定义属性** 面板上，点击 **添加属性** 按钮，添加属性信息。
   - 如果要修改 TTL：在 **设置TTL** 面板上，修改或 `TTL_COL` 和 `TTL_DURATION` 设置。
   - 如果要删除所有已经配置的 TTL：在 **设置TTL** 面板的左上角，点击勾选框，取消选择。
   - 如果要设置 TTL：在 **设置TTL** 面板的左上角，点击勾选框，开始设置 TTL。
6. 完成设置后，在 **对应的nGQL语句** 面板上，用户能看到修改后的 nGQL 语句。

删除边类型

> **Danger**
>
> 删除标签前先确认影响，已删除的数据如未备份无法恢复。

按以下步骤使用 **Schema** 删除边类型：

1. 在工具栏中，点击 **Schema** 页签。
2. 在 **图空间列表** 中，找到图空间，点击图空间名称或者在 **操作** 列中点击 图标。
3. 在 **当前图空间** 里确认图空间名称。用户也可以通过选择图空间名称切换图空间。
4. 点击 **边类型** 页签，找到需要修改的边类型，并在 **操作** 列中，点击 图标。
5. 在弹出的对话框中点击 **确认**。

后续操作

边类型创建成功后，用户可以在 **控制台** 上逐条插入边数据，或者使用 **导入** 功能批量插入边数据。

最后更新: 2021年5月17日

2021 Vesoft Inc.

操作索引

用户可以为标签和边类型创建索引，使得图查询时可以从拥有共同属性的同一类型的点或边开始遍历，使大型图的查询更为高效。Nebula Graph 支持两种类型的索引：标签索引和边类型索引。用户可以选择使用 **控制台** 或者 **Schema** 操作索引。本文仅说明如何使用 **Schema** 操作索引。

> **Note**
>
> 一般在创建了标签或者边类型之后即可创建索引，但是，索引会影响写性能，所以，建议先导入数据，再批量重建索引。关于索引的详细信息，参考《nGQL 用户手册》。

支持版本

Studio vbook 及以后版本。请更新版本，详细操作参考 版本更新。

前提条件

在 Studio 上操作索引之前，用户必须确认以下信息：

- Studio 已经连接到 Nebula Graph 数据库。
- 图空间、标签和边类型已经创建。
- 当前登录的账号拥有 GOD、ADMIN 或者 DBA 的权限。

2021 Vesoft Inc.

创建索引

按以下步骤使用 **Schema** 创建索引：

1. 在工具栏中，点击 **Schema** 页签。
2. 在 **图空间列表** 中，找到图空间，点击图空间名称或者在 **操作** 列中点击 图标。
3. 在 **当前图空间** 里确认图空间名称。用户也可以通过选择图空间名称切换图空间。
4. 点击 **索引** 页签，再点击 **创建** 按钮。
5. 在 **创建** 页面上，完成以下设置：
   a. **索引类型**：确认或修改索引类型，即 **标签** 或者 **边类型**。本示例中选择 **边类型**。
   b. **名称**：选择要创建索引的标签或边类型名称。本示例中选择 **follow**。
   c. **索引名称**：按规定指定索引名称。本示例中输入 **follow_index**。
   d. **索引属性**：点击 **添加**，在 **选择关联的属性** 列表里选择需要索引的属性，并点击 **确定** 按钮。如果需要关联多个属性，重复这一步操作。用户可以按界面提示重排索引属性的顺序。本示例中选择 `degree`。

> **Note**
>
> 索引属性的顺序会影响 `LOOKUP` 语句的查询结果。详细信息，参考 《nGQL 用户手册》。

6. 完成设置后，在 **对应的 nGQL**面板，用户能看到与上述配置等价的 nGQL 语句。

7. 确认无误后，点击 **+ 创建** 按钮。 如果索引创建成功，**定义属性**面板会显示这个索引的属性列表。

2021 Vesoft Inc.

查看索引

按以下步骤使用 **Schema** 查看索引：

1. 在工具栏中，点击 **Schema** 页签。
2. 在 **图空间列表** 中，找到图空间，点击图空间名称或者在 **操作** 列中点击 图标。
3. 在 **当前图空间** 里确认图空间名称。用户也可以通过选择图空间名称切换图空间。
4. 点击 **索引** 页签，在列表左上方，选择需要查看的索引类型。
5. 在列表中，找到需要查看的索引，点击索引所在行。界面上即列出索引相关的所有属性。

删除索引

按以下步骤使用 **Schema** 删除索引：

1. 在工具栏中，点击 **Schema** 页签。
2. 在 **图空间列表** 中，找到图空间，点击图空间名称或者在 **操作** 列中点击 图标。
3. 在 **当前图空间** 里确认图空间名称。用户也可以通过选择图空间名称切换图空间。
4. 点击 **索引** 页签，找到需要修改的索引，并在 **操作** 列中，点击 图标。
5. 在弹出的对话框中点击 **确认**。

最后更新: 2021年5月17日

2021 Vesoft Inc.

11.4.2 使用控制台

控制台界面

Studio 的控制台界面如下图所示。

下表列出了控制台界面上的各种功能。

编号	功能	说明
1	工具栏	点击 **控制台** 页签进入控制台页面。
2	选择图空间	在 **当前图空间** 列表中选择一个图空间。 **说明**：Studio 不支持直接在输入框中运行 `USE <space_name>` 语句。
3	输入框	在输入框中输入 nGQL 语句后，点击 按钮运行语句。用户可以同时输入多个语句同时运行，语句之间以 `;` 分隔。
4	清空输入框	点击 按钮，清空输入框中已经输入的内容。
5	重复语句输入	点击 按钮，在语句运行记录列表里，点击其中一个语句，输入框中即自动输入该语句。列表里提供最近 15 次语句运行记录。
6	运行	在输入框中输入 nGQL 语句后，点击 按钮即开始运行语句。
7	语句运行状态	运行 nGQL 语句后，这里显示语句运行状态。如果语句运行成功，语句以绿色显示。如果语句运行失败，语句以红色显示。
8	结果窗口	显示语句运行结果。如果语句会返回结果，结果窗口会以表格形式呈现返回的结果。
9	导出CSV文件	运行 nGQL 语句返回结果后，点击 **导出CSV文件** 按钮即能将结果以 CSV 文件的形式导出。
10	图探索功能键	根据运行的 nGQL 语句，用户可以点击图探索功能键将返回的结果导入 **图探索** 进行可视化展现，例如 导入图探索 和 查看子图。

最后更新: 2021年5月14日

2021 Vesoft Inc.

导入图探索

用户可以在 **控制台** 上使用 nGQL 语句查询得到点或边的信息，再借助 **导入图探索** 功能实现查询结果的可视化。

支持版本

Studio vbook 及以后版本。请更新版本，详细操作参考 版本更新。

前提条件

使用导入图探索前，用户需要确认以下信息：

- Studio 已经连接到 Nebula Graph 数据库。详细信息参考 连接数据库。
- 已经导入数据集。详细操作参考 导入数据。

2021 Vesoft Inc.

导入边数据

按以下步骤将 **控制台** 查询得到的边数据结果导入 **图探索**：

2021 Vesoft Inc.

1. 在工具栏里，点击 **控制台** 页签。
2. 在 **当前Space** 中选择一个图空间。在本示例中，选择 **basketballplayer**。
3. 在命令行中，输入查询语句，并点击 图标。

> **Note**
>
> 查询结果中必须包括边起点和终点 VID 信息。

查询语句示例如下：

```
nebula> GO FROM "player102" OVER serve YIELD serve._src,serve._dst;
```

查询结果可以看到 `playerId` 为 `palyer102` 的球员服务球队的起始年份及终止年份。如下图所示。

4. 点击 **导入图探索** 按钮。
5. 在弹出对话框中，完成如下配置：
   a. 点击 **边类型**。
   b. 在 **Edge Type** 字段，填写边类型名称。在本示例中，填写 `serve`。
   c. 在 **Src ID** 字段，选择查询结果中代表边起点 VID 的列名。在本示例中，选择 `serve._src`。
   d. 在 **Dst ID** 字段，选择查询结果中代表边终点 VID 的列名。在本示例中，选择 `serve._dst`。
   e. （可选）如果返回的边数据中有边权重（`rank`）信息，则在 **Rank** 字段，选择代表边权重的列名。如果 **Rank** 字段未设置，默认为 0。
   f. 完成配置后，点击 **导入** 按钮。

2021 Vesoft Inc.

6. 如果 **图探索** 页面此前已有数据，在弹出的窗口中选择数据插入方式：
   - **增量插入**：在画图板原来的数据基础上插入新的数据。
   - **清除插入**：清除画图板上原来的数据后，再插入新的数据。

数据插入成功后可以看到其可视化表现。

2021 Vesoft Inc.

导入点数据结果

按以下步骤将 **控制台** 查询得到的点数据结果导入 **图探索**：

1. 在工具栏里，点击 **控制台** 页签。
2. 在 **当前Space** 中选择一个图空间。在本示例中，选择 **basketballplayer**。
3. 在命令行中，输入查询语句，并点击 图标。

> **Note**
>
> 查询结果中必须包括点的 VID 信息。

查询语句示例如下：

```
nebula> FETCH PROP ON player "player100" YIELD player.name;
```

查询得到 `playerId` 为 `player100` 的球员信息。如下图所示。

4. 点击 **导入图探索** 按钮。
5. 在弹出对话框中，配置如下：
   a. 点击 **点**。
   b. 在 **Vertex ID** 字段，选择查询结果中代表点 VID 的列名。在本示例中，选择 `VertexID`。
   c. 完成配置后，点击 **导入** 按钮。

6. 如果 **图探索** 页面此前已有数据，在弹出的窗口中选择数据插入方式：
   - **增量插入**：在画图板原来的数据基础上插入新的数据。
   - **清除插入**：清除画图板上原来的数据后，再插入新的数据。

数据插入成功后，用户可以看到查询得到的点数据的可视化表现。

2021 Vesoft Inc.

后续操作

数据导入图探索后，用户可以对数据进行拓展分析。

最后更新: 2021年5月17日

2021 Vesoft Inc.

查看子图

在 Studio 里，用户可以在 **控制台** 上运行 `FIND SHORTEST | ALL PATH` 语句查询得到指定点之间的所有路径或最短路径，然后再通过 **查看子图** 功能将查询得到的路径导入 **图探索** 进行可视化展示。

关于 `FIND SHORTEST | ALL PATH` 语句的详细信息，参考 nGQL 用户手册。

支持版本

Studio vbook 及以后版本。请更新版本，详细操作参考 版本更新。

前提条件

在 **控制台** 上运行 `FIND PATH` 语句并查看子图之前，用户需要确认以下信息：

- Studio 版本为 vbook 及以后版本。
- Studio 已经连接到 Nebula Graph 数据库。详细信息参考 连接数据库。
- 已经导入数据集。详细操作参考 导入数据。

操作步骤

按以下步骤在 **控制台** 运行 `FIND PATH` 语句并将结果导入 **图探索**：

1. 在工具栏里，点击 **控制台** 页签。
2. 在 **当前Space** 中选择一个图空间。在本示例中，选择 **mooc_actions**。
3. 在命令行中，输入 `FIND SHORTEST PATH` 或者 `FIND ALL PATH` 语句，并点击 图标。

   查询语句示例如下：

   ```
   nebula> FIND ALL PATH FROM "player114" to "player100" OVER follow;
   ```

   查询得到如下图所示路径信息。

4. 点击 **查看子图** 按钮。
5. 如果 **图探索** 页面此前已有数据，选择一种数据插入方式：
   - **增量插入**：在画图板原来的数据基础上插入新的数据。
   - **清除插入**：清除画图板上原来的数据后，再插入新的数据。

2021 Vesoft Inc.

数据插入成功后，用户可以看到查询结果的可视化表现。

后续操作

数据导入图探索后，用户可以对数据进行拓展分析。

最后更新: 2021年5月25日

2021 Vesoft Inc.

11.5 故障排查

11.5.1 连接数据库错误

问题描述

按 连接 Studio 文档操作，提示 **配置失败**。

可能的原因及解决方法

用户可以按以下步骤排查问题。

第 1 步. 确认 HOST 字段的格式是否正确

必须填写 Nebula Graph 图数据库 Graph 服务的 IP 地址（`graph_server_ip`）和端口。如果未做修改，端口默认为 `9669`。即使 Nebula Graph 与 Studio 都部署在当前机器上，用户也必须使用本机 IP 地址，而不能使用 `127.0.0.1`、`localhost` 或者 `0.0.0.0`。

第 2 步. 确认 用户名 和 密码 是否正确

如果 Nebula Graph 没有开启身份认证，用户可以填写任意字符串登录。

如果已经开启身份认证，用户必须使用分配的账号登录。

第 3 步. 确认 NEBULA GRAPH 服务是否正常

检查 Nebula Graph 服务状态。关于查看服务的操作：

- 如果在 Linux 服务器上通过编译部署的 Nebula Graph，参考 查看 Nebula Graph 服务。
- 如果使用 Docker Compose 部署和 RPM 部署的 Nebula Graph，参考 查看 Nebula Graph 服务状态和端口。

如果 Nebula Graph 服务正常，进入第 4 步继续排查问题。否则，请重启 Nebula Graph 服务。

> **Note**
>
> 如果之前使用 `docker-compose up -d` 启动 Nebula Graph，必须运行 `docker-compose down` 命令停止 Nebula Graph。

第 4 步. 确认 GRAPH 服务的网络连接是否正常

在 Studio 机器上运行命令（例如 `telnet <graph_server_ip> 9669`）确认 Nebula Graph 的 Graph 服务网络连接是否正常。

如果连接失败，则按以下要求检查：

- 如果 Studio 与 Nebula Graph 在同一台机器上，检查端口是否已暴露。
- 如果两者不在同一台机器上，检查 Nebula Graph 服务器的网络配置，例如，防火墙、网关以及端口。

如果按上述步骤排查后仍无法连接 Nebula Graph 服务，请前往 Nebula Graph 官方论坛 咨询。

最后更新: 2021年5月17日

2021 Vesoft Inc.

11.5.2 无法访问 Studio

问题描述

我按照文档描述启动 Studio 后访问 `127.0.0.1:7001` 或者 `0.0.0.0:7001`，但是打不开页面，为什么？

可能的原因及解决方法

用户可以按以下顺序排查问题。

第 1 步. 确认系统架构

需要确认部署 Studio 服务的机器是否为 x86_64 架构。目前 Studio 仅支持 x86_64 系统架构。

第 2 步. 检查 STUDIO 服务是否正常启动

运行 `docker-compose ps` 查看服务是否已经正常启动。

如果服务正常，返回结果如下。其中，`State` 列应全部显示为 `Up`。

```
     Name                        Command               State              Ports
-------------------------------------------------------------------------------------------------
nebula-web-docker_client_1     ./nebula-go-api                  Up      0.0.0.0:32782->8080/tcp
nebula-web-docker_importer_1   nebula-importer --port=569 ...   Up      0.0.0.0:32783->5699/tcp
nebula-web-docker_nginx_1      /docker-entrypoint.sh ngin ...   Up      0.0.0.0:7001->7001/tcp, 80/tcp
nebula-web-docker_web_1        docker-entrypoint.sh npm r ...   Up      0.0.0.0:32784->7001/tcp
```

如果没有返回以上结果，则先停止 Studio 重新启动。详细信息，参考 部署 Studio。

> **Note**
>
> 如果之前使用 `docker-compose up -d` 启动 Studio，必须运行 `docker-compose down` 命令停止 Studio。

第 3 步. 确认访问地址

如果 Studio 与浏览器在同一台机器上，用户可以在浏览器里使用 `localhost:7001`、`127.0.0.1:7001` 或者 `0.0.0.0:7001` 访问 Studio。

如果两者不在同一台机器上，必须在浏览器里输入 `<studio_server_ip>:7001`。其中，`studio_server_ip` 是指部署 Studio 服务的机器的 IP 地址。

第 4 步. 确认网络连通性

运行 `curl <studio_server_ip>:7001 -I` 确认是否正常。如果返回 `HTTP/1.1 200 OK`，表示网络连通正常。

如果连接被拒绝，则按以下要求检查：

- 如果浏览器与 Studio 在同一台机器上，检查端口是否已暴露。
- 如果两者不在同一台机器上，检查 Studio 所在机器的网络配置，例如，防火墙、网关以及端口。

如果按上述步骤排查后仍无法访问 Studio，请前往 Nebula Graph 官方论坛 咨询。

最后更新: 2021年5月14日

2021 Vesoft Inc.

12. Nebula Importer

12.1 Nebula Importer

Nebula Importer（简称Importer）是一款Nebula Graph的CSV文件单机导入工具。Importer可以读取本地的CSV文件，然后导入数据至Nebula Graph图数据库中。

12.1.1 适用场景

Importer适用于将本地CSV文件的内容导入至Nebula Graph中。

12.1.2 优势

- 轻量快捷：不需要复杂环境即可使用，快速导入数据。
- 灵活筛选：通过配置文件可以实现对CSV文件数据的灵活筛选。

12.1.3 前提条件

在使用Nebula Importer之前，请确保：

- 已部署Nebula Graph服务。目前有三种部署方式：
  - Docker Compose部署（快速部署）
  - RPM/DEB包安装
  - 源码编译安装
- Nebula Graph中已创建Schema，包括图空间、标签和边类型，或者通过参数 `clientSettings.postStart.commands` 设置。
- 运行Importer的机器已部署Golang环境。详情请参见Golang 环境搭建。

12.1.4 操作步骤

配置yaml文件并准备好待导入的CSV文件，即可使用本工具向Nebula Graph批量写入数据。

源码编译运行

1. 克隆仓库。

```
$ git clone --branch <branch> https://github.com/vesoft-inc/nebula-importer.git
```

> **Note**
>
> 请使用正确的分支。 Nebula Graph 1.x和2.x的rpc协议不同，因此：
>
> - Nebula Importer v1分支只能连接Nebula Graph 1.x。
> - Nebula Importer master分支和v2分支可以连接Nebula Graph 2.x。

2. 进入目录 `nebula-importer` 。

```
$ cd nebula-importer
```

3. 编译源码。

```
$ make build
```


2021 Vesoft Inc.

4. 启动服务。

```
$ ./nebula-importer --config <yaml_config_file_path>
```

> **Note**
>
> yaml配置文件说明请参见配置文件。

无网络编译方式

如果服务器不能联网，建议在能联网的机器上将源码和各种依赖打包上传到对应的服务器上编译即可，操作步骤如下：

1. 克隆仓库。

```
$ git clone --branch <branch> https://github.com/vesoft-inc/nebula-importer.git
```

2. 使用如下的命令下载并打包依赖的源码。

```
$ cd nebula-importer
$ go mod vendor
$ cd .. && tar -zcvf nebula-importer.tar.gz nebula-importer
```

3. 将压缩包上传到不能联网的服务器上。
4. 解压并编译。

```
$ tar -zxvf nebula-importer.tar.gz
$ cd nebula-importer
$ go build -mod vendor cmd/
```

Docker方式运行

使用Docker可以不必在本地安装Go语言环境，只需要拉取Nebula Importer的镜像，并将本地配置文件和CSV数据文件挂载到容器中。命令如下：

```
$ docker run --rm -ti \
    --network=host \
    -v <config_file>:<config_file> \
    -v <csv_data_dir>:<csv_data_dir> \
    vesoft/nebula-importer:<version>
    --config <config_file>
```

- `<config_file>`：本地yaml配置文件的绝对路径。
- `<csv_data_dir>`：本地CSV数据文件的绝对路径。
- `<version>`：Nebula Graph 2.x请填写`v2`。

> **Note**
>
> 建议使用相对路径。如果使用本地绝对路径，请检查路径映射到Docker中的路径。

12.1.5 配置文件说明

Nebula Importer通过docker-compose.yaml配置文件来描述待导入文件信息、Nebula Graph服务器信息等。用户可以参考示例配置文件：无表头配置/有表头配置。下文将分类介绍配置文件内的字段。

基本配置

示例配置如下：

2021 Vesoft Inc.

```
version: v2
description: example
removeTempFiles: false
```

参数	默认值	是否必须	说明
`version`	v2	是	目标Nebula Graph的版本。
`description`	example	否	配置文件的描述。
`removeTempFiles`	false	否	是否删除临时生成的日志和错误数据文件。

客户端配置

客户端配置存储客户端连接Nebula Graph相关的配置。

示例配置如下：

```
clientSettings:
  retry: 3
  concurrency: 10
  channelBufferSize: 128
  space: test
  connection:
    user: user
    password: password
    address: 192.168.11.13:9669,192.168.11.14:9669
  postStart:
    commands: |
      UPDATE CONFIGS storage:wal_ttl=3600;
      UPDATE CONFIGS storage:rocksdb_column_family_options = { disable_auto_compactions = true };
    afterPeriod: 8s
  preStop:
    commands: |
      UPDATE CONFIGS storage:wal_ttl=86400;
      UPDATE CONFIGS storage:rocksdb_column_family_options = { disable_auto_compactions = false };
```

参数	默认值	是否必须	说明
`clientSettings.retry`	3	否	nGQL语句执行失败的重试次数。
`clientSettings.concurrency`	10	否	Nebula Graph客户端并发数。
`clientSettings.channelBufferSize`	128	否	每个Nebula Graph客户端的缓存队列大小。
`clientSettings.space`	-	是	指定数据要导入的Nebula Graph图空间。不要同时导入多个空间，以免影响性能。
`clientSettings.connection.user`	-	是	Nebula Graph的用户名。
`clientSettings.connection.password`	-	是	Nebula Graph用户名对应的密码。
`clientSettings.connection.address`	-	是	所有Graph服务的地址和端口。
`clientSettings.postStart.commands`	-	否	配置连接Nebula Graph服务器之后，在插入数据之前执行的一些操作。
`clientSettings.postStart.afterPeriod`	-	否	执行上述 `commands` 命令后到执行插入数据命令之间的间隔，例如 `8s` 。
`clientSettings.preStop.commands`	-	否	配置断开Nebula Graph服务器连接之前执行的一些操作。

文件配置

文件配置存储数据文件和日志的相关配置，以及Schema的具体信息。

文件和日志配置

示例配置如下：

```
logPath: ./err/test.log
files:
```


2021 Vesoft Inc.

```
  - path: ./student_without_header.csv
    failDataPath: ./err/studenterr.csv
    batchSize: 128
    limit: 10
    inOrder: false
    type: csv
    csv:
      withHeader: false
      withLabel: false
      delimiter: ","
```

参数	默认值	是否必须	说明
`logPath`	-	否	导入过程中的错误等日志信息输出的文件路径。
`files.path`	-	是	数据文件的存放路径，如果使用相对路径，则会将路径和当前配置文件的目录拼接。
`files.failDataPath`	-	是	插入失败的数据文件存放路径，以便后面补写数据。
`files.batchSize`	128	否	单批次插入数据的语句数量。
`files.limit`	-	否	读取数据的行数限制。
`files.inOrder`	-	否	是否按顺序在文件中插入数据行。如果为 `false`，可以避免数据倾斜导致的导入速率降低。
`files.type`	-	是	文件类型。
`files.csv.withHeader`	`false`	是	是否有表头。详情请参见关于CSV文件表头。
`files.csv.withLabel`	`false`	是	是否有LABEL。详情请参见有表头配置说明。
`files.csv.delimiter`	`","`	是	指定csv文件的分隔符。只支持一个字符的字符串分隔符。

SCHEMA配置

Schema配置描述当前数据文件的Meta信息，Schema的类型分为点和边两类，可以同时配置多个点或边。

- 点配置

示例配置如下：

```
schema:
  type: vertex
  vertex:
    vid:
      type: string
      index: 0
    tags:
      - name: student
        props:
          - name: name
            type: string
            index: 1
          - name: age
            type: int
            index: 2
          - name: gender
```


2021 Vesoft Inc.

```
        type: string
        index: 3
```

参数	默认值	是否必须	说明
`files.schema.type`	-	是	Schema的类型，可选值为 `vertex` 和 `edge` 。
`files.schema.vertex.vid.type`	-	否	点ID的数据类型，可选值为 `int` 和 `string` 。
`files.schema.vertex.vid.index`	-	否	点ID对应CSV文件中列的序号。
`files.schema.vertex.tags.name`	-	是	标签名称。
`files.schema.vertex.tags.props.name`	-	是	标签属性名称，必须和Nebula Graph中的标签属性一致。
`files.schema.vertex.tags.props.type`	-	否	属性数据类型，支持 `bool` 、 `int` 、 `float` 、 `double` 、 `timestamp` 和 `string` 。
`files.schema.vertex.tags.props.index`	-	否	属性对应CSV文件中列的序号。

Note

CSV文件中列的序号从0开始，即第一列的序号为0，第二列的序号为1。

- 边配置

示例配置如下：

```
schema:
  type: edge
  edge:
    name: follow
    withRanking: true
    srcVID:
      type: string
      index: 0
    dstVID:
      type: string
      index: 1
    rank:
      index: 2
    props:
      - name: degree
        type: double
        index: 3
```

参数	默认值	是否必须	说明
`files.schema.type`	-	是	Schema的类型，可选值为 `vertex` 和 `edge` 。
`files.schema.edge.name`	-	是	边类型名称。
`files.schema.edge.srcVID.type`	-	否	边的起始点ID的数据类型。
`files.schema.edge.srcVID.index`	-	否	边的起始点ID对应CSV文件中列的序号。
`files.schema.edge.dstVID.type`	-	否	边的目的点ID的数据类型。
`files.schema.edge.dstVID.index`	-	否	边的目的点ID对应CSV文件中列的序号。
`files.schema.edge.rank.index`	-	否	边的rank值对应CSV文件中列的序号。
`files.schema.edge.props.name`	-	是	边类型属性名称，必须和Nebula Graph中的边类型属性一致。
`files.schema.edge.props.type`	-	否	属性类型，支持 `bool` 、 `int` 、 `float` 、 `double` 、 `timestamp` 和 `string` 。
`files.schema.edge.props.index`	-	否	属性对应CSV文件中列的序号。

2021 Vesoft Inc.

12.1.6 关于CSV文件表头（header）

Importer根据CSV文件有无表头，需要对配置文件进行不同的设置，相关示例和说明请参见：

- 无表头配置说明
- 有表头配置说明

最后更新: 2021年5月26日

2021 Vesoft Inc.

12.2 有表头配置说明

对于有表头（header）的CSV文件，需要在配置文件里设置 `withHeader` 为 `true`，表示CSV文件中第一行为表头，表头内容具有特殊含义。

> **Caution**
>
> 如果CSV文件中含有header，Importer就会按照header来解析每行数据的Schema，并忽略yaml文件中的点或边设置。

12.2.1 示例文件

有表头的CSV文件示例如下：

- 点示例

  `student_with_header.csv` 的示例数据：

  ```
  :VID(string),student.name:string,student.age:int,student.gender:string
  student100,Monica,16,female
  student101,Mike,18,male
  student102,Jane,17,female
  ```

  第一列为点ID，后面三列为属性 `name`、`age` 和 `gender`。

- 边示例

  `follow_with_header.csv` 的示例数据：

  ```
  :SRC_VID(string),:DST_VID(string),:RANK,follow.degree:double
  student100,student101,0,92.5
  student101,student100,1,85.6
  student101,student102,2,93.2
  student100,student102,1,96.2
  ```

  前两列的数据分别为起始点ID和目的点ID，第三列为rank，第四列为属性 `degree`。

12.2.2 表头格式说明

表头通过一些关键词定义起始点、目的点、rank以及一些特殊功能，说明如下：

- `:VID` （必填）：点ID。需要用 `:VID(type)` 形式设置数据类型，例如 `:VID(string)` 或 `:VID(int)`。
- `:SRC_VID` （必填）：边的起始点ID。需要用 `:SRC_VID(type)` 形式设置数据类型。
- `:DST_VID` （必填）：边的目的点ID。需要用 `:DST_VID(type)` 形式设置数据类型。
- `:RANK` （可选）：边的rank值。
- `:IGNORE` （可选）：插入数据时忽略这一列。
- `:LABEL` （可选）：表示对该行进行插入（+）或删除（-）操作。必须为第一列。例如：

  ```
  :LABEL,
  +,
  -,
  ```

> **Note**
>
> 除了 `:LABEL` 列之外的所有列都可以按任何顺序排序，因此针对较大的CSV文件，用户可以灵活地设置header来选择需要的列。

2021 Vesoft Inc.

对于标签或边类型的属性，格式为 `<tag_name/edge_name>.<prop_name>:<prop_type>`，说明如下：

- `<tag_name/edge_name>`：标签或者边类型的名称。
- `<prop_name>`：属性名称。
- `<prop_type>`：属性类型。支持 `bool`、`int`、`float`、`double`、`timestamp` 和 `string`，默认为 `string`。

例如 `student.name:string`、`follow.degree:double`。

12.2.3 配置示例

```yaml
# 连接的Nebula Graph版本，连接2.x时设置为v2。
version: v2

description: example

# 是否删除临时生成的日志和错误数据文件。
removeTempFiles: false

clientSettings:

  # nGQL语句执行失败的重试次数。
  retry: 3

  # Nebula Graph客户端并发数。
  concurrency: 10

  # 每个Nebula Graph客户端的缓存队列大小。
  channelBufferSize: 128

  # 指定数据要导入的Nebula Graph图空间。
  space: student

  # 连接信息。
  connection:
    user: root
    password: nebula
    address: 192.168.11.13:9669

  postStart:
    # 配置连接Nebula Graph服务器之后，在插入数据之前执行的一些操作。
    commands: |
      DROP SPACE IF EXISTS student;
      CREATE SPACE IF NOT EXISTS student(partition_num=5, replica_factor=1, vid_type=FIXED_STRING(20));
      USE student;
      CREATE TAG student(name string, age int,gender string);
      CREATE EDGE follow(degree int);

    # 执行上述命令后到执行插入数据命令之间的间隔。
    afterPeriod: 15s

  preStop:
    # 配置断开Nebula Graph服务器连接之前执行的一些操作。
    commands: |

# 错误等日志信息输出的文件路径。
logPath: ./err/test.log

# CSV文件相关设置。
files:

    # 数据文件的存放路径，如果使用相对路径，则会将路径和当前配置文件的目录拼接。本示例第一个数据文件为点的数据。
  - path: ./student_with_header.csv

    # 插入失败的数据文件存放路径，以便后面补写数据。
    failDataPath: ./err/studenterr.csv

    # 单批次插入数据的语句数量。
    batchSize: 10

    # 读取数据的行数限制。
    limit: 10

    # 是否按顺序在文件中插入数据行。如果为false，可以避免数据倾斜导致的导入速率降低。
    inOrder: true

    # 文件类型，当前仅支持csv。
    type: csv

    csv:
      # 是否有表头。
      withHeader: true

      # 是否有LABEL。
      withLabel: false
```


2021 Vesoft Inc.

```
      # 指定csv文件的分隔符。只支持一个字符的字符串分隔符。
      delimiter: ","

    schema:
      # Schema的类型，可选值为vertex和edge。
      type: vertex

    # 本示例第二个数据文件为边的数据。
  - path: ./follow_with_header.csv
    failDataPath: ./err/followerr.csv
    batchSize: 10
    limit: 10
    inOrder: true
    type: csv
    csv:
      withHeader: true
      withLabel: false
    schema:
      # Schema的类型为edge。
      type: edge
      edge:
        # 边类型名称。
        name: follow

        # 是否包含rank。
        withRanking: true
```

Note

点ID的数据类型需要和 `clientSettings.postStart.commands` 中的创建图空间语句的数据类型一致。

最后更新: 2021年5月26日

2021 Vesoft Inc.

12.3 无表头配置说明

对于无表头（header）的CSV文件，需要在配置文件里设置 `withHeader` 为 `false`，表示CSV文件中只含有数据（不含第一行表头），同时可能还需要设置数据类型、对应的列等。

12.3.1 示例文件

无表头的CSV文件示例如下：

- 点示例

  `student_without_header.csv` 的示例数据：

  ```
  student100,Monica,16,female
  student101,Mike,18,male
  student102,Jane,17,female
  ```

  第一列为点ID，后面三列为属性 `name`、`age` 和 `gender`。

- 边示例

  `follow_without_header.csv` 的示例数据：

  ```
  student100,student101,0,92.5
  student101,student100,1,85.6
  student101,student102,2,93.2
  student100,student102,1,96.2
  ```

  前两列的数据分别为起始点ID和目的点ID，第三列为rank，第四列为属性 `degree`。

12.3.2 配置示例

```yaml
# 连接的Nebula Graph版本，连接2.x时设置为v2。
version: v2

description: example

# 是否删除临时生成的日志和错误数据文件。
removeTempFiles: false

clientSettings:

  # nGQL语句执行失败的重试次数。
  retry: 3

  # Nebula Graph客户端并发数。
  concurrency: 10

  # 每个Nebula Graph客户端的缓存队列大小。
  channelBufferSize: 128

  # 指定数据要导入的Nebula Graph图空间。
  space: student

  # 连接信息。
  connection:
    user: root
    password: nebula
    address: 192.168.11.13:9669

  postStart:
    # 配置连接Nebula Graph服务器之后，在插入数据之前执行的一些操作。
    commands: |
      DROP SPACE IF EXISTS student;
      CREATE SPACE IF NOT EXISTS student(partition_num=5, replica_factor=1, vid_type=FIXED_STRING(20));
      USE student;
      CREATE TAG student(name string, age int,gender string);
      CREATE EDGE follow(degree int);

    # 执行上述命令后到执行插入数据命令之间的间隔。
    afterPeriod: 15s

  preStop:
    # 配置断开Nebula Graph服务器连接之前执行的一些操作。
    commands: |

# 错误等日志信息输出的文件路径。
```


2021 Vesoft Inc.

```
logPath: ./err/test.log

# CSV文件相关设置。
files:

    # 数据文件的存放路径，如果使用相对路径，则会将路径和当前配置文件的目录拼接。本示例第一个数据文件为点的数据。
  - path: ./student_without_header.csv

    # 插入失败的数据文件存放路径，以便后面补写数据。
    failDataPath: ./err/studenterr.csv

    # 单批次插入数据的语句数量。
    batchSize: 10

    # 读取数据的行数限制。
    limit: 10

    # 是否按顺序在文件中插入数据行。如果为false，可以避免数据倾斜导致的导入速率降低。
    inOrder: true

    # 文件类型，当前仅支持csv。
    type: csv

    csv:
      # 是否有表头。
      withHeader: false

      # 是否有LABEL。
      withLabel: false

      # 指定csv文件的分隔符。只支持一个字符的字符串分隔符。
      delimiter: ","

    schema:
      # Schema的类型，可选值为vertex和edge。
      type: vertex

      vertex:

        # 点ID设置。
        vid:
           # 点ID对应CSV文件中列的序号。CSV文件中列的序号从0开始。
           index: 0

           # 点ID的数据类型，可选值为int和string，分别对应Nebula Graph中的INT64和FIXED_STRING。
           type: string

        # 标签设置。
        tags:
            # 标签名称。
          - name: student

            # 标签内的属性设置。
            props:
                # 属性名称。
              - name: name

                # 属性数据类型。
                type: string

                # 属性对应CSV文件中列的序号。
                index: 1

              - name: age
                type: int
                index: 2
              - name: gender
                type: string
                index: 3

    # 本示例第二个数据文件为边的数据。
  - path: ./follow_without_header.csv
    failDataPath: ./err/followerr.csv
    batchSize: 10
    limit: 10
    inOrder: true
    type: csv
    csv:
      withHeader: false
      withLabel: false
    schema:
      # Schema的类型为edge。
      type: edge
      edge:
        # 边类型名称。
        name: follow

        # 是否包含rank。
        withRanking: true

        # 起始点ID设置。
        srcVID:
           # 数据类型。
           type: string
```


2021 Vesoft Inc.

```
      # 起始点ID对应CSV文件中列的序号。
      index: 0

   # 目的点ID设置。
   dstVID:
      type: string
      index: 1

   # rank设置。
   rank:
      # rank值对应CSV文件中列的序号。如果没有设置index，请务必在第三列设置rank的值。之后的列依次设置各属性。
      index: 2

   # 边类型内的属性设置。
   props:
        # 属性名称。
      - name: degree

        # 属性数据类型。
        type: double

        # 属性对应CSV文件中列的序号。
        index: 3
```

Note

- CSV文件中列的序号从0开始，即第一列的序号为0，第二列的序号为1。

- 点ID的数据类型需要和 `clientSettings.postStart.commands` 中的创建图空间语句的数据类型一致。

- 如果没有设置index字段指定列的序号，CSV文件必须遵守如下规则：

```
+ 在点数据文件中，第一列必须为点ID，后面的列为属性，且需要和配置文件内的顺序一一对应。

+ 在边数据文件中，第一列必须为起始点ID，第二列必须为目的点ID，如果`withRanking`为`true`，第三列必须为rank值，后面的列为属性，且需要和配置文件内的顺序一一对应。
```

最后更新: 2021年5月26日

2021 Vesoft Inc.

13. Nebula Spark Connector

Nebula Spark Connector是一个Spark连接器，提供通过Spark标准形式读写Nebula Graph数据的能力。Nebula Spark Connector由Reader和Writer两部分组成。

- Reader

  提供一个Spark SQL接口，用户可以使用该接口编程读取Nebula Graph图数据，单次读取一个点或边类型的数据，并将读取的结果组装成Spark的DataFrame。

- Writer

  提供一个Spark SQL接口，用户可以使用该接口编程将DataFrame格式的数据逐条或批量写入Nebula Graph。

更多使用说明请参见Nebula Spark Connector。

13.1 适用场景

Nebula Spark Connector适用于以下场景：

- 在不同的Nebula Graph集群之间迁移数据。
- 在同一个Nebula Graph集群内不同图空间之间迁移数据。
- Nebula Graph与其他数据源之间迁移数据。
- 结合Nebula Algorithm进行图计算。

13.2 优势

- 提供多种连接配置项，如超时时间、连接重试次数、执行重试次数等。
- 提供多种数据配置项，如写入数据时设置对应列为点ID、起始点ID、目的点ID或属性。
- Reader支持无属性读取和全属性读取。
- Reader支持将Nebula Graph数据读取成Graphx的VertexRDD和EdgeRDD，支持非Long型点ID。
- Nebula Spark Connector 2.0统一了SparkSQL的扩展数据源，统一采用DataSourceV2进行Nebula Graph数据扩展。

13.3 获取Nebula Spark Connector

13.3.1 编译打包

1. 克隆仓库 `nebula-spark-utils` 。

```
$ git clone https://github.com/vesoft-inc/nebula-spark-utils.git
```

2. 进入目录 `nebula-spark-connector` 。

```
$ cd nebula-spark-utils/nebula-spark-connector
```

3. 编译打包。

```
$ mvn clean package -Dmaven.test.skip=true -Dgpg.skip -Dmaven.javadoc.skip=true
```

编译完成后，在目录 `nebula-spark-connector/target` 下生成类似文件 `nebula-spark-connector-2.0.0-SHANPSHOT.jar` 。

2021 Vesoft Inc.

13.3.2 Maven远程仓库下载

下载地址

13.4 使用方法

使用Nebula Spark Connector读写Nebula Graph数据库时，只需要编写以下代码即可实现。

```
# 从Nebula Graph读取点边数据。
spark.read.nebula().loadVerticesToDF()
spark.read.nebula().loadEdgesToDF()

# 将dataframe数据作为点和边写入Nebula Graph中。
dataframe.write.nebula().writeVertices()
dataframe.write.nebula().writeEdges()
```

`nebula()` 接收两个配置参数，包括连接配置和读写配置。

13.4.1 从Nebula Graph读取数据

```
val config = NebulaConnectionConfig
  .builder()
  .withMetaAddress("127.0.0.1:9559")
  .withConenctionRetry(2)
  .withExecuteRetry(2)
  .withTimeout(6000)
  .build()

val nebulaReadVertexConfig: ReadNebulaConfig = ReadNebulaConfig
  .builder()
  .withSpace("test")
  .withLabel("person")
  .withNoColumn(false)
  .withReturnCols(List("birthday"))
  .withLimit(10)
  .withPartitionNum(10)
  .build()
val vertex = spark.read.nebula(config, nebulaReadVertexConfig).loadVerticesToDF()

val nebulaReadEdgeConfig: ReadNebulaConfig = ReadNebulaConfig
  .builder()
  .withSpace("test")
  .withLabel("knows")
  .withNoColumn(false)
  .withReturnCols(List("degree"))
  .withLimit(10)
  .withPartitionNum(10)
```


2021 Vesoft Inc.

```
  .build()
val edge = spark.read.nebula(config, nebulaReadEdgeConfig).loadEdgesToDF()
```

- `NebulaConnectionConfig` 是连接Nebula Graph的配置，说明如下。

参数	是否必须	说明
`withMetaAddress`	是	所有Meta服务的地址，多个地址用英文逗号（,）隔开，格式为 `ip1:port1,ip2:port2,...` 。读取数据不需要配置 `withGraphAddress` 。
`withConnectionRetry`	否	Nebula Java Client连接Nebula Graph的重试次数。默认值为 `1` 。
`withExecuteRetry`	否	Nebula Java Client执行查询语句的重试次数。默认值为 `1` 。
`withTimeout`	否	Nebula Java Client请求响应的超时时间。默认值为 `6000` ，单位：毫秒（ms）。

- `ReadNebulaConfig` 是读取Nebula Graph数据的配置，说明如下。

参数	是否必须	说明
`withSpace`	是	Nebula Graph图空间名称。
`withLabel`	是	Nebula Graph图空间内的标签或边类型名称。
`withNoColumn`	否	是否不读取属性。默认值为 `false` ，表示读取属性。取值为 `true` 时，表示不读取属性，此时 `withReturnCols` 配置无效。
`withReturnCols`	否	配置要读取的点或边的属性集。格式为 `List(property1,property2,...)` ，默认值为 `List()` ，表示读取全部属性。
`withLimit`	否	配置Nebula Java Storage Client一次从服务端读取的数据行数。默认值为1000。
`withPartitionNum`	否	配置读取Nebula Graph数据时Spark的分区数。默认值为100。该值的配置最好不超过图空间的的分片数量（partition_num）。
### 向Nebula Graph写入数据		

```
val config = NebulaConnectionConfig
  .builder()
  .withMetaAddress("127.0.0.1:9559")
  .withGraphAddress("127.0.0.1:9669")
  .withConenctionRetry(2)
  .build()

val nebulaWriteVertexConfig: WriteNebulaVertexConfig = WriteNebulaVertexConfig
  .builder()
  .withSpace("test")
  .withTag("person")
  .withVidField("id")
  .withVidPolicy("hash")
  .withVidAsProp(true)
  .withUser("root")
  .withPasswd("nebula")
  .withBatch(1000)
  .build()
df.write.nebula(config, nebulaWriteVertexConfig).writeVertices()

val nebulaWriteEdgeConfig: WriteNebulaEdgeConfig = WriteNebulaEdgeConfig
  .builder()
  .withSpace("test")
  .withEdge("friend")
  .withSrcIdField("src")
  .withSrcPolicy(null)
  .withDstIdField("dst")
  .withDstPolicy(null)
  .withRankField("degree")
  .withSrcAsProperty(true)
  .withDstAsProperty(true)
  .withRankAsProperty(true)
  .withUser("root")
  .withPasswd("nebula")
  .withBatch(1000)
```


2021 Vesoft Inc.

```
  .build()
df.write.nebula(config, nebulaWriteEdgeConfig).writeEdges()
```

- `NebulaConnectionConfig` 是连接Nebula Graph的配置，说明如下。

参数	是否必须	说明
`withMetaAddress`	是	所有Meta服务的地址，多个地址用英文逗号（,）隔开，格式为 `ip1:port1,ip2:port2,...` 。
`withGraphAddress`	是	Graph服务的地址，多个地址用英文逗号（,）隔开，格式为 `ip1:port1,ip2:port2,...` 。
`withConnectionRetry`	否	Nebula Java Client连接Nebula Graph的重试次数。默认值为 `1` 。

- `WriteNebulaVertexConfig` 是写入点的配置，说明如下。

参数	是否必须	说明
`withSpace`	是	Nebula Graph图空间名称。
`withTag`	是	写入点时需要关联的标签名称。
`withVidField`	是	DataFrame中作为点ID的列。
`withVidPolicy`	否	写入点ID时，采用的映射函数，Nebula Graph 2.0仅支持HASH。默认不做映射。
`withVidAsProp`	否	DataFrame中作为点ID的列是否也作为属性写入。默认值为 `false` 。如果配置为 `true` ，请确保标签中有和 `VidField` 相同的属性名。
`withUser`	否	Nebula Graph用户名。若未开启身份验证，无需配置用户名和密码。
`withPasswd`	否	Nebula Graph用户名对应的密码。
`withBatch`	是	一次写入的数据行数。默认值为 `1000` .

- `WriteNebulaEdgeConfig` 是写入边的配置，说明如下。

参数	是否必须	说明
`withSpace`	是	Nebula Graph图空间名称。
`withEdge`	是	写入边时需要关联的边类型名称。
`withSrcIdField`	是	DataFrame中作为起始点的列。
`withSrcPolicy`	否	写入起始点时，采用的映射函数，Nebula Graph 2.0仅支持HASH。默认不做映射。
`withDstIdField`	是	DataFrame中作为目的点的列。
`withDstPolicy`	否	写入目的点时，采用的映射函数，Nebula Graph 2.0仅支持HASH。默认不做映射。
`withRankField`	否	DataFrame中作为rank的列。默认不写入rank。
`withSrcAsProperty`	否	DataFrame中作为起始点的列是否也作为属性写入。默认值为 `false` 。如果配置为 `true` ，请确保边类型中有和 `SrcIdField` 相同的属性名。
`withDstAsProperty`	否	DataFrame中作为目的点的列是否也作为属性写入。默认值为 `false` 。如果配置为 `true` ，请确保边类型中有和 `DstIdField` 相同的属性名。
`withRankAsProperty`	否	DataFrame中作为rank的列是否也作为属性写入。默认值为 `false` 。如果配置为 `true` ，请确保边类型中有和 `RankField` 相同的属性名。
`withUser`	否	Nebula Graph用户名。若未开启身份验证，无需配置用户名和密码。
`withPasswd`	否	Nebula Graph用户名对应的密码。
`withBatch`	是	一次写入的数据行数。默认值为 `1000` .

2021 Vesoft Inc.

最后更新: 2021年5月24日

2021 Vesoft Inc.

14. Nebula Flink Connector

Nebula Flink Connector是一款帮助Flink用户快速访问Nebula Graph的连接器，支持从Nebula Graph图数据库中读取数据，或者将其他外部数据源读取的数据写入Nebula Graph图数据库。

更多使用说明请参见Nebula Flink Connector。

14.1 适用场景

Nebula Flink Connector适用于以下场景：

- 在不同的Nebula Graph集群之间迁移数据。
- 在同一个Nebula Graph集群内不同图空间之间迁移数据。
- Nebula Graph与其他数据源之间迁移数据。

最后更新: 2021年5月13日

2021 Vesoft Inc.

15. Nebula Algorithm

Nebula Algorithm （简称Algorithm）是一款基于GraphX的Spark应用程序，通过提交Spark任务的形式使用完整的算法工具对Nebula Graph数据库中的数据执行图计算，也可以通过编程形式调用lib库下的算法针对DataFrame执行图计算。

15.1 使用限制

点ID的数据必须为整数，即点ID可以是INT类型，或者是String类型但数据本身为整数。

15.2 支持算法

Nebula Algorithm支持的图计算算法如下。

算法名	说明	应用场景
PageRank	页面排序	网页排序、重点节点挖掘
Louvain	社区发现	社团挖掘、层次化聚类
KCore	K核	社区发现、金融风控
LabelPropagation	标签传播	资讯传播、广告推荐、社区发现
ConnectedComponent	联通分量	社区发现、孤岛发现
StronglyConnectedComponent	强联通分量	社区发现
ShortestPath	最短路径	路径规划、网络规划
TriangleCount	三角形计数	网络结构分析
BetweennessCentrality	介数中心性	关键节点挖掘，节点影响力计算
DegreeStatic	度统计	图结构分析

15.3 实现方法

Nebula Algorithm实现图计算的流程如下：

1. 利用Nebula Spark Connector从Nebula Graph数据库中读取图数据为DataFrame。
2. 将DataFrame转换为GraphX的图。
3. 调用GraphX提供的图算法（例如PageRank）或者自行实现的算法（例如Louvain社区发现）。

详细的实现方法可以参见相关Scala文件。

15.4 获取Nebula Algorithm

15.4.1 编译打包

1. 克隆仓库 `nebula-spark-utils` 。

```
$ git clone https://github.com/vesoft-inc/nebula-spark-utils.git
```

2. 进入目录 `nebula-algorithm` 。

```
$ cd nebula-spark-utils/nebula-algorithm
```


2021 Vesoft Inc.

3. 编译打包。

```
$ mvn clean package -Dgpg.skip -Dmaven.javadoc.skip=true -Dmaven.test.skip=true
```

编译完成后，在目录 `nebula-algorithm/target` 下生成类似文件 `nebula-algorithm-2.0.0.jar` 。

15.4.2 Maven远程仓库下载

下载地址

15.5 使用方法

15.5.1 调用算法接口（推荐）

`lib` 库中提供了10种常用图计算算法，用户可以通过编程调用的形式调用算法。

1. 在文件 `pom.xml` 中添加依赖。

```
<dependency>
<groupId>com.vesoft</groupId>
<artifactId>nebula-algorithm</artifactId>
<version>2.0.0</version>
</dependency>
```

2. 传入参数调用算法（以PageRank为例）。更多算法请参见测试用例。

> **Note**
>
> 执行算法的DataFrame默认第一列是起始点，第二列是目的点，第三列是边权重（非Nebula Graph中的rank）。

```
val prConfig = new PRConfig(5, 1.0)
val louvainResult = PageRankAlgo.apply(spark, data, prConfig, false)
```


2021 Vesoft Inc.

15.5.2 直接提交算法包

> **Note**
>
> 使用封装好的算法包有一定的局限性，例如落库到Nebula Graph时，落库的图空间中创建的标签的属性名称必须和代码内预设的名称保持一致。如果用户有开发能力，推荐使用第一种方法。

2021 Vesoft Inc.

1. 设置配置文件。

```
{
    # Spark相关配置
    spark: {
    app: {
        name: LPA
        # Spark分片数量
        partitionNum:100
    }
    master:local
    }

    data: {
    # 数据源，可选值为nebula、csv、json。
    source: nebula
    # 数据落库，即图计算的结果写入的目标，可选值为nebula、csv、json。
    sink: nebula
    # 算法是否需要权重。
    hasWeight: false
    }

    # Nebula Graph相关配置
    nebula: {
    # 数据源。Nebula Graph作为图计算的数据源时，nebula.read的配置才生效。
    read: {
        # 所有Meta服务的IP地址和端口，多个地址用英文逗号（,）分隔。格式："ip1:port1,ip2:port2"。
        metaAddress: "192.168.*.10:9559"
        # Nebula Graph图空间名称
        space: basketballplayer
        # Nebula Graph边类型，多个labels时，多个边的数据将合并。
        labels: ["serve"]
        # Nebula Graph每个边类型的属性名称，此属性将作为算法的权重列，请确保和边类型对应。
        weightCols: ["start_year"]
    }

    # 数据落库。图计算结果落库到Nebula Graph时，nebula.write的配置才生效。
    write:{
        # Graph服务的IP地址和端口，多个地址用英文逗号（,）分隔。格式："ip1:port1,ip2:port2"。
        graphAddress: "192.168.*.11:9669"
        # 所有Meta服务的IP地址和端口，多个地址用英文逗号（,）分隔。格式："ip1:port1,ip2:port2"。
        metaAddress: "192.168.*.12:9559"
        user:root
        pswd:nebula
        # Nebula Graph图空间名称
        space:nb
        # Nebula Graph标签名称，图计算结果会写入该标签。标签中的属性名称固定如下：
        # PageRank：pagerank
        # Louvain：louvain
        # ConnectedComponent：cc
        # StronglyConnectedComponent：scc
        # LabelPropagation：lpa
        # ShortestPath：shortestpath
        # DegreeStatic：degree、inDegree、outDegree
        # KCore：kcore
        # TriangleCount：tranglecpunt
        # BetweennessCentrality：betweennedss
        tag:pagerank
    }
    }

    local: {
    # 数据源。图计算的数据源为csv文件或json文件时，local.read的配置才生效。
    read:{
        filePath: "hdfs://127.0.0.1:9000/edge/work_for.csv"
        # 如果CSV文件没有表头，使用[_c0, _c1, _c2, ..., _cn]表示其表头，有表头或者是json文件时，直接使用表头名称即可。
        # 起始点ID列的表头。
        srcId:"_c0"
        # 目的点ID列的表头。
        dstId:"_c1"
        # 权重列的表头
        weight: "_c2"
        # csv文件是否有表头
        header: false
        # csv文件的分隔符
        delimiter:","
    }

    # 数据落库。图计算结果落库到csv文件或text文件时，local.write的配置才生效。
    write:{
        resultPath:/tmp/
    }
    }


    algorithm: {
    # 需要执行的算法，可选值为：pagerank、louvain、connectedcomponent、
    # labelpropagation、shortestpaths、degreestatic、kcore、
    # stronglyconnectedcomponent、trianglecount、betweenness
    executeAlgo: pagerank

    # PageRank参数
    pagerank: {
```


2021 Vesoft Inc.

```
        maxIter: 10
        resetProb: 0.15  # 默认为0.15
    }

    # Louvain参数
    louvain: {
        maxIter: 20
        internalIter: 10
        tol: 0.5
    }

   # ConnectedComponent/StronglyConnectedComponent参数
   connectedcomponent: {
       maxIter: 20
   }

   # LabelPropagation参数
   labelpropagation: {
       maxIter: 20
   }

    # ShortestPath参数
    shortestpaths: {
        # several vertices to compute the shortest path to all vertices.
        landmarks: "1"
    }

    # DegreeStatic参数
    degreestatic: {}

    # KCore参数
    kcore:{
        maxIter:10
        degree:1
    }

    # TriangleCount参数
    trianglecount:{}

    # BetweennessCentrality参数
    betweenness:{
        maxIter:5
    }
}
}
```

2. 提交图计算任务。

```
${SPARK_HOME}/bin/spark-submit --master <mode> --class com.vesoft.nebula.algorithm.Main <nebula-algorithm-2.0.0.jar_path> -p <application.conf_path>
```

示例：

```
${SPARK_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.algorithm.Main /root/nebula-spark-utils/nebula-algorithm/target/nebula-algorithm-2.0.0.jar -p /root/nebula-spark-utils/nebula-algorithm/src/main/resources/application.conf
```

最后更新: 2021年5月24日

2021 Vesoft Inc.

16. Nebula Exchange

16.1 认识Nebula Exchange

16.1.1 什么是Nebula Exchange

Nebula Exchange（简称Exchange）是一款Apache Spark™应用，用于在分布式环境中将集群中的数据批量迁移到Nebula Graph中，能支持多种不同格式的批式数据和流式数据的迁移。

Exchange由Reader、Processor和Writer三部分组成。Reader读取不同来源的数据返回DataFrame后，Processor遍历DataFrame的每一行，根据配置文件中 `fields` 的映射关系，按列名获取对应的值。在遍历指定批处理的行数后，Writer会将获取的数据一次性写入到Nebula Graph中。下图描述了Exchange完成数据转换和迁移的过程。

适用场景

Exchange适用于以下场景：

- 需要将来自Kafka、Pulsar平台的流式数据，如日志文件、网购数据、游戏内玩家活动、社交网站信息、金融交易大厅或地理空间服务，以及来自数据中心内所连接设备或仪器的遥测数据等转化为属性图的点或边数据，并导入Nebula Graph数据库。
- 需要从关系型数据库（如MySQL）或者分布式文件系统（如HDFS）中读取批式数据，如某个时间段内的数据，将它们转化为属性图的点或边数据，并导入 Nebula Graph数据库。
- 需要将大批量数据生成Nebula Graph能识别的SST文件，再导入Nebula Graph数据库。

2021 Vesoft Inc.

产品优点

Exchange具有以下优点：

- 适应性强：支持将多种不同格式或不同来源的数据导入Nebula Graph数据库，便于迁移数据。
- 支持导入SST：支持将不同来源的数据转换为SST文件，用于数据导入。
- 支持断点续传：导入数据时支持断点续传，有助于节省时间，提高数据导入效率。

> **Note**
>
> 目前仅迁移Neo4j数据时支持断点续传。

- 异步操作：会在源数据中生成一条插入语句，发送给Graph服务，最后再执行插入操作。
- 灵活性强：支持同时导入多个标签和边类型，不同标签和边类型可以是不同的数据来源或格式。
- 统计功能：使用Apache Spark™中的累加器统计插入操作的成功和失败次数。
- 易于使用：采用HOCON（Human-Optimized Config Object Notation）配置文件格式，具有面向对象风格，便于理解和操作。

数据格式和来源

Exchange 2.0支持将以下格式或来源的数据转换为Nebula Graph能识别的点和边数据：

- 存储在HDFS的数据，包括：
  - Apache Parquet
  - Apache ORC
  - JSON
  - CSV
- Apache HBase™
- 数据仓库：Hive
- 图数据库：Neo4j（Client版本2.4.5-M1）
- 关系型数据库：MySQL
- 流处理软件平台：Apache Kafka®
- 发布/订阅消息平台：Apache Pulsar 2.4.5

最后更新: 2021年5月14日

2021 Vesoft Inc.

16.1.2 使用限制

本文描述Exchange 2.0的一些使用限制。

Nebula Graph版本

Exchange client版本（即JAR包版本）和Nebula Graph的版本对应关系如下。

Exchange client版本	Nebula Graph版本
2.0-SNAPSHOT	v2-nightly
2.0.1	2.0.0、2.0.1
2.0.0	2.0.0、2.0.1

JAR包有两种获取方式：自行编译或者从maven仓库下载。

如果正在使用Nebula Graph 1.x，请使用Nebula Exchange 1.x。

使用环境

Exchange 2.0 支持以下操作系统：

- CentOS 7
- macOS

软件依赖

为保证Exchange正常工作，请确认机器上已经安装如下软件：

- Apache Spark：2.3.0及以上版本
- Java：1.8
- Scala：2.10.7、2.11.12或2.12.10

在以下使用场景，还需要部署Hadoop Distributed File System (HDFS)：

- 迁移HDFS的数据
- 生成SST文件

最后更新: 2021年5月24日

2021 Vesoft Inc.

16.2 编译Exchange

本文介绍如何编译Nebula Exchange。用户也可以直接下载编译完成的 `.jar` 文件。

16.2.1 前提条件

安装Maven。

16.2.2 编译流程

1. 克隆仓库 `nebula-java` 。

```
git clone -b v2.0.0-ga https://github.com/vesoft-inc/nebula-java.git
```

2. 切换到目录 `nebula-java` 。

```
cd nebula-java
```

3. 安装Nebula Java Client 2.0.0。

```
mvn clean install -Dmaven.test.skip=true -Dgpg.skip -Dmaven.javadoc.skip=true
```

> **Note**
>
> 安装后在本地Maven仓库会生成JAR文件，例如 `com/vesoft/client/2.0.0/client-2.0.0.jar` 。

4. 返回根目录克隆仓库 `nebula-spark-utils` 。

```
cd ~ && git clone -b v2.0.0 https://github.com/vesoft-inc/nebula-spark-utils.git
```

5. 切换到目录 `nebula-exchange` 。

```
cd nebula-spark-utils/nebula-exchange
```

6. 打包Nebula Exchange 2.0.0。

```
mvn clean package -Dmaven.test.skip=true -Dgpg.skip -Dmaven.javadoc.skip=true
```

> **Note**
>
> 如果报错 `Could not resolve dependencies for project xxx` ，请修改Maven安装目录下 `libexec/conf/settings.xml` 文件的 `mirror` 部分：
>
> ```
> <mirror>
> <id>alimaven</id>
> <mirrorOf>central</mirrorOf>
> <name>aliyun maven</name>
> <url>http://maven.aliyun.com/nexus/content/repositories/central/</url>
> </mirror>
> ```

编译成功后，用户可以在当前目录里查看到类似如下目录结构。

```
.
├── README-CN.md
├── README.md
├── pom.xml
├── src
│   ├── main
│   └── test
└── target
    ├── classes
    ├── classes.timestamp
    ├── maven-archiver
    ├── nebula-exchange-2.x.y-javadoc.jar
    ├── nebula-exchange-2.x.y-sources.jar
```


2021 Vesoft Inc.

```
    ├── nebula-exchange-2.x.y.jar
    ├── original-nebula-exchange-2.x.y.jar
    └── site
```

在 `target` 目录下，用户可以找到 `exchange-2.x.y.jar` 文件。

> **Note**
>
> JAR文件版本号会因Nebula Java Client的发布版本而变化。用户可以在Releases页面查看最新版本。

迁移数据时，用户可以参考配置文件 `target/classes/application.conf` 。

最后更新: 2021年5月24日

2021 Vesoft Inc.

16.3 参数说明

16.3.1 导入命令参数

完成配置文件修改后，可以运行以下命令将指定来源的数据导入Nebula Graph数据库。

```
<spark_install_path>/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange <nebula-exchange-2.x.y.jar_path> -c <csv_application.conf_path>
```

> **Note**
>
> JAR文件版本号以实际编译得到的JAR文件名称为准。

下表列出了命令的相关参数。

参数	是否必需	默认值	说明
`--class`	是	无	指定驱动的主类。
`--master`	是	无	指定Spark集群中master进程的URL。详情请参见master-urls。
`-c` / `--config`	是	无	指定配置文件的路径。
`-h` / `--hive`	否	`false`	添加这个参数表示支持从Hive中导入数据。
`-D` / `--dry`	否	`false`	添加这个参数表示检查配置文件的格式是否符合要求，但不会校验 `tags` 和 `edges` 的配置项是否正确。正式导入数据时不能添加这个参数。

最后更新: 2021年5月24日

2021 Vesoft Inc.

16.3.2 配置说明

本文介绍使用Nebula Exchange时如何修改配置文件 `application.conf` 。

修改配置文件之前，建议根据数据源复制并修改文件名称，便于区分。例如数据源为CSV文件，可以复制为 `csv_application.conf` 。

配置文件的内容主要分为如下几类：

- Spark相关配置
- Hive配置（可选）
- Nebula Graph相关配置
- 点配置
- 边配置

Spark相关配置

本文只列出部分Spark参数，更多参数请参见官方文档。

参数	数据类型	默认值	是否必须	说明
`spark.app.name`	string	`Nebula Exchange 2.0`	否	Spark驱动程序名称。
`spark.driver.cores`	int	`1`	否	驱动程序使用的CPU核数，仅适用于集群模式。
`spark.driver.maxResultSize`	string	`1G`	否	单个Spark操作（例如collect）时，所有分区的序列化结果的总大小限制（字节为单位）。最小值为1M，0表示无限制。
`spark.executor.memory`	string	`1G`	否	Spark驱动程序使用的内存量，可以指定单位，例如512M、1G。
`spark.cores.max`	int	`16`	否	当驱动程序以“粗粒度”共享模式在独立部署集群或Mesos集群上运行时，跨集群（而非从每台计算机）请求应用程序的最大CPU核数。如果未设置，则值为Spark的独立集群管理器上的 `spark.deploy.defaultCores` 或Mesos上的infinite（所有可用的内核）。

2021 Vesoft Inc.

Hive配置（可选）

如果Spark和Hive部署在不同集群，才需要配置连接Hive的参数，否则请忽略这些配置。

参数	数据类型	默认值	是否必须	说明
`hive.warehouse`	string	-	是	HDFS中的warehouse路径。用双引号括起路径，以 `hdfs://` 开头。
`hive.connectionURL`	string	-	是	JDBC连接的URL。例如 `"jdbc:mysql://127.0.0.1:3306/hive_spark?characterEncoding=UTF-8"` 。
`hive.connectionDriverName`	string	`"com.mysql.jdbc.Driver"`	是	驱动名称。
`hive.connectionUserName`	list[string]	-	是	连接的用户名。
`hive.connectionPassword`	list[string]	-	是	用户名对应的密码。

Nebula Graph相关配置

参数	数据类型	默认值	是否必须	说明
`nebula.address.graph`	list[string]	`["127.0.0.1:9669"]`	是	所有Graph服务的地址，包括IP和端口，多个地址用英文逗号（,）分隔。格式为 `["ip1:port1","ip2:port2","ip3:port3"]` 。
`nebula.address.meta`	list[string]	`["127.0.0.1:9559"]`	是	所有Meta服务的地址，包括IP和端口，多个地址用英文逗号（,）分隔。格式为 `["ip1:port1","ip2:port2","ip3:port3"]` 。
`nebula.user`	string	-	是	拥有Nebula Graph写权限的用户名。
`nebula.pswd`	string	-	是	用户名对应的密码。
`nebula.space`	string	-	是	需要导入数据的的图空间名称。
`nebula.connection.timeout`	int	`3000`	否	Thrift连接的超时时间，单位为 ms。
`nebula.connection.retry`	int	`3`	否	Thrift连接重试次数。
`nebula.execution.retry`	int	`3`	否	nGQL语句执行重试次数。
`nebula.error.max`	int	`32`	否	导入过程中的最大失败次数。当失败次数达到最大值时，提交的Spark作业将自动停止。
`nebula.error.output`	string	`/tmp/errors`	否	输出错误日志的路径。错误日志保存执行失败的nGQL语句。
`nebula.rate.limit`	int	`1024`	否	导入数据时令牌桶的令牌数量限制。
`nebula.rate.timeout`	int	`1000`	否	令牌桶中拿取令牌的超时时间，单位：毫秒。

点配置

对于不同的数据源，点的配置也有所不同，有很多通用参数，也有部分特有参数，配置时需要配置通用参数和不同数据源的特有参数。

2021 Vesoft Inc.

通用参数

参数	数据类型	默认值	是否必须	说明
`tags.name`	string	-	是	Nebula Graph中定义的标签名称。
`tags.type.source`	string	-	是	指定数据源。例如`csv`。
`tags.type.sink`	string	`client`	是	指定导入方式，可选值为`client`和`SST`（不支持）。
`tags.fields`	list[string]	-	是	属性对应的列的表头或列名。如果有表头或列名，请直接使用该名称。如果CSV文件没有表头，用`[_c0, _c1, _c2]`的形式表示第一列、第二列、第三列，以此类推。
`tags.nebula.fields`	list[string]	-	是	Nebula Graph中定义的属性名称，顺序必须和`tags.fields`一一对应。例如`[_c1, _c2]`对应`[name, age]`，表示第二列为属性name的值，第三列为属性age的值。
`tags.vertex.field`	string	-	是	点ID的列。例如CSV文件没有表头时，可以用`_c0`表示第一列的值作为点ID。
`tags.batch`	int	`256`	是	单批次写入Nebula Graph的最大点数量。
`tags.partition`	int	`32`	是	Spark分片数量。

PARQUET/JSON/ORC源特有参数

参数	数据类型	默认值	是否必须	说明
`tags.path`	string	-	是	HDFS中点数据文件的路径。用双引号括起路径，以`hdfs://`开头。

CSV源特有参数

参数	数据类型	默认值	是否必须	说明
`tags.path`	string	-	是	HDFS中点数据文件的路径。用双引号括起路径，以`hdfs://`开头。
`tags.separator`	string	`,`	是	分隔符。默认值为英文逗号（,）。
`tags.header`	bool	`true`	是	文件是否有表头。

HIVE源特有参数

参数	数据类型	默认值	是否必须	说明
`tags.exec`	string	-	是	查询数据源的语句。例如`select name,age from mooc.users`。

2021 Vesoft Inc.

NEO4J源特有参数

参数	数据类型	默认值	是否必须	说明
`tags.exec`	string	-	是	查询数据源的语句。例如 `match (n:label) return n.neo4j-field-0` 。
`tags.server`	string	`"bolt://127.0.0.1:7687"`	是	Neo4j服务器地址。
`tags.user`	string	-	是	拥有读取权限的Neo4j用户名。
`tags.password`	string	-	是	用户名对应密码。
`tags.database`	string	-	是	Neo4j中保存源数据的数据库名。
`tags.check_point_path`	string	`/tmp/test`	否	设置保存导入进度信息的目录，用于断点续传。如果未设置，表示不启用断点续传。

MYSQL源特有参数

参数	数据类型	默认值	是否必须	说明
`tags.host`	string	-	是	MySQL服务器地址。
`tags.port`	string	-	是	MySQL服务器端口。
`tags.database`	string	-	是	数据库名称。
`tags.table`	string	-	是	需要作为数据源的表名称。
`tags.user`	string	-	是	拥有读取权限的MySQL用户名。
`tags.password`	string	-	是	用户名对应密码。
`tags.sentence`	string	-	是	查询数据源的语句。例如 `"select teamid, name from basketball.team order by teamid;"` 。

HBASE源特有参数

参数	数据类型	默认值	是否必须	说明
`tags.host`	string	`127.0.0.1`	是	Hbase服务器地址。
`tags.port`	string	`2181`	是	Hbase服务器端口。
`tags.table`	string	-	是	需要作为数据源的表名称。
`tags.columnFamily`	string	-	是	表所属的列族（column family）。

PULSAR源特有参数

参数	数据类型	默认值	是否必须	说明
`tags.service`	string	`"pulsar://localhost:6650"`	是	Pulsar服务器地址。
`tags.admin`	string	`"http://localhost:8081"`	是	连接pulsar的admin.url。
`tags.options.<topic|topics|topicsPattern>`	string	-	是	Pulsar的选项，可以从 `topic` 、 `topics` 和 `topicsPattern` 选择一个进行配置。
`tags.interval.seconds`	int	`10`	是	读取消息的间隔。单位：秒。

2021 Vesoft Inc.

KAFKA源特有参数

参数	数据类型	默认值	是否必须	说明
`tags.service`	string	-	是	Kafka服务器地址。
`tags.topic`	string	-	是	消息类别。
`tags.interval.seconds`	int	`10`	是	读取消息的间隔。单位：秒。

边配置

对于不同的数据源，边的配置也有所不同，有很多通用参数，也有部分特有参数，配置时需要配置通用参数和不同数据源的特有参数。

边配置的不同数据源特有参数请参见上方点配置内的特有参数介绍，注意区分tags和edges即可。

通用参数

参数	数据类型	默认值	是否必须	说明
`edges.name`	string	-	是	Nebula Graph中定义的边类型名称。
`edges.type.source`	string	-	是	指定数据源。例如 `csv` 。
`edges.type.sink`	string	`client`	是	指定导入方式，可选值为 `client` 和 `SST` （不支持）。
`edges.fields`	list[string]	-	是	属性对应的列的表头或列名。如果有表头或列名，请直接使用该名称。如果CSV文件没有表头，用 `[_c0, _c1, _c2]` 的形式表示第一列、第二列、第三列，以此类推。
`edges.nebula.fields`	list[string]	-	是	Nebula Graph中定义的属性名称，顺序必须和 `edges.fields` 一一对应。例如 `[_c2, _c3]` 对应 `[start_year, end_year]` ，表示第三列为开始年份的值，第四列为结束年份的值。
`edges.source.field`	string	-	是	边的起始点的列。例如 `_c0` 表示第一列的值作为边的起始点。
`edges.target.field`	string	-	是	边的目的点的列。例如 `_c1` 表示第二列的值作为边的目的点。
`edges.ranking`	int	-	否	rank值的列。没有指定时，默认所有rank值为 `0` 。
`edges.batch`	int	`256`	是	单批次写入Nebula Graph的最大边数量。
`edges.partition`	int	`32`	是	Spark分片数量。

最后更新: 2021年5月25日

2021 Vesoft Inc.

16.4 使用Nebula Exchange

16.4.1 导入CSV文件数据

本文以一个示例说明如何使用Exchange将存储在HDFS上的CSV文件数据导入Nebula Graph。

如果要向Nebula Graph导入本地CSV文件，请参见Nebula Importer。

数据集

本文以basketballplayer数据集为例。

环境配置

本文示例在MacOS下完成，以下是相关的环境配置信息：

- 硬件规格：
  - CPU：1.7 GHz Quad-Core Intel Core i7
  - 内存：16 GB
- Spark：2.4.7 单机版
- Hadoop：2.9.2 伪分布式部署
- Nebula Graph：2.0.0。使用Docker Compose部署。

前提条件

开始导入数据之前，用户需要确认以下信息：

- 已经安装部署Nebula Graph并获取如下信息：
  - Graph服务和Meta服务的的IP地址和端口。
  - 拥有Nebula Graph写权限的用户名和密码。
- 已经编译Exchange。详情请参见编译Exchange。本示例中使用Exchange 2.0。
- 已经安装Spark。
- 了解Nebula Graph中创建Schema的信息，包括标签和边类型的名称、属性等。
- 已经安装并开启Hadoop服务。

2021 Vesoft Inc.

操作步骤

步骤 1：在NEBULA GRAPH中创建SCHEMA

分析CSV文件中的数据，按以下步骤在Nebula Graph中创建Schema：

1. 确认Schema要素。Nebula Graph中的Schema要素如下表所示。

要素	名称	属性
标签（Tag）	`player`	`name string, age int`
标签（Tag）	`team`	`name string`
边类型（Edge Type）	`follow`	`degree int`
边类型（Edge Type）	`serve`	`start_year int, end_year int`

2. 使用Nebula Console创建一个图空间**basketballplayer**，并创建一个Schema，如下所示。

```
## 创建图空间
nebula> CREATE SPACE basketballplayer \
        (partition_num = 10, \
        replica_factor = 1, \
        vid_type = FIXED_STRING(30));

## 选择图空间basketballplayer
nebula> USE basketballplayer;

## 创建标签player
nebula> CREATE TAG player(name string, age int);

## 创建标签team
nebula> CREATE TAG team(name string);

## 创建边类型follow
nebula> CREATE EDGE follow(degree int);

## 创建边类型serve
nebula> CREATE EDGE serve(start_year int, end_year int);
```

更多信息，请参见快速开始。

步骤 2：处理CSV文件

确认以下信息：

1. 处理CSV文件以满足Schema的要求。

> **Note**
>
> Exchange支持上传有表头或者无表头的CSV文件。

2. CSV文件必须存储在HDFS中，并已获取文件存储路径。

步骤 3：修改配置文件

编译Exchange后，复制 `target/classes/application.conf` 文件设置CSV数据源相关的配置。在本示例中，复制的文件名为 `csv_application.conf` 。各个配置项的详细说明请参见配置说明。

```
{
  # Spark相关配置
  spark: {
    app: {
      name: Nebula Exchange 2.0
    }
    driver: {
      cores: 1
      maxResultSize: 1G
    }
    executor: {
        memory:1G
    }

    cores {
      max: 16
```


2021 Vesoft Inc.

```
  }
}

# Nebula Graph相关配置
nebula: {
  address:{
    # 指定Graph服务和所有Meta服务的IP地址和端口。
    # 如果有多台服务器，地址之间用英文逗号（,）分隔。
    # 格式: "ip1:port","ip2:port","ip3:port"
    graph:["127.0.0.1:9669"]
    meta:["127.0.0.1:9559"]
  }

  # 指定拥有Nebula Graph写权限的用户名和密码。
  user: root
  pswd: nebula

  # 指定图空间名称。
  space: basketballplayer
  connection {
    timeout: 3000
    retry: 3
  }
  execution {
    retry: 3
  }
  error: {
    max: 32
    output: /tmp/errors
  }
  rate: {
    limit: 1024
    timeout: 1000
  }
}

# 处理点
tags: [
  # 设置标签player相关信息。
  {
    # 指定Nebula Graph中定义的标签名称。
    name: player
    type: {
      # 指定数据源，使用CSV。
      source: csv

      # 指定如何将点数据导入Nebula Graph：Client或SST。
      sink: client
    }

    # 指定CSV文件的HDFS路径。
    # 用双引号括起路径，以hdfs://开头。
    path: "hdfs://192.168.11.13:9000/data/vertex_player.csv"

    # 如果CSV文件没有表头，使用[_c0, _c1, _c2, ..., _cn]表示其表头，并将列指示为属性值的源。
    # 如果CSV文件有表头，则使用实际的列名。
    fields: [_c1, _c2]

    # 指定Nebula Graph中定义的属性名称。
    # fields与nebula.fields的顺序必须一一对应。
    nebula.fields: [age, name]

    # 指定一个列作为VID的源。
    # vertex的值必须与上述fields或者csv.fields中的列名保持一致。
    # 目前，Nebula Graph 2.0.0仅支持字符串或整数类型的VID。
    # 不要使用vertex.policy映射。
    vertex: {
      field:_c0
      # policy:hash
    }

    # 指定的分隔符。默认值为英文逗号（,）。
    separator: ","

    # 如果CSV文件有表头，请将header设置为true。
    # 如果CSV文件没有表头，请将header设置为false。默认值为false。
    header: false

    # 指定单批次写入Nebula Graph的最大点数量。
    batch: 256

    # 指定Spark分片数量。
    partition: 32
  }

  # 设置标签team相关信息。
  {
    # 指定Nebula Graph中定义的标签名称。
    name: team
    type: {
      # 指定数据源，使用CSV。
      source: csv

      # 指定如何将点数据导入Nebula Graph：Client或SST。
```


2021 Vesoft Inc.

```
        sink: client
      }

      # 指定CSV文件的HDFS路径。
      # 用双引号括起路径，以hdfs://开头。
      path: "hdfs://192.168.11.13:9000/data/vertex_team.csv"

      # 如果CSV文件没有表头，使用[_c0, _c1, _c2, ..., _cn]表示其表头，并将列指示为属性值的源。
      # 如果CSV文件有表头，则使用实际的列名。
      fields: [_c1]

      # 指定Nebula Graph中定义的属性名称。
      # fields与nebula.fields的顺序必须一一对应。
      nebula.fields: [name]

      # 指定一个列作为VID的源。
      # vertex的值必须与上述fields或者csv.fields中的列名保持一致。
      # 目前，Nebula Graph 2.0.0仅支持字符串或整数类型的VID。
      # 不要使用vertex.policy映射。
      vertex: {
        field:_c0
        # policy:hash
      }

      # 指定的分隔符。默认值为英文逗号（,）。
      separator: ","

      # 如果CSV文件有表头，请将header设置为true。
      # 如果CSV文件没有表头，请将header设置为false。默认值为false。
      header: false

      # 指定单批次写入Nebula Graph的最大点数量。
      batch: 256

      # 指定Spark分片数量。
      partition: 32
    }


    # 如果需要添加更多点，请参考前面的配置进行添加。
  ]
  # 处理边
  edges: [
    # 设置边类型follow相关信息。
    {
      # 指定Nebula Graph中定义的边类型名称。
      name: follow
      type: {
        # 指定数据源，使用CSV。
        source: csv

        # 指定如何将点数据导入Nebula Graph：Client或SST。
        sink: client
      }

      # 指定CSV文件的HDFS路径。
      # 用双引号括起路径，以hdfs://开头。
      path: "hdfs://192.168.11.13:9000/data/edge_follow.csv"

      # 如果CSV文件没有表头，使用[_c0, _c1, _c2, ..., _cn]表示其表头，并将列指示为属性值的源。
      # 如果CSV文件有表头，则使用实际的列名。
      fields: [_c2]

      # 指定Nebula Graph中定义的属性名称。
      # fields与nebula.fields的顺序必须一一对应。
      nebula.fields: [degree]

      # 指定一个列作为起始点和目的点的源。
      # vertex的值必须与上述fields或者csv.fields中的列名保持一致。
      # 目前，Nebula Graph 2.0.0仅支持字符串或整数类型的VID。
      # 不要使用vertex.policy映射。
      source: {
        field: _c0
      }
      target: {
        field: _c1
      }

      # 指定的分隔符。默认值为英文逗号（,）。
      separator: ","

      # 指定一个列作为rank的源(可选)。

      #ranking: rank

      # 如果CSV文件有表头，请将header设置为true。
      # 如果CSV文件没有表头，请将header设置为false。默认值为false。
      header: false

      # 指定单批次写入Nebula Graph的最大边数量。
      batch: 256

      # 指定Spark分片数量。
      partition: 32
```


2021 Vesoft Inc.

```
    }

    # 设置边类型serve相关信息。
    {
      # 指定Nebula Graph中定义的边类型名称。
      name: serve
      type: {
        # 指定数据源，使用CSV。
        source: csv

        # 指定如何将点数据导入Nebula Graph：Client或SST。
        sink: client
      }

      # 指定CSV文件的HDFS路径。
      # 用双引号括起路径，以hdfs://开头。
      path: "hdfs://192.168.11.13:9000/data/edge_serve.csv"

      # 如果CSV文件没有表头，使用[_c0, _c1, _c2, ..., _cn]表示其表头，并将列指示为属性值的源。
      # 如果CSV文件有表头，则使用实际的列名。
      fields: [_c2,_c3]

      # 指定Nebula Graph中定义的属性名称。
      # fields与nebula.fields的顺序必须一一对应。
      nebula.fields: [start_year, end_year]

      # 指定一个列作为起始点和目的点的源。
      # vertex的值必须与上述fields或者csv.fields中的列名保持一致。
      # 目前，Nebula Graph 2.0.0仅支持字符串或整数类型的VID。
      # 不要使用vertex.policy映射。
      source: {
        field: _c0
      }
      target: {
        field: _c1
      }

      # 指定的分隔符。默认值为英文逗号（,）。
      separator: ","

      # 指定一个列作为rank的源(可选)。
      #ranking: _c5

      # 如果CSV文件有表头，请将header设置为true。
      # 如果CSV文件没有表头，请将header设置为false。默认值为false。
      header: false

      # 指定单批次写入Nebula Graph的最大边数量。
      batch: 256

      # 指定Spark分片数量。
      partition: 32
    }

  ]
  # 如果需要添加更多边，请参考前面的配置进行添加。
}
```

步骤 4：向NEBULA GRAPH导入数据

运行如下命令将CSV文件数据导入到Nebula Graph中。关于参数的说明，请参见导入命令参数。

```
${SPARK_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange <nebula-exchange-2.0.0.jar_path> -c <csv_application.conf_path>
```

> **Note**
>
> JAR包有两种获取方式：自行编译或者从maven仓库下载。

示例：

```
${SPARK_HOME}/bin/spark-submit  --master "local" --class com.vesoft.nebula.exchange.Exchange  /root/nebula-spark-utils/nebula-exchange/target/nebula-exchange-2.0.0.jar  -c /root/nebula-spark-utils/nebula-exchange/target/classes/csv_application.conf
```

用户可以在返回信息中搜索 `batchSuccess.<tag_name/edge_name>`，确认成功的数量。例如例如 `batchSuccess.follow: 300` 。

步骤 5：（可选）验证数据

用户可以在Nebula Graph客户端（例如Nebula Graph Studio）中执行查询语句，确认数据是否已导入。例如：

```
GO FROM "player100" OVER follow;
```


2021 Vesoft Inc.

用户也可以使用命令 `SHOW STATS` 查看统计数据。

步骤 6：（如有）在NEBULA GRAPH中重建索引

导入数据后，用户可以在Nebula Graph中重新创建并重建索引。详情请参见索引介绍。

最后更新: 2021年5月26日

2021 Vesoft Inc.

16.4.2 导入JSON文件数据

本文以一个示例说明如何使用Exchange将存储在HDFS上的JSON文件数据导入Nebula Graph。

数据集

本文以basketballplayer数据集为例。部分示例数据如下：

- player

```
{"id":"player100","age":42,"name":"Tim Duncan"}
{"id":"player101","age":36,"name":"Tony Parker"}
{"id":"player102","age":33,"name":"LaMarcus Aldridge"}
{"id":"player103","age":32,"name":"Rudy Gay"}
...
```

- team

```
{"id":"team200","name":"Warriors"}
{"id":"team201","name":"Nuggets"}
...
```

- follow

```
{"src":"player100","dst":"player101","degree":95}
{"src":"player101","dst":"player102","degree":90}
...
```

- serve

```
{"src":"player100","dst":"team204","start_year":"1997","end_year":"2016"}
{"src":"player101","dst":"team204","start_year":"1999","end_year":"2018"}
...
```

环境配置

本文示例在MacOS下完成，以下是相关的环境配置信息：

- 硬件规格：
  - CPU：1.7 GHz Quad-Core Intel Core i7
  - 内存：16 GB
- Spark：2.3.0，单机版
- Hadoop：2.9.2，伪分布式部署
- Nebula Graph：2.0.0。使用Docker Compose部署。

2021 Vesoft Inc.

前提条件

开始导入数据之前，用户需要确认以下信息：

- 已经安装部署Nebula Graph并获取如下信息：
  - Graph服务和Meta服务的的IP地址和端口。
  - 拥有Nebula Graph写权限的用户名和密码。
- 已经编译Exchange。详情请参见编译Exchange。本示例中使用Exchange 2.0。
- 已经安装Spark。
- 了解Nebula Graph中创建Schema的信息，包括标签和边类型的名称、属性等。
- 已经安装并开启Hadoop服务。

操作步骤

步骤 1：在NEBULA GRAPH中创建SCHEMA

分析文件中的数据，按以下步骤在Nebula Graph中创建Schema：

1. 确认Schema要素。Nebula Graph中的Schema要素如下表所示。

要素	名称	属性
标签（Tag）	`player`	`name string, age int`
标签（Tag）	`team`	`name string`
边类型（Edge Type）	`follow`	`degree int`
边类型（Edge Type）	`serve`	`start_year int, end_year int`

2. 使用Nebula Console创建一个图空间**basketballplayer**，并创建一个Schema，如下所示。

```
## 创建图空间
nebula> CREATE SPACE basketballplayer \
        (partition_num = 10, \
        replica_factor = 1, \
        vid_type = FIXED_STRING(30));

## 选择图空间basketballplayer
nebula> USE basketballplayer;

## 创建标签player
nebula> CREATE TAG player(name string, age int);

## 创建标签team
nebula> CREATE TAG team(name string);

## 创建边类型follow
nebula> CREATE EDGE follow(degree int);

## 创建边类型serve
nebula> CREATE EDGE serve(start_year int, end_year int);
```

更多信息，请参见快速开始。

步骤 2：处理JSON文件

确认以下信息：

1. 处理JSON文件以满足Schema的要求。
2. JSON文件必须存储在HDFS中，并已获取文件存储路径。

步骤 3. 修改配置文件

编译Exchange后，复制 `target/classes/application.conf` 文件设置JSON数据源相关的配置。在本示例中，复制的文件名为 `json_application.conf` 。各个配置项的详细说明请参见配置说明。

2021 Vesoft Inc.

```
{
  # Spark相关配置
  spark: {
    app: {
      name: Nebula Exchange 2.0
    }
    driver: {
      cores: 1
      maxResultSize: 1G
    }
    executor: {
        memory:1G
    }

    cores {
      max: 16
    }
  }

  # Nebula Graph相关配置
  nebula: {
    address:{
      # 指定Graph服务和所有Meta服务的IP地址和端口。
      # 如果有多台服务器，地址之间用英文逗号（,）分隔。
      # 格式: "ip1:port","ip2:port","ip3:port"
      graph:["127.0.0.1:9669"]
      meta:["127.0.0.1:9559"]
    }

    # 指定拥有Nebula Graph写权限的用户名和密码。
    user: root
    pswd: nebula

    # 指定图空间名称。
    space: basketballplayer
    connection {
      timeout: 3000
      retry: 3
    }
    execution {
      retry: 3
    }
    error: {
      max: 32
      output: /tmp/errors
    }
    rate: {
      limit: 1024
      timeout: 1000
    }
  }

  # 处理点
  tags: [
    # 设置标签player相关信息。
    {
      # 指定Nebula Graph中定义的标签名称。
      name: player
      type: {
        # 指定数据源，使用JSON。
        source: json

        # 指定如何将点数据导入Nebula Graph：Client或SST。
        sink: client
      }

      # 指定JSON文件的HDFS路径。
      # 用双引号括起路径，以hdfs://开头。
      path: "hdfs://192.168.11.13:9000/data/vertex_player.json"

      # 在fields里指定JSON文件中key名称，其对应的value会作为Nebula Graph中指定属性的数据源。
      # 如果需要指定多个值，用英文逗号（,）隔开。
      fields: [age,name]

      # 指定Nebula Graph中定义的属性名称。
      # fields与nebula.fields的顺序必须一一对应。
      nebula.fields: [age, name]

      # 指定一个列作为VID的源。
      # vertex的值必须与JSON文件中的字段保持一致。
      # 目前，Nebula Graph 2.0.0仅支持字符串或整数类型的VID。
      # 不要使用vertex.policy映射。
      vertex: {
        field:id
      }

      # 指定单批次写入Nebula Graph的最大点数量。
      batch: 256

      # 指定Spark分片数量。
      partition: 32
    }

    # 设置标签team相关信息。
```


2021 Vesoft Inc.

```
    {
      # 指定Nebula Graph中定义的标签名称。
      name: team
      type: {
        # 指定数据源，使用JSON。
        source: json

        # 指定如何将点数据导入Nebula Graph：Client或SST。
        sink: client
      }

      # 指定JSON文件的HDFS路径。
      # 用双引号括起路径，以hdfs://开头。
      path: "hdfs://192.168.11.13:9000/data/vertex_team.json"

      # 在fields里指定JSON文件中key名称，其对应的value会作为Nebula Graph中指定属性的数据源。
      # 如果需要指定多个值，用英文逗号（,）隔开。
      fields: [name]

      # 指定Nebula Graph中定义的属性名称。
      # fields与nebula.fields的顺序必须一一对应。
      nebula.fields: [name]

      # 指定一个列作为VID的源。
      # vertex的值必须与JSON文件中的字段保持一致。
      # 目前，Nebula Graph 2.0.0仅支持字符串或整数类型的VID。
      # 不要使用vertex.policy映射。
      vertex: {
        field:id
      }


      # 指定单批次写入Nebula Graph的最大点数量。
      batch: 256

      # 指定Spark分片数量。
      partition: 32
    }


    # 如果需要添加更多点，请参考前面的配置进行添加。
  ]
  # 处理边
  edges: [
    # 设置边类型follow相关信息。
    {
      # 指定Nebula Graph中定义的边类型名称。
      name: follow
      type: {
        # 指定数据源，使用JSON。
        source: json

        # 指定如何将点数据导入Nebula Graph：Client或SST。
        sink: client
      }

      # 指定JSON文件的HDFS路径。
      # 用双引号括起路径，以hdfs://开头。
      path: "hdfs://192.168.11.13:9000/data/edge_follow.json"

      # 在fields里指定JSON文件中key名称，其对应的value会作为Nebula Graph中指定属性的数据源。
      # 如果需要指定多个值，用英文逗号（,）隔开。
      fields: [degree]

      # 指定Nebula Graph中定义的属性名称。
      # fields与nebula.fields的顺序必须一一对应。
      nebula.fields: [degree]

      # 指定一个列作为起始点和目的点的源。
      # vertex的值必须与JSON文件中的字段保持一致。
      # 目前，Nebula Graph 2.0.0仅支持字符串或整数类型的VID。
      # 不要使用vertex.policy映射。
      source: {
        field: src
      }
      target: {
        field: dst
      }


      # 指定一个列作为rank的源(可选)。
      #ranking: rank

      # 指定单批次写入Nebula Graph的最大边数量。
      batch: 256

      # 指定Spark分片数量。
      partition: 32
    }

    # 设置边类型serve相关信息。
    {
      # 指定Nebula Graph中定义的边类型名称。
      name: serve
```


2021 Vesoft Inc.

```
      type: {
        # 指定数据源，使用JSON。
        source: json

        # 指定如何将点数据导入Nebula Graph：Client或SST。
        sink: client
      }

      # 指定JSON文件的HDFS路径。
      # 用双引号括起路径，以hdfs://开头。
      path: "hdfs://192.168.11.13:9000/data/edge_serve.json"

      # 在fields里指定JSON文件中key名称，其对应的value会作为Nebula Graph中指定属性的数据源。
      # 如果需要指定多个值，用英文逗号（,）隔开。
      fields: [start_year,end_year]

      # 指定Nebula Graph中定义的属性名称。
      # fields与nebula.fields的顺序必须一一对应。
      nebula.fields: [start_year, end_year]

      # 指定一个列作为起始点和目的点的源。
      # vertex的值必须与JSON文件中的字段保持一致。
      # 目前，Nebula Graph 2.0.0仅支持字符串或整数类型的VID。
      # 不要使用vertex.policy映射。
      source: {
        field: src
      }
      target: {
        field: dst
      }


      # 指定一个列作为rank的源(可选)。
      #ranking: _c5


      # 指定单批次写入Nebula Graph的最大边数量。
      batch: 256

      # 指定Spark分片数量。
      partition: 32
    }

  ]
  # 如果需要添加更多边，请参考前面的配置进行添加。
}
```

步骤 4：向NEBULA GRAPH导入数据

运行如下命令将JSON文件数据导入到Nebula Graph中。关于参数的说明，请参见导入命令参数。

```
${SPARK_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange <nebula-exchange-2.0.0.jar_path> -c <json_application.conf_path>
```

> **Note**
>
> JAR包有两种获取方式：自行编译或者从maven仓库下载。

示例：

```
${SPARK_HOME}/bin/spark-submit  --master "local" --class com.vesoft.nebula.exchange.Exchange  /root/nebula-spark-utils/nebula-exchange/target/nebula-exchange-2.0.0.jar  -c /root/nebula-spark-utils/nebula-exchange/target/classes/json_application.conf
```

用户可以在返回信息中搜索 `batchSuccess.<tag_name/edge_name>`，确认成功的数量。例如例如 `batchSuccess.follow: 300` 。

步骤 5：（可选）验证数据

用户可以在Nebula Graph客户端（例如Nebula Graph Studio）中执行查询语句，确认数据是否已导入。例如：

```
GO FROM "player100" OVER follow;
```

用户也可以使用命令 `SHOW STATS` 查看统计数据。

步骤 6：（如有）在NEBULA GRAPH中重建索引

导入数据后，用户可以在Nebula Graph中重新创建并重建索引。详情请参见索引介绍。

2021 Vesoft Inc.

最后更新: 2021年5月26日

2021 Vesoft Inc.

16.4.3 导入ORC文件数据

本文以一个示例说明如何使用Exchange将存储在HDFS上的ORC文件数据导入Nebula Graph。

如果要向Nebula Graph导入本地ORC文件，请参见Nebula Importer。

数据集

本文以basketballplayer数据集为例。

环境配置

本文示例在MacOS下完成，以下是相关的环境配置信息：

- 硬件规格：
  - CPU：1.7 GHz Quad-Core Intel Core i7
  - 内存：16 GB
- Spark：2.4.7 单机版
- Hadoop：2.9.2 伪分布式部署
- Nebula Graph：2.0.0。使用Docker Compose部署。

前提条件

开始导入数据之前，用户需要确认以下信息：

- 已经安装部署Nebula Graph并获取如下信息：
  - Graph服务和Meta服务的的IP地址和端口。
  - 拥有Nebula Graph写权限的用户名和密码。
- 已经编译Exchange。详情请参见编译Exchange。本示例中使用Exchange 2.0。
- 已经安装Spark。
- 了解Nebula Graph中创建Schema的信息，包括标签和边类型的名称、属性等。
- 已经安装并开启Hadoop服务。

2021 Vesoft Inc.

操作步骤

步骤 1：在NEBULA GRAPH中创建SCHEMA

分析ORC文件中的数据，按以下步骤在Nebula Graph中创建Schema：

1. 确认Schema要素。Nebula Graph中的Schema要素如下表所示。

要素	名称	属性
标签（Tag）	`player`	`name string, age int`
标签（Tag）	`team`	`name string`
边类型（Edge Type）	`follow`	`degree int`
边类型（Edge Type）	`serve`	`start_year int, end_year int`

2. 使用Nebula Console创建一个图空间**basketballplayer**，并创建一个Schema，如下所示。

```
## 创建图空间
nebula> CREATE SPACE basketballplayer \
        (partition_num = 10, \
        replica_factor = 1, \
        vid_type = FIXED_STRING(30));

## 选择图空间basketballplayer
nebula> USE basketballplayer;

## 创建标签player
nebula> CREATE TAG player(name string, age int);

## 创建标签team
nebula> CREATE TAG team(name string);

## 创建边类型follow
nebula> CREATE EDGE follow(degree int);

## 创建边类型serve
nebula> CREATE EDGE serve(start_year int, end_year int);
```

更多信息，请参见快速开始。

步骤 2：处理ORC文件

确认以下信息：

1. 处理ORC文件以满足Schema的要求。
2. ORC文件必须存储在HDFS中，并已获取文件存储路径。

步骤 3：修改配置文件

编译Exchange后，复制`target/classes/application.conf`文件设置ORC数据源相关的配置。在本示例中，复制的文件名为`orc_application.conf`。各个配置项的详细说明请参见配置说明。

```
{
  # Spark相关配置
  spark: {
    app: {
      name: Nebula Exchange 2.0
    }
    driver: {
      cores: 1
      maxResultSize: 1G
    }
    executor: {
        memory:1G
    }

    cores {
      max: 16
    }
  }

  # Nebula Graph相关配置
  nebula: {
    address:{
      # 指定Graph服务和所有Meta服务的IP地址和端口。
      # 如果有多台服务器，地址之间用英文逗号（,）分隔。
      # 格式: "ip1:port","ip2:port","ip3:port"
```


2021 Vesoft Inc.

```
    graph:["127.0.0.1:9669"]
    meta:["127.0.0.1:9559"]
  }

  # 指定拥有Nebula Graph写权限的用户名和密码。
  user: root
  pswd: nebula

  # 指定图空间名称。
  space: basketballplayer
  connection {
    timeout: 3000
    retry: 3
  }
  execution {
    retry: 3
  }
  error: {
    max: 32
    output: /tmp/errors
  }
  rate: {
    limit: 1024
    timeout: 1000
  }
}

# 处理点
tags: [
  # 设置标签player相关信息。
  {
    # 指定Nebula Graph中定义的标签名称。
    name: player
    type: {
      # 指定数据源，使用ORC。
      source: orc

      # 指定如何将点数据导入Nebula Graph：Client或SST。
      sink: client
    }

    # 指定ORC文件的HDFS路径。
    # 用双引号括起路径，以hdfs://开头。
    path: "hdfs://192.168.11.13:9000/data/vertex_player.orc"

    # 在fields里指定ORC文件中key名称，其对应的value会作为Nebula Graph中指定属性的数据源。
    # 如果需要指定多个值，用英文逗号（,）隔开。
    fields: [age,name]

    # 指定Nebula Graph中定义的属性名称。
    # fields与nebula.fields的顺序必须一一对应。
    nebula.fields: [age, name]

    # 指定一个列作为VID的源。
    # vertex的值必须与ORC文件中的字段保持一致。
    # 目前，Nebula Graph 2.0.0仅支持字符串或整数类型的VID。
    # 不要使用vertex.policy映射。
    vertex: {
      field:id
    }

    # 指定单批次写入Nebula Graph的最大点数量。
    batch: 256

    # 指定Spark分片数量。
    partition: 32
  }

  # 设置标签team相关信息。
  {
    # 指定Nebula Graph中定义的标签名称。
    name: team
    type: {
      # 指定数据源，使用ORC。
      source: orc

      # 指定如何将点数据导入Nebula Graph：Client或SST。
      sink: client
    }

    # 指定ORC文件的HDFS路径。
    # 用双引号括起路径，以hdfs://开头。
    path: "hdfs://192.168.11.13:9000/data/vertex_team.orc"

    # 在fields里指定ORC文件中key名称，其对应的value会作为Nebula Graph中指定属性的数据源。
    # 如果需要指定多个值，用英文逗号（,）隔开。
    fields: [name]

    # 指定Nebula Graph中定义的属性名称。
    # fields与nebula.fields的顺序必须一一对应。
    nebula.fields: [name]

    # 指定一个列作为VID的源。
    # vertex的值必须与ORC文件中的字段保持一致。
```


2021 Vesoft Inc.

```
      # 目前，Nebula Graph 2.0.0仅支持字符串或整数类型的VID。
      # 不要使用vertex.policy映射。
      vertex: {
        field:id
      }


      # 指定单批次写入Nebula Graph的最大点数量。
      batch: 256

      # 指定Spark分片数量。
      partition: 32
    }


    # 如果需要添加更多点，请参考前面的配置进行添加。
  ]
  # 处理边
  edges: [
    # 设置边类型follow相关信息。
    {
      # 指定Nebula Graph中定义的边类型名称。
      name: follow
      type: {
        # 指定数据源，使用ORC。
        source: orc

        # 指定如何将点数据导入Nebula Graph：Client或SST。
        sink: client
      }

      # 指定ORC文件的HDFS路径。
      # 用双引号括起路径，以hdfs://开头。
      path: "hdfs://192.168.11.13:9000/data/edge_follow.orc"

      # 在fields里指定ORC文件中key名称，其对应的value会作为Nebula Graph中指定属性的数据源。
      # 如果需要指定多个值，用英文逗号（,）隔开。
      fields: [degree]

      # 指定Nebula Graph中定义的属性名称。
      # fields与nebula.fields的顺序必须一一对应。
      nebula.fields: [degree]

      # 指定一个列作为起始点和目的点的源。
      # vertex的值必须与ORC文件中的字段保持一致。
      # 目前，Nebula Graph 2.0.0仅支持字符串或整数类型的VID。
      # 不要使用vertex.policy映射。
      source: {
        field: src
      }
      target: {
        field: dst
      }


      # 指定一个列作为rank的源(可选)。
      #ranking: rank

      # 指定单批次写入Nebula Graph的最大边数量。
      batch: 256

      # 指定Spark分片数量。
      partition: 32
    }

    # 设置边类型serve相关信息。
    {
      # 指定Nebula Graph中定义的边类型名称。
      name: serve
      type: {
        # 指定数据源，使用ORC。
        source: orc

        # 指定如何将点数据导入Nebula Graph：Client或SST。
        sink: client
      }

      # 指定ORC文件的HDFS路径。
      # 用双引号括起路径，以hdfs://开头。
      path: "hdfs://192.168.11.13:9000/data/edge_serve.orc"

      # 在fields里指定ORC文件中key名称，其对应的value会作为Nebula Graph中指定属性的数据源。
      # 如果需要指定多个值，用英文逗号（,）隔开。
      fields: [start_year,end_year]

      # 指定Nebula Graph中定义的属性名称。
      # fields与nebula.fields的顺序必须一一对应。
      nebula.fields: [start_year, end_year]

      # 指定一个列作为起始点和目的点的源。
      # vertex的值必须与ORC文件中的字段保持一致。
      # 目前，Nebula Graph 2.0.0仅支持字符串或整数类型的VID。
      # 不要使用vertex.policy映射。
      source: {
```


2021 Vesoft Inc.

```
        field: src
      }
      target: {
        field: dst
      }


      # 指定一个列作为rank的源(可选)。
      #ranking: _c5


      # 指定单批次写入Nebula Graph的最大边数量。
      batch: 256

      # 指定Spark分片数量。
      partition: 32
    }

  ]
  # 如果需要添加更多边，请参考前面的配置进行添加。
}
```

步骤 4：向NEBULA GRAPH导入数据

运行如下命令将ORC文件数据导入到Nebula Graph中。关于参数的说明，请参见导入命令参数。

```
${SPARK_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange <nebula-exchange-2.0.0.jar_path> -c <orc_application.conf_path>
```

> **Note**
>
> JAR包有两种获取方式：自行编译或者从maven仓库下载。

示例：

```
${SPARK_HOME}/bin/spark-submit  --master "local" --class com.vesoft.nebula.exchange.Exchange  /root/nebula-spark-utils/nebula-exchange/target/nebula-exchange-2.0.0.jar  -c /root/nebula-spark-utils/nebula-exchange/target/classes/orc_application.conf
```

用户可以在返回信息中搜索 `batchSuccess.<tag_name/edge_name>`，确认成功的数量。例如例如 `batchSuccess.follow: 300` 。

步骤 5：（可选）验证数据

用户可以在Nebula Graph客户端（例如Nebula Graph Studio）中执行查询语句，确认数据是否已导入。例如：

```
GO FROM "player100" OVER follow;
```

用户也可以使用命令 `SHOW STATS` 查看统计数据。

步骤 6：（如有）在NEBULA GRAPH中重建索引

导入数据后，用户可以在Nebula Graph中重新创建并重建索引。详情请参见索引介绍。

最后更新: 2021年5月26日

2021 Vesoft Inc.

16.4.4 导入Parquet文件数据

本文以一个示例说明如何使用Exchange将存储在HDFS上的Parquet文件数据导入Nebula Graph。

如果要向Nebula Graph导入本地Parquet文件，请参见Nebula Importer。

数据集

本文以basketballplayer数据集为例。

环境配置

本文示例在MacOS下完成，以下是相关的环境配置信息：

- 硬件规格：
  - CPU：1.7 GHz Quad-Core Intel Core i7
  - 内存：16 GB
- Spark：2.4.7 单机版
- Hadoop：2.9.2 伪分布式部署
- Nebula Graph：2.0.0。使用Docker Compose部署。

前提条件

开始导入数据之前，用户需要确认以下信息：

- 已经安装部署Nebula Graph并获取如下信息：
  - Graph服务和Meta服务的的IP地址和端口。
  - 拥有Nebula Graph写权限的用户名和密码。
- 已经编译Exchange。详情请参见编译Exchange。本示例中使用Exchange 2.0。
- 已经安装Spark。
- 了解Nebula Graph中创建Schema的信息，包括标签和边类型的名称、属性等。
- 已经安装并开启Hadoop服务。

2021 Vesoft Inc.

操作步骤

步骤 1：在NEBULA GRAPH中创建SCHEMA

分析Parquet文件中的数据，按以下步骤在Nebula Graph中创建Schema：

1. 确认Schema要素。Nebula Graph中的Schema要素如下表所示。

要素	名称	属性
标签（Tag）	`player`	`name string, age int`
标签（Tag）	`team`	`name string`
边类型（Edge Type）	`follow`	`degree int`
边类型（Edge Type）	`serve`	`start_year int, end_year int`

2. 使用Nebula Console创建一个图空间**basketballplayer**，并创建一个Schema，如下所示。

```
## 创建图空间
nebula> CREATE SPACE basketballplayer \
        (partition_num = 10, \
        replica_factor = 1, \
        vid_type = FIXED_STRING(30));

## 选择图空间basketballplayer
nebula> USE basketballplayer;

## 创建标签player
nebula> CREATE TAG player(name string, age int);

## 创建标签team
nebula> CREATE TAG team(name string);

## 创建边类型follow
nebula> CREATE EDGE follow(degree int);

## 创建边类型serve
nebula> CREATE EDGE serve(start_year int, end_year int);
```

更多信息，请参见快速开始。

步骤 2：处理PARQUET文件

确认以下信息：

1. 处理Parquet文件以满足Schema的要求。
2. Parquet文件必须存储在HDFS中，并已获取文件存储路径。

步骤 3：修改配置文件

编译Exchange后，复制`target/classes/application.conf`文件设置Parquet数据源相关的配置。在本示例中，复制的文件名为`parquet_application.conf`。各个配置项的详细说明请参见配置说明。

```
{
  # Spark相关配置
  spark: {
    app: {
      name: Nebula Exchange 2.0
    }
    driver: {
      cores: 1
      maxResultSize: 1G
    }
    executor: {
        memory:1G
    }

    cores {
      max: 16
    }
  }

  # Nebula Graph相关配置
  nebula: {
    address:{
      # 指定Graph服务和所有Meta服务的IP地址和端口。
      # 如果有多台服务器，地址之间用英文逗号（,）分隔。
      # 格式: "ip1:port","ip2:port","ip3:port"
```


2021 Vesoft Inc.

```
    graph:["127.0.0.1:9669"]
    meta:["127.0.0.1:9559"]
  }

  # 指定拥有Nebula Graph写权限的用户名和密码。
  user: root
  pswd: nebula

  # 指定图空间名称。
  space: basketballplayer
  connection {
    timeout: 3000
    retry: 3
  }
  execution {
    retry: 3
  }
  error: {
    max: 32
    output: /tmp/errors
  }
  rate: {
    limit: 1024
    timeout: 1000
  }
}

# 处理点
tags: [
  # 设置标签player相关信息。
  {
    # 指定Nebula Graph中定义的标签名称。
    name: player
    type: {
      # 指定数据源，使用Parquet。
      source: parquet

      # 指定如何将点数据导入Nebula Graph：Client或SST。
      sink: client
    }

    # 指定Parquet文件的HDFS路径。
    # 用双引号括起路径，以hdfs://开头。
    path: "hdfs://192.168.11.139000/data/vertex_player.parquet"

    # 在fields里指定Parquet文件中key名称，其对应的value会作为Nebula Graph中指定属性的数据源。
    # 如果需要指定多个值，用英文逗号（,）隔开。
    fields: [age,name]

    # 指定Nebula Graph中定义的属性名称。
    # fields与nebula.fields的顺序必须一一对应。
    nebula.fields: [age, name]

    # 指定一个列作为VID的源。
    # vertex的值必须与Parquet文件中的字段保持一致。
    # 目前，Nebula Graph 2.0.0仅支持字符串或整数类型的VID。
    # 不要使用vertex.policy映射。
    vertex: {
      field:id
    }

    # 指定单批次写入Nebula Graph的最大点数量。
    batch: 256

    # 指定Spark分片数量。
    partition: 32
  }

  # 设置标签team相关信息。
  {
    # 指定Nebula Graph中定义的标签名称。
    name: team
    type: {
      # 指定数据源，使用Parquet。
      source: parquet

      # 指定如何将点数据导入Nebula Graph：Client或SST。
      sink: client
    }

    # 指定Parquet文件的HDFS路径。
    # 用双引号括起路径，以hdfs://开头。
    path: "hdfs://192.168.11.13:9000/data/vertex_team.parquet"

    # 在fields里指定Parquet文件中key名称，其对应的value会作为Nebula Graph中指定属性的数据源。
    # 如果需要指定多个值，用英文逗号（,）隔开。
    fields: [name]

    # 指定Nebula Graph中定义的属性名称。
    # fields与nebula.fields的顺序必须一一对应。
    nebula.fields: [name]

    # 指定一个列作为VID的源。
    # vertex的值必须与Parquet文件中的字段保持一致。
```


2021 Vesoft Inc.

```
    # 目前，Nebula Graph 2.0.0仅支持字符串或整数类型的VID。
    # 不要使用vertex.policy映射。
    vertex: {
      field:id
    }


    # 指定单批次写入Nebula Graph的最大点数量。
    batch: 256

    # 指定Spark分片数量。
    partition: 32
  }


  # 如果需要添加更多点，请参考前面的配置进行添加。
]
# 处理边
edges: [
  # 设置边类型follow相关信息。
  {
    # 指定Nebula Graph中定义的边类型名称。
    name: follow
    type: {
      # 指定数据源，使用Parquet。
      source: parquet

      # 指定如何将点数据导入Nebula Graph：Client或SST。
      sink: client
    }

    # 指定Parquet文件的HDFS路径。
    # 用双引号括起路径，以hdfs://开头。
    path: "hdfs://192.168.11.13:9000/data/edge_follow.parquet"

    # 在fields里指定Parquet文件中key名称，其对应的value会作为Nebula Graph中指定属性的数据源。
    # 如果需要指定多个值，用英文逗号（,）隔开。
    fields: [degree]

    # 指定Nebula Graph中定义的属性名称。
    # fields与nebula.fields的顺序必须一一对应。
    nebula.fields: [degree]

    # 指定一个列作为起始点和目的点的源。
    # vertex的值必须与Parquet文件中的字段保持一致。
    # 目前，Nebula Graph 2.0.0仅支持字符串或整数类型的VID。
    # 不要使用vertex.policy映射。
    source: {
      field: src
    }
    target: {
      field: dst
    }


    # 指定一个列作为rank的源(可选)。
    #ranking: rank

    # 指定单批次写入Nebula Graph的最大边数量。
    batch: 256

    # 指定Spark分片数量。
    partition: 32
  }

  # 设置边类型serve相关信息。
  {
    # 指定Nebula Graph中定义的边类型名称。
    name: serve
    type: {
      # 指定数据源，使用Parquet。
      source: parquet

      # 指定如何将点数据导入Nebula Graph：Client或SST。
      sink: client
    }

    # 指定Parquet文件的HDFS路径。
    # 用双引号括起路径，以hdfs://开头。
    path: "hdfs://192.168.11.13:9000/data/edge_serve.parquet"

    # 在fields里指定Parquet文件中key名称，其对应的value会作为Nebula Graph中指定属性的数据源。
    # 如果需要指定多个值，用英文逗号（,）隔开。
    fields: [start_year,end_year]

    # 指定Nebula Graph中定义的属性名称。
    # fields与nebula.fields的顺序必须一一对应。
    nebula.fields: [start_year, end_year]

    # 指定一个列作为起始点和目的点的源。
    # vertex的值必须与Parquet文件中的字段保持一致。
    # 目前，Nebula Graph 2.0.0仅支持字符串或整数类型的VID。
    # 不要使用vertex.policy映射。
    source: {
```


2021 Vesoft Inc.

```
        field: src
      }
      target: {
        field: dst
      }


      # 指定一个列作为rank的源(可选)。
      #ranking: _c5


      # 指定单批次写入Nebula Graph的最大边数量。
      batch: 256

      # 指定Spark分片数量。
      partition: 32
    }

  ]
  # 如果需要添加更多边，请参考前面的配置进行添加。
}
```

步骤 4：向NEBULA GRAPH导入数据

运行如下命令将Parquet文件数据导入到Nebula Graph中。关于参数的说明，请参见导入命令参数。

```
${SPARK_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange <nebula-exchange-2.0.0.jar_path> -c <parquet_application.conf_path>
```

> **Note**
>
> JAR包有两种获取方式：自行编译或者从maven仓库下载。

示例：

```
${SPARK_HOME}/bin/spark-submit  --master "local" --class com.vesoft.nebula.exchange.Exchange  /root/nebula-spark-utils/nebula-exchange/target/nebula-exchange-2.0.0.jar  -c /root/nebula-spark-utils/nebula-exchange/target/classes/parquet_application.conf
```

用户可以在返回信息中搜索 `batchSuccess.<tag_name/edge_name>`，确认成功的数量。例如例如 `batchSuccess.follow: 300` 。

步骤 5：（可选）验证数据

用户可以在Nebula Graph客户端（例如Nebula Graph Studio）中执行查询语句，确认数据是否已导入。例如：

```
GO FROM "player100" OVER follow;
```

用户也可以使用命令 `SHOW STATS` 查看统计数据。

步骤 6：（如有）在NEBULA GRAPH中重建索引

导入数据后，用户可以在Nebula Graph中重新创建并重建索引。详情请参见索引介绍。

最后更新: 2021年5月26日

2021 Vesoft Inc.

16.4.5 导入HBase数据

本文以一个示例说明如何使用Exchange将存储在HBase上的数据导入Nebula Graph。

数据集

本文以basketballplayer数据集为例。

在本示例中，该数据集已经存入HBase中，以 `player` 、 `team` 、 `follow` 和 `serve` 四个表存储了所有点和边的信息。以下为各个表的部分数据。

```
hbase(main):002:0> scan "player"
ROW                                    COLUMN+CELL
 player100                             column=cf:age, timestamp=1618881347530, value=42
 player100                             column=cf:name, timestamp=1618881354604, value=Tim Duncan
 player101                             column=cf:age, timestamp=1618881369124, value=36
 player101                             column=cf:name, timestamp=1618881379102, value=Tony Parker
 player102                             column=cf:age, timestamp=1618881386987, value=33
 player102                             column=cf:name, timestamp=1618881393370, value=LaMarcus Aldridge
 player103                             column=cf:age, timestamp=1618881402002, value=32
 player103                             column=cf:name, timestamp=1618881407882, value=Rudy Gay
 ...

hbase(main):003:0> scan "team"
ROW                                    COLUMN+CELL
 team200                               column=cf:name, timestamp=1618881445563, value=Warriors
 team201                               column=cf:name, timestamp=1618881453636, value=Nuggets
 ...

hbase(main):004:0> scan "follow"
ROW                                    COLUMN+CELL
 player100                             column=cf:degree, timestamp=1618881804853, value=95
 player100                             column=cf:dst_player, timestamp=1618881791522, value=player101
 player101                             column=cf:degree, timestamp=1618881824685, value=90
 player101                             column=cf:dst_player, timestamp=1618881816042, value=player102
 ...

hbase(main):005:0> scan "serve"
ROW                                    COLUMN+CELL
 player100                             column=cf:end_year, timestamp=1618881899333, value=2016
 player100                             column=cf:start_year, timestamp=1618881890117, value=1997
 player100                             column=cf:teamid, timestamp=1618881875739, value=team204
 ...
```

环境配置

本文示例在MacOS下完成，以下是相关的环境配置信息：

- 硬件规格：
  - CPU：1.7 GHz Quad-Core Intel Core i7
  - 内存：16 GB
- Spark：2.4.7，单机版
- Hadoop：2.9.2，伪分布式部署
- HBase：2.2.7
- Nebula Graph：2.0.0。使用Docker Compose部署。

2021 Vesoft Inc.

前提条件

开始导入数据之前，用户需要确认以下信息：

- 已经安装部署Nebula Graph并获取如下信息：
  - Graph服务和Meta服务的的IP地址和端口。
  - 拥有Nebula Graph写权限的用户名和密码。
- 已经编译Exchange。详情请参见编译Exchange。本示例中使用Exchange 2.0。
- 已经安装Spark。
- 了解Nebula Graph中创建Schema的信息，包括标签和边类型的名称、属性等。
- 已经安装并开启Hadoop服务。

操作步骤

步骤 1：在NEBULA GRAPH中创建SCHEMA

分析数据，按以下步骤在Nebula Graph中创建Schema：

1. 确认Schema要素。Nebula Graph中的Schema要素如下表所示。

要素	名称	属性
标签（Tag）	`player`	`name string, age int`
标签（Tag）	`team`	`name string`
边类型（Edge Type）	`follow`	`degree int`
边类型（Edge Type）	`serve`	`start_year int, end_year int`

2. 在Nebula Graph中创建一个图空间**basketballplayer**，并创建一个Schema，如下所示。

```
## 创建图空间
nebula> CREATE SPACE basketballplayer \
        (partition_num = 10, \
        replica_factor = 1, \
        vid_type = FIXED_STRING(30));

## 选择图空间basketballplayer
nebula> USE basketballplayer;

## 创建标签player
nebula> CREATE TAG player(name string, age int);

## 创建标签team
nebula> CREATE TAG team(name string);

## 创建边类型follow
nebula> CREATE EDGE follow(degree int);

## 创建边类型serve
nebula> CREATE EDGE serve(start_year int, end_year int);
```

更多信息，请参见快速开始。

步骤 2：修改配置文件

编译Exchange后，复制 `target/classes/application.conf` 文件设置HBase数据源相关的配置。在本示例中，复制的文件名为 `hbase_application.conf` 。各个配置项的详细说明请参见配置说明。

```
{
  # Spark相关配置
  spark: {
    app: {
      name: Nebula Exchange 2.0
    }
    driver: {
      cores: 1
      maxResultSize: 1G
```


2021 Vesoft Inc.

```
  }
  cores {
    max: 16
  }
}


# Nebula Graph相关配置
nebula: {
  address:{
    # 以下为Nebula Graph的Graph服务和Meta服务所在机器的IP地址及端口。
    # 如果有多个地址，格式为 "ip1:port","ip2:port","ip3:port"。
    # 不同地址之间以英文逗号 (,) 隔开。
    graph:["127.0.0.1:9669"]
    meta:["127.0.0.1:9559"]
  }
  # 填写的账号必须拥有Nebula Graph相应图空间的写数据权限。
  user: root
  pswd: nebula
  # 填写Nebula Graph中需要写入数据的图空间名称。
  space: basketballplayer
  connection {
    timeout: 3000
    retry: 3
  }
  execution {
    retry: 3
  }
  error: {
    max: 32
    output: /tmp/errors
  }
  rate: {
    limit: 1024
    timeout: 1000
  }
}
# 处理点
tags: [
  # 设置标签player相关信息。
  # 如果需要将rowkey设置为数据源，请填写“rowkey”,列族内的列请填写实际列名。
  {
    # Nebula Graph中对应的标签名称。
    name: player
    type: {
      # 指定数据源文件格式，设置为HBase。
      source: hbase
      # 指定如何将点数据导入Nebula Graph：Client或SST。
      sink: client
    }
    host:192.168.11.13
    port:2181
    table:"player"
    columnFamily:"cf"

    # 在fields里指定player表中的列名称，其对应的value会作为Nebula Graph中指定属性。
    # fields和nebula.fields里的配置必须一一对应。
    # 如果需要指定多个列名称，用英文逗号 (,) 隔开。
    fields: [age,name]
    nebula.fields: [age,name]

    # 指定表中某一列数据为Nebula Graph中点VID的来源。
    # 例如rowkey作为VID的来源，请填写“rowkey”。
    vertex:{
        field:rowkey
    }


    # 单次写入 Nebula Graph 的最大点数据量。
    batch: 256

    # Spark 分区数量
    partition: 32
  }
  # 设置标签team相关信息。
  {
    name: team
    type: {
      source: hbase
      sink: client
    }
    host:192.168.11.13
    port:2181
    table:"team"
    columnFamily:"cf"
    fields: [name]
    nebula.fields: [name]
    vertex:{
        field:rowkey
    }
    batch: 256
    partition: 32
  }
```


2021 Vesoft Inc.

```
  ]

  # 处理边数据
  edges: [
    # 设置边类型follow相关信息
    {
      # Nebula Graph中对应的边类型名称。
      name: follow

      type: {
        # 指定数据源文件格式，设置为HBase。
        source: hbase

        # 指定边数据导入Nebula Graph的方式，
        # 指定如何将点数据导入Nebula Graph：Client或SST。
        sink: client
      }

      host:192.168.11.13
      port:2181
      table:"follow"
      columnFamily:"cf"

      # 在fields里指定follow表中的列名称，其对应的value会作为Nebula Graph中指定属性。
      # fields和nebula.fields里的配置必须一一对应。
      # 如果需要指定多个列名称，用英文逗号（,）隔开。
      fields: [degree]
      nebula.fields: [degree]

      # 在source里，将follow表中某一列作为边的起始点数据源。示例使用rowkey。
      # 在target里，将follow表中某一列作为边的目的点数据源。示例使用列dst_player。
      source:{
          field:rowkey
      }


      target:{
          field:dst_player
      }


      # 单次写入 Nebula Graph 的最大点数据量。
      batch: 256

      # Spark 分区数量
      partition: 32
    }

    # 设置边类型serve相关信息
    {
      name: serve
      type: {
        source: hbase
        sink: client
      }
      host:192.168.11.13
      port:2181
      table:"serve"
      columnFamily:"cf"

      fields: [start_year,end_year]
      nebula.fields: [start_year,end_year]
      source:{
          field:rowkey
      }

      target:{
          field:teamid
      }

      batch: 256
      partition: 32
    }
  ]
}
```

步骤 3：向NEBULA GRAPH导入数据

运行如下命令将HBase数据导入到Nebula Graph中。关于参数的说明，请参见导入命令参数。

```
${SPARK_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange <nebula-exchange-2.0.0.jar_path> -c <hbase_application.conf_path>
```

> **Note**
>
> JAR包有两种获取方式：自行编译或者从maven仓库下载。

2021 Vesoft Inc.

示例：

```
${SPARK_HOME}/bin/spark-submit  --master "local" --class com.vesoft.nebula.exchange.Exchange  /root/nebula-spark-utils/nebula-exchange/target/nebula-exchange-2.0.0.jar  -c /root/nebula-spark-utils/nebula-exchange/target/classes/hbase_application.conf
```

用户可以在返回信息中搜索 `batchSuccess.<tag_name/edge_name>`，确认成功的数量。例如例如 `batchSuccess.follow: 300` 。

步骤 4：（可选）验证数据

用户可以在Nebula Graph客户端（例如Nebula Graph Studio）中执行查询语句，确认数据是否已导入。例如：

```
GO FROM "player100" OVER follow;
```

用户也可以使用命令 `SHOW STATS` 查看统计数据。

步骤 5：（如有）在NEBULA GRAPH中重建索引

导入数据后，用户可以在Nebula Graph中重新创建并重建索引。详情请参见索引介绍。

最后更新: 2021年5月26日

2021 Vesoft Inc.

16.4.6 导入MySQL数据

本文以一个示例说明如何使用Exchange将存储在MySQL上的数据导入Nebula Graph。

数据集

本文以basketballplayer数据集为例。

在本示例中，该数据集已经存入MySQL中名为 `basketball` 的数据库中，以 `player` 、 `team` 、 `follow` 和 `serve` 四个表存储了所有点和边的信息。以下为各个表的结构。

```
mysql> desc player;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| playerid | varchar(30) | YES  |     | NULL    |       |
| age      | int         | YES  |     | NULL    |       |
| name     | varchar(30) | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+

mysql> desc team;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| teamid | varchar(30) | YES  |     | NULL    |       |
| name   | varchar(30) | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+

mysql> desc follow;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| src_player | varchar(30) | YES  |     | NULL    |       |
| dst_player | varchar(30) | YES  |     | NULL    |       |
| degree     | int         | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+

mysql> desc serve;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| playerid   | varchar(30) | YES  |     | NULL    |       |
| teamid     | varchar(30) | YES  |     | NULL    |       |
| start_year | int         | YES  |     | NULL    |       |
| end_year   | int         | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
```

环境配置

本文示例在MacOS下完成，以下是相关的环境配置信息：

- 硬件规格：
  - CPU：1.7 GHz Quad-Core Intel Core i7
  - 内存：16 GB
- Spark：2.4.7，单机版
- Hadoop：2.9.2，伪分布式部署
- MySQL： 8.0.23
- Nebula Graph：2.0.0。使用Docker Compose部署。

2021 Vesoft Inc.

前提条件

开始导入数据之前，用户需要确认以下信息：

- 已经安装部署Nebula Graph并获取如下信息：
  - Graph服务和Meta服务的的IP地址和端口。
  - 拥有Nebula Graph写权限的用户名和密码。
- 已经编译Exchange。详情请参见编译Exchange。本示例中使用Exchange 2.0。
- 已经安装Spark。
- 了解Nebula Graph中创建Schema的信息，包括标签和边类型的名称、属性等。
- 已经安装并开启Hadoop服务。

操作步骤

步骤 1：在NEBULA GRAPH中创建SCHEMA

分析数据，按以下步骤在Nebula Graph中创建Schema：

1. 确认Schema要素。Nebula Graph中的Schema要素如下表所示。

要素	名称	属性
标签（Tag）	`player`	`name string, age int`
标签（Tag）	`team`	`name string`
边类型（Edge Type）	`follow`	`degree int`
边类型（Edge Type）	`serve`	`start_year int, end_year int`

2. 在Nebula Graph中创建一个图空间**basketballplayer**，并创建一个Schema，如下所示。

```
## 创建图空间
nebula> CREATE SPACE basketballplayer \
        (partition_num = 10, \
        replica_factor = 1, \
        vid_type = FIXED_STRING(30));

## 选择图空间basketballplayer
nebula> USE basketballplayer;

## 创建标签player
nebula> CREATE TAG player(name string, age int);

## 创建标签team
nebula> CREATE TAG team(name string);

## 创建边类型follow
nebula> CREATE EDGE follow(degree int);

## 创建边类型serve
nebula> CREATE EDGE serve(start_year int, end_year int);
```

更多信息，请参见快速开始。

步骤 2：修改配置文件

编译Exchange后，复制 `target/classes/application.conf` 文件设置MySQL数据源相关的配置。在本示例中，复制的文件名为 `mysql_application.conf` 。各个配置项的详细说明请参见配置说明。

```
{
  # Spark相关配置
  spark: {
    app: {
      name: Nebula Exchange 2.0
    }
    driver: {
      cores: 1
      maxResultSize: 1G
```


2021 Vesoft Inc.

```
    }
    cores {
      max: 16
    }
  }

# Nebula Graph相关配置
  nebula: {
    address:{
      # 以下为Nebula Graph的Graph服务和Meta服务所在机器的IP地址及端口。
      # 如果有多个地址，格式为 "ip1:port","ip2:port","ip3:port"。
      # 不同地址之间以英文逗号 (,) 隔开。
      graph:["127.0.0.1:9669"]
      meta:["127.0.0.1:9559"]
    }
    # 填写的账号必须拥有Nebula Graph相应图空间的写数据权限。
    user: root
    pswd: nebula
    # 填写Nebula Graph中需要写入数据的图空间名称。
    space: basketballplayer
    connection {
      timeout: 3000
      retry: 3
    }
    execution {
      retry: 3
    }
    error: {
      max: 32
      output: /tmp/errors
    }
    rate: {
      limit: 1024
      timeout: 1000
    }
  }
  # 处理点
  tags: [
    # 设置标签player相关信息。
    {
      # Nebula Graph中对应的标签名称。
      name: player
      type: {
        # 指定数据源文件格式，设置为MySQL。
        source: mysql
        # 指定如何将点数据导入Nebula Graph：Client或SST。
        sink: client
      }

      host:192.168.11.13
      port:3306
      database:"basketball"
      table:"player"
      user:"test"
      password:"123456"
      sentence:"select playerid, age, name from basketball.player order by playerid;"

      # 在fields里指定player表中的列名称，其对应的value会作为Nebula Graph中指定属性。
      # fields和nebula.fields里的配置必须一一对应。
      # 如果需要指定多个列名称，用英文逗号 (,) 隔开。
      fields: [age,name]
      nebula.fields: [age,name]

      # 指定表中某一列数据为Nebula Graph中点VID的来源。
      # vertex.field的值必须与上述fields中的列名保持一致。
      vertex: {
        field:playerid
      }

      # 单次写入 Nebula Graph 的最大点数据量。
      batch: 256

      # Spark 分区数量
      partition: 32
    }
    # 设置标签team相关信息。
    {
      name: team
      type: {
        source: mysql
        sink: client
      }

      host:192.168.11.13
      port:3306
      database:"basketball"
      table:"team"
      user:"test"
      password:"123456"
      sentence:"select teamid, name from basketball.team order by teamid;"

      fields: [name]
      nebula.fields: [name]
      vertex: {
```


2021 Vesoft Inc.

```
        field: teamid
      }
      batch: 256
      partition: 32
    }

  ]

  # 处理边数据
  edges: [
    # 设置边类型follow相关信息
    {
      # Nebula Graph中对应的边类型名称。
      name: follow

      type: {
        # 指定数据源文件格式，设置为MySQL。
        source: mysql

        # 指定边数据导入Nebula Graph的方式，
        # 指定如何将点数据导入Nebula Graph：Client或SST。
        sink: client
      }

      host:192.168.11.13
      port:3306
      database:"basketball"
      table:"follow"
      user:"test"
      password:"123456"
      sentence:"select src_player,dst_player,degree from basketball.follow order by src_player;"

      # 在fields里指定follow表中的列名称，其对应的value会作为Nebula Graph中指定属性。
      # fields和nebula.fields里的配置必须一一对应。
      # 如果需要指定多个列名称，用英文逗号（,）隔开。
      fields: [degree]
      nebula.fields: [degree]

      # 在source里，将follow表中某一列作为边的起始点数据源。
      # 在target里，将follow表中某一列作为边的目的点数据源。
      source: {
        field: src_player
      }

      target: {
        field: dst_player
      }

      # 单次写入 Nebula Graph 的最大点数据量。
      batch: 256

      # Spark 分区数量
      partition: 32
    }

    # 设置边类型serve相关信息
    {
      name: serve
      type: {
        source: mysql
        sink: client
      }

      host:192.168.11.13
      port:3306
      database:"basketball"
      table:"serve"
      user:"test"
      password:"123456"
      sentence:"select playerid,teamid,start_year,end_year from basketball.serve order by playerid;"
      fields: [start_year,end_year]
      nebula.fields: [start_year,end_year]
      source: {
        field: playerid
      }
      target: {
        field: teamid
      }
      batch: 256
      partition: 32
    }
  ]
}
```

步骤 3：向NEBULA GRAPH导入数据

运行如下命令将MySQL数据导入到Nebula Graph中。关于参数的说明，请参见导入命令参数。

```
${SPARK_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange <nebula-exchange-2.0.0.jar_path> -c <mysql_application.conf_path>
```


2021 Vesoft Inc.

> **Note**
>
> JAR包有两种获取方式：自行编译或者从maven仓库下载。

示例：

```
${SPARK_HOME}/bin/spark-submit  --master "local" --class com.vesoft.nebula.exchange.Exchange  /root/nebula-spark-utils/nebula-exchange/target/nebula-exchange-2.0.0.jar  -c /root/nebula-spark-utils/nebula-exchange/target/classes/mysql_application.conf
```

用户可以在返回信息中搜索 `batchSuccess.<tag_name/edge_name>`，确认成功的数量。例如例如 `batchSuccess.follow: 300` 。

步骤 4：（可选）验证数据

用户可以在Nebula Graph客户端（例如Nebula Graph Studio）中执行查询语句，确认数据是否已导入。例如：

```
GO FROM "player100" OVER follow;
```

用户也可以使用命令 `SHOW STATS` 查看统计数据。

步骤 5：（如有）在NEBULA GRAPH中重建索引

导入数据后，用户可以在Nebula Graph中重新创建并重建索引。详情请参见索引介绍。

最后更新: 2021年5月26日

2021 Vesoft Inc.

16.4.7 导入Neo4j数据

本文以一个示例说明如何使用Exchange将存储在Neo4j的数据导入Nebula Graph。

实现方法

Exchange使用**Neo4j Driver 4.0.1**实现对Neo4j数据的读取。执行批量导出之前，用户需要在配置文件中写入针对标签（label）和关系类型（Relationship Type）自动执行的Cypher语句，以及Spark分区数，提高数据导出性能。

Exchange读取Neo4j数据时需要完成以下工作：

1. Exchange中的Reader会将配置文件中 `exec` 部分的Cypher `RETURN` 语句后面的语句替换为 `COUNT(*)`，并执行这个语句，从而获取数据总量，再根据Spark分区数量计算每个分区的起始偏移量和大小。
2. （可选）如果用户配置了 `check_point_path` 目录，Reader会读取目录中的文件。如果处于续传状态，Reader会计算每个Spark分区应该有的偏移量和大小。
3. 在每个Spark分区里，Exchange中的Reader会在Cypher语句后面添加不同的 `SKIP` 和 `LIMIT` 语句，调用Neo4j Driver并行执行，将数据分布到不同的Spark分区中。
4. Reader最后将返回的数据处理成DataFrame。

至此，Exchange即完成了对Neo4j数据的导出。之后，数据被并行写入Nebula Graph数据库中。

整个过程如下图所示。

数据集

本文以basketballplayer数据集为例。

2021 Vesoft Inc.

环境配置

本文示例在MacOS下完成，以下是相关的环境配置信息：

- 硬件规格：
  - CPU：Intel(R) Xeon(R) CPU E5-2697 v3 @ 2.60GHz
  - CPU 内核数：14
  - 内存：251 GB
- Spark：单机版，2.4.6 pre-build for Hadoop 2.7
- Neo4j：3.5.20 Community Edition
- Nebula Graph：2.0.0。使用Docker Compose部署。

前提条件

开始导入数据之前，用户需要确认以下信息：

- 已经安装部署Nebula Graph并获取如下信息：
  - Graph服务和Meta服务的的IP地址和端口。
  - 拥有Nebula Graph写权限的用户名和密码。
- 已经编译Exchange。详情请参见编译Exchange。本示例中使用Exchange 2.0。
- 已经安装Spark。
- 了解Nebula Graph中创建Schema的信息，包括标签和边类型的名称、属性等。

2021 Vesoft Inc.

操作步骤

步骤 1：在NEBULA GRAPH中创建SCHEMA

分析数据，按以下步骤在Nebula Graph中创建Schema：

1. 确认Schema要素。Nebula Graph中的Schema要素如下表所示。

要素	名称	属性
标签（Tag）	`player`	`name string, age int`
标签（Tag）	`team`	`name string`
边类型（Edge Type）	`follow`	`degree int`
边类型（Edge Type）	`serve`	`start_year int, end_year int`

2. 使用Nebula Console创建一个图空间**basketballplayer**，并创建一个Schema，如下所示。

```
## 创建图空间
nebula> CREATE SPACE basketballplayer \
        (partition_num = 10, \
        replica_factor = 1, \
        vid_type = FIXED_STRING(30));

## 选择图空间basketballplayer
nebula> USE basketballplayer;

## 创建标签player
nebula> CREATE TAG player(name string, age int);

## 创建标签team
nebula> CREATE TAG team(name string);

## 创建边类型follow
nebula> CREATE EDGE follow(degree int);

## 创建边类型serve
nebula> CREATE EDGE serve(start_year int, end_year int);
```

更多信息，请参见快速开始。

步骤 2：配置源数据

为了提高Neo4j数据的导出速度，在Neo4j数据库中为相应属性创建索引。详细信息，参考Neo4j用户手册。

步骤 3：修改配置文件

编译Exchange后，复制 `target/classes/application.conf` 文件设置数据源相关的配置。在本示例中，复制的文件名为 `neo4j_application.conf` 。各个配置项的详细说明请参见配置说明。

```
{
  # Spark相关配置
  spark: {
    app: {
      name: Nebula Exchange 2.0
    }

    driver: {
      cores: 1
      maxResultSize: 1G
    }

    executor: {
        memory:1G
    }

    cores:{
      max: 16
    }
  }


  # Nebula Graph相关配置
  nebula: {
    address:{
      graph:["127.0.0.1:9669"]
      meta:["127.0.0.1:9559"]
    }
    user: root
    pswd: nebula
```


2021 Vesoft Inc.

```
  space: basketballplayer

  connection {
    timeout: 3000
    retry: 3
  }

  execution {
    retry: 3
  }

  error: {
    max: 32
    output: /tmp/errors
  }

  rate: {
    limit: 1024
    timeout: 1000
  }
}

# 处理点
tags: [


  # 设置标签player相关信息。
  {
    name: player
    type: {
      source: neo4j
      sink: client
    }
    server: "bolt://192.168.11.13:7687"
    user: neo4j
    password:neo4j
    database:neo4j
    exec: "match (n:player) return n.id as id, n.age as age, n.name as name"
    fields: [age,name]
    nebula.fields: [age,name]
    vertex: {
      field:id
    }
    partition: 10
    batch: 1000
    check_point_path: /tmp/test
 }
# 设置标签team相关信息。
{
    name: team
    type: {
      source: neo4j
      sink: client
    }
    server: "bolt://192.168.11.13:7687"
    user: neo4j
    password:neo4j
    database:neo4j
    exec: "match (n:team) return n.id as id,n.name as name"
    fields: [name]
    nebula.fields: [name]
    vertex: {
      field:id
    }
    partition: 10
    batch: 1000
    check_point_path: /tmp/test
 }
]

# 处理边数据
edges: [
  # 设置边类型follow相关信息
  {
    name: follow
    type: {
      source: neo4j
      sink: client
    }
    server: "bolt://192.168.11.13:7687"
    user: neo4j
    password:neo4j
    database:neo4j
    exec: "match (a:player)-[r:follow]->(b:player) return a.id as src, b.id as dst, r.degree as degree  order by id(r)"
    fields: [degree]
    nebula.fields: [degree]
    source: {
      field: src
    }
    target: {
      field: dst
    }
    partition: 10
    batch: 1000
```


2021 Vesoft Inc.

```
      check_point_path: /tmp/test
    }
    # 设置边类型serve相关信息
    {
      name: serve
      type: {
        source: neo4j
        sink: client
      }
      server: "bolt://192.168.11.13:7687"
      user: neo4j
      password:neo4j
      database:neo4j
      exec: "match (a:player)-[r:serve]->(b:team) return a.id as src, b.id as dst, r.start_year as start_year, r.end_year as end_year  order by id(r)"
      fields: [start_year,end_year]
      nebula.fields: [start_year,end_year]
      source: {
        field: src
      }
      target: {
        field: dst
      }
      partition: 10
      batch: 1000
      check_point_path: /tmp/test
    }
  ]
}
```

exec配置说明

在配置 `tags.exec` 或者 `edges.exec` 参数时，需要填写Cypher查询语句。为了保证每次查询结果排序一致，并且为了防止在导入时丢失数据，强烈建议在Cypher查询语句中加入 `ORDER BY` 子句，同时，为了提高数据导入效率，最好选取有索引的属性作为排序的属性。如果没有索引，用户也可以观察默认的排序，选择合适的属性用于排序，以提高效率。如果默认的排序找不到规律，用户可以根据点或关系的ID进行排序，并且将 `partition` 设置为一个尽量小的值，减轻Neo4j的排序压力。

> **说明**：使用 `ORDER BY` 子句会延长数据导入的时间。

另外，Exchange需要在不同Spark分区执行不同 `SKIP` 和 `LIMIT` 的Cypher语句，所以在 `tags.exec` 和 `edges.exec` 对应的Cypher语句中不能含有 `SKIP` 和 `LIMIT` 子句。

tags.vertex或edges.vertex配置说明

Nebula Graph在创建点和边时会将ID作为唯一主键，如果主键已存在则会覆盖该主键中的数据。所以，假如将某个Neo4j属性值作为Nebula Graph的ID，而这个属性值在Neo4j中是有重复的，就会导致重复ID，它们对应的数据有且只有一条会存入Nebula Graph中，其它的则会被覆盖掉。由于数据导入过程是并发地往Nebula Graph中写数据，最终保存的数据并不能保证是Neo4j中最新的数据。

check_point_path配置说明

如果启用了断点续传功能，为避免数据丢失，在断点和续传之间，数据库不应该改变状态，例如不能添加数据或删除数据，同时，不能更改 `partition` 数量配置。

步骤 4：向NEBULA GRAPH导入数据

运行如下命令将文件数据导入到Nebula Graph中。关于参数的说明，请参见导入命令参数。

```
${SPARK_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange <nebula-exchange-2.0.0.jar_path> -c <neo4j_application.conf_path>
```

> **Note**
>
> JAR包有两种获取方式：自行编译或者从maven仓库下载。

示例：

```
${SPARK_HOME}/bin/spark-submit  --master "local" --class com.vesoft.nebula.exchange.Exchange  /root/nebula-spark-utils/nebula-exchange/target/nebula-exchange-2.0.0.jar  -c /root/nebula-spark-utils/nebula-exchange/target/classes/neo4j_application.conf
```

用户可以在返回信息中搜索 `batchSuccess.<tag_name/edge_name>`，确认成功的数量。例如例如 `batchSuccess.follow: 300` 。

2021 Vesoft Inc.

步骤 5：（可选）验证数据

用户可以在Nebula Graph客户端（例如Nebula Graph Studio）中执行查询语句，确认数据是否已导入。例如：

```
GO FROM "player100" OVER follow;
```

用户也可以使用命令 `SHOW STATS` 查看统计数据。

步骤 6：（如有）在NEBULA GRAPH中重建索引

导入数据后，用户可以在Nebula Graph中重新创建并重建索引。详情请参见索引介绍。

最后更新: 2021年5月26日

2021 Vesoft Inc.

16.4.8 导入Hive数据

本文以一个示例说明如何使用Exchange将存储在Hive上的数据导入Nebula Graph。

数据集

本文以basketballplayer数据集为例。

在本示例中，该数据集已经存入Hive中名为 `basketball` 的数据库中，以 `player` 、 `team` 、 `follow` 和 `serve` 四个表存储了所有点和边的信息。以下为各个表的结构。

```
scala> spark.sql("describe basketball.player").show
+--------+---------+-------+
|col_name|data_type|comment|
+--------+---------+-------+
|playerid|   string|   null|
|     age|   bigint|   null|
|    name|   string|   null|
+--------+---------+-------+

scala> spark.sql("describe basketball.team").show
+----------+---------+-------+
|  col_name|data_type|comment|
+----------+---------+-------+
|    teamid|   string|   null|
|      name|   string|   null|
+----------+---------+-------+

scala> spark.sql("describe basketball.follow").show
+----------+---------+-------+
|  col_name|data_type|comment|
+----------+---------+-------+
|src_player|   string|   null|
|dst_player|   string|   null|
|    degree|   bigint|   null|
+----------+---------+-------+

scala> spark.sql("describe basketball.serve").show
+----------+---------+-------+
|  col_name|data_type|comment|
+----------+---------+-------+
|  playerid|   string|   null|
|    teamid|   string|   null|
|start_year|   bigint|   null|
|  end_year|   bigint|   null|
+----------+---------+-------+
```

说明：Hive的数据类型 `bigint` 与Nebula Graph的 `int` 对应。

环境配置

本文示例在MacOS下完成，以下是相关的环境配置信息：

- 硬件规格：
  - CPU：1.7 GHz Quad-Core Intel Core i7
  - 内存：16 GB
- Spark：2.4.7，单机版
- Hadoop：2.9.2，伪分布式部署
- Hive：2.3.7，Hive Metastore 数据库为 MySQL 8.0.22
- Nebula Graph：2.0.0。使用Docker Compose部署。

2021 Vesoft Inc.

前提条件

开始导入数据之前，用户需要确认以下信息：

- 已经安装部署Nebula Graph并获取如下信息：
  - Graph服务和Meta服务的的IP地址和端口。
  - 拥有Nebula Graph写权限的用户名和密码。
- 已经编译Exchange。详情请参见编译Exchange。本示例中使用Exchange 2.0。
- 已经安装Spark。
- 了解Nebula Graph中创建Schema的信息，包括标签和边类型的名称、属性等。
- 已经安装并开启Hadoop服务，并已启动Hive Metastore数据库（本示例中为 MySQL）。

操作步骤

步骤 1：在NEBULA GRAPH中创建SCHEMA

分析数据，按以下步骤在Nebula Graph中创建Schema：

1. 确认Schema要素。Nebula Graph中的Schema要素如下表所示。

要素	名称	属性
标签（Tag）	`player`	`name string, age int`
标签（Tag）	`team`	`name string`
边类型（Edge Type）	`follow`	`degree int`
边类型（Edge Type）	`serve`	`start_year int, end_year int`

2. 在Nebula Graph中创建一个图空间**basketballplayer**，并创建一个Schema，如下所示。

```
## 创建图空间
nebula> CREATE SPACE basketballplayer \
        (partition_num = 10, \
        replica_factor = 1, \
        vid_type = FIXED_STRING(30));

## 选择图空间basketballplayer
nebula> USE basketballplayer;

## 创建标签player
nebula> CREATE TAG player(name string, age int);

## 创建标签team
nebula> CREATE TAG team(name string);

## 创建边类型follow
nebula> CREATE EDGE follow(degree int);

## 创建边类型serve
nebula> CREATE EDGE serve(start_year int, end_year int);
```

更多信息，请参见快速开始。

步骤 2：使用SPARK SQL确认HIVE SQL语句

启动spark-shell环境后，依次运行以下语句，确认Spark能读取Hive中的数据。

```
scala> sql("select playerid, age, name from basketball.player").show
scala> sql("select teamid, name from basketball.team").show
scala> sql("select src_player, dst_player, degree from basketball.follow").show
scala> sql("select playerid, teamid, start_year, end_year from basketball.serve").show
```

以下为表 `basketball.player` 中读出的结果。

```
+---------+----+----------------+
| playerid| age|            name|
```


2021 Vesoft Inc.

```
+---------+----+----------------+
|player100|  42|       Tim Duncan|
|player101|  36|      Tony Parker|
|player102|  33|LaMarcus Aldridge|
|player103|  32|         Rudy Gay|
|player104|  32|  Marco Belinelli|
+---------+----+----------------+
...
```

步骤 3：修改配置文件

编译Exchange后，复制 `target/classes/application.conf` 文件设置Hive数据源相关的配置。在本示例中，复制的文件名为 `hive_application.conf` 。各个配置项的详细说明请参见配置说明。

```
{
  # Spark相关配置
  spark: {
    app: {
      name: Nebula Exchange 2.0
    }
    driver: {
      cores: 1
      maxResultSize: 1G
    }
    cores {
      max: 16
    }
  }

  # 如果Spark和Hive部署在不同集群，才需要配置连接Hive的参数，否则请忽略这些配置。
  #hive: {
  #  waredir: "hdfs://NAMENODE_IP:9000/apps/svr/hive-xxx/warehouse/"
  #  connectionURL: "jdbc:mysql://your_ip:3306/hive_spark?characterEncoding=UTF-8"
  #  connectionDriverName: "com.mysql.jdbc.Driver"
  #  connectionUserName: "user"
  #  connectionPassword: "password"
  #}

  # Nebula Graph相关配置
  nebula: {
    address:{
      # 以下为Nebula Graph的Graph服务和所有Meta服务所在机器的IP地址及端口。
      # 如果有多个地址，格式为 "ip1:port","ip2:port","ip3:port"。
      # 不同地址之间以英文逗号 (,) 隔开。
      graph:["127.0.0.1:9669"]
      meta:["127.0.0.1:9559"]
    }
    # 填写的账号必须拥有Nebula Graph相应图空间的写数据权限。
    user: root
    pswd: nebula
    # 填写Nebula Graph中需要写入数据的图空间名称。
    space: basketballplayer
    connection {
      timeout: 3000
      retry: 3
    }
    execution {
      retry: 3
    }
    error: {
      max: 32
      output: /tmp/errors
    }
    rate: {
      limit: 1024
      timeout: 1000
    }
  }
  # 处理点
  tags: [
    # 设置标签player相关信息。
    {
      # Nebula Graph中对应的标签名称。
      name: player
      type: {
        # 指定数据源文件格式，设置为hive。
        source: hive
        # 指定如何将点数据导入Nebula Graph：Client或SST。
        sink: client
      }

      # 设置读取数据库basketball中player表数据的SQL语句
      exec: "select playerid, age, name from basketball.player"

      # 在fields里指定player表中的列名称，其对应的value会作为Nebula Graph中指定属性。
      # fields和nebula.fields里的配置必须一一对应。
      # 如果需要指定多个列名称，用英文逗号 (,) 隔开。
      fields: [age,name]
      nebula.fields: [age,name]

      # 指定表中某一列数据为Nebula Graph中点VID的来源。
```


2021 Vesoft Inc.

```
      # vertex.field的值必须与上述fields中的列名保持一致。
      vertex:{
        field:playerid
      }

      # 单批次写入 Nebula Graph 的最大点数据量。
      batch: 256

      # Spark 分区数量
      partition: 32
    }
    # 设置标签team相关信息。
    {
      name: team
      type: {
        source: hive
        sink: client
      }
      exec: "select teamid, name from basketball.team"
      fields: [name]
      nebula.fields: [name]
      vertex: {
        field: teamid
      }
      batch: 256
      partition: 32
    }

  ]

  # 处理边数据
  edges: [
    # 设置边类型follow相关信息
    {
      # Nebula Graph中对应的边类型名称。
      name: follow

      type: {
        # 指定数据源文件格式，设置为hive。
        source: hive

        # 指定边数据导入Nebula Graph的方式，
        # 指定如何将点数据导入Nebula Graph：Client或SST。
        sink: client
      }

      # 设置读取数据库basketball中follow表数据的SQL语句。
      exec: "select src_player, dst_player, degree from basketball.follow"

      # 在fields里指定follow表中的列名称，其对应的value会作为Nebula Graph中指定属性。
      # fields和nebula.fields里的配置必须一一对应。
      # 如果需要指定多个列名称，用英文逗号（,）隔开。
      fields: [degree]
      nebula.fields: [degree]

      # 在source里，将follow表中某一列作为边的起始点数据源。
      # 在target里，将follow表中某一列作为边的目的点数据源。
      source: {
        field: src_player
      }

      target: {
        field: dst_player
      }

      # 单批次写入 Nebula Graph 的最大点数据量。
      batch: 256

      # Spark 分区数量
      partition: 32
    }

    # 设置边类型serve相关信息
    {
      name: serve
      type: {
        source: hive
        sink: client
      }
      exec: "select playerid, teamid, start_year, end_year from basketball.serve"
      fields: [start_year,end_year]
      nebula.fields: [start_year,end_year]
      source: {
        field: playerid
      }
      target: {
        field: teamid
      }
      batch: 256
      partition: 32
    }
  ]
}
```


2021 Vesoft Inc.

步骤 4：向NEBULA GRAPH导入数据

运行如下命令将Hive数据导入到Nebula Graph中。关于参数的说明，请参见导入命令参数。

```
${SPARK_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange <nebula-exchange-2.0.0.jar_path> -c <hive_application.conf_path> -h
```

> **Note**
>
> JAR包有两种获取方式：自行编译或者从maven仓库下载。

示例：

```
${SPARK_HOME}/bin/spark-submit  --master "local" --class com.vesoft.nebula.exchange.Exchange  /root/nebula-spark-utils/nebula-exchange/target/nebula-exchange-2.0.0.jar  -c /root/nebula-spark-utils/nebula-exchange/target/classes/hive_application.conf -h
```

用户可以在返回信息中搜索 `batchSuccess.<tag_name/edge_name>`，确认成功的数量。例如例如 `batchSuccess.follow: 300` 。

步骤 5：（可选）验证数据

用户可以在Nebula Graph客户端（例如Nebula Graph Studio）中执行查询语句，确认数据是否已导入。例如：

```
GO FROM "player100" OVER follow;
```

用户也可以使用命令 `SHOW STATS` 查看统计数据。

步骤 6：（如有）在NEBULA GRAPH中重建索引

导入数据后，用户可以在Nebula Graph中重新创建并重建索引。详情请参见索引介绍。

最后更新: 2021年5月14日

2021 Vesoft Inc.

16.4.9 导入Pulsar数据

本文简单说明如何使用Exchange将存储在Pulsar上的数据导入Nebula Graph。

环境配置

本文示例在MacOS下完成，以下是相关的环境配置信息：

- 硬件规格：
  - CPU：1.7 GHz Quad-Core Intel Core i7
  - 内存：16 GB
- Spark：2.4.7，单机版
- Nebula Graph：2.0.0。使用Docker Compose部署。

前提条件

开始导入数据之前，用户需要确认以下信息：

- 已经安装部署Nebula Graph并获取如下信息：
  - Graph服务和Meta服务的的IP地址和端口。
  - 拥有Nebula Graph写权限的用户名和密码。
- 已经编译Exchange。详情请参见编译Exchange。本示例中使用Exchange 2.0。
- 已经安装Spark。
- 了解Nebula Graph中创建Schema的信息，包括标签和边类型的名称、属性等。
- 已经安装并开启Pulsar服务。

2021 Vesoft Inc.

操作步骤

步骤 1：在NEBULA GRAPH中创建SCHEMA

分析数据，按以下步骤在Nebula Graph中创建Schema：

1. 确认Schema要素。Nebula Graph中的Schema要素如下表所示。

要素	名称	属性
标签（Tag）	`player`	`name string, age int`
标签（Tag）	`team`	`name string`
边类型（Edge Type）	`follow`	`degree int`
边类型（Edge Type）	`serve`	`start_year int, end_year int`

2. 在Nebula Graph中创建一个图空间**basketballplayer**，并创建一个Schema，如下所示。

```
## 创建图空间
nebula> CREATE SPACE basketballplayer \
        (partition_num = 10, \
        replica_factor = 1, \
        vid_type = FIXED_STRING(30));

## 选择图空间basketballplayer
nebula> USE basketballplayer;

## 创建标签player
nebula> CREATE TAG player(name string, age int);

## 创建标签team
nebula> CREATE TAG team(name string);

## 创建边类型follow
nebula> CREATE EDGE follow(degree int);

## 创建边类型serve
nebula> CREATE EDGE serve(start_year int, end_year int);
```

更多信息，请参见快速开始。

步骤 2：修改配置文件

编译Exchange后，复制 `target/classes/application.conf` 文件设置Pulsar数据源相关的配置。在本示例中，复制的文件名为 `pulsar_application.conf` 。各个配置项的详细说明请参见配置说明。

```
{
  # Spark相关配置
  spark: {
    app: {
      name: Nebula Exchange 2.0
    }
    driver: {
      cores: 1
      maxResultSize: 1G
    }
    cores {
      max: 16
    }
  }


  # Nebula Graph相关配置
  nebula: {
    address:{
      # 以下为Nebula Graph的Graph服务和Meta服务所在机器的IP地址及端口。
      # 如果有多个地址，格式为 "ip1:port","ip2:port","ip3:port"。
      # 不同地址之间以英文逗号 (,) 隔开。
      graph:["127.0.0.1:9669"]
      meta:["127.0.0.1:9559"]
    }
    # 填写的账号必须拥有Nebula Graph相应图空间的写数据权限。
    user: root
    pswd: nebula
    # 填写Nebula Graph中需要写入数据的图空间名称。
    space: basketballplayer
    connection {
      timeout: 3000
      retry: 3
    }
    execution {
```


2021 Vesoft Inc.

```
    retry: 3
  }
  error: {
    max: 32
    output: /tmp/errors
  }
  rate: {
    limit: 1024
    timeout: 1000
  }
}
# 处理点
tags: [
  # 设置标签player相关信息。
  {
    # Nebula Graph中对应的标签名称。
    name: player
    type: {
      # 指定数据源文件格式，设置为Pulsar。
      source: pulsar
      # 指定如何将点数据导入Nebula Graph：Client或SST。
      sink: client
    }
    # Pulsar服务器地址。
    service: "pulsar://127.0.0.1:6650"
    # 连接pulsar的admin.url。
    admin: "http://127.0.0.1:8081"
    # Pulsar的选项，可以从topic、topics和topicsPattern选择一个进行配置。
    options: {
      topics: "topic1,topic2"
    }

    # 在fields里指定player表中的列名称，其对应的value会作为Nebula Graph中指定属性。
    # fields和nebula.fields里的配置必须一一对应。
    # 如果需要指定多个列名称，用英文逗号（,）隔开。
    fields: [age,name]
    nebula.fields: [age,name]

    # 指定表中某一列数据为Nebula Graph中点VID的来源。
    vertex:{
        field:playerid
    }


    # 单次写入 Nebula Graph 的最大点数据量。
    batch: 10

    # Spark 分区数量
    partition: 10
    # 读取消息的间隔。单位：秒。
    interval.seconds: 10
  }
  # 设置标签team相关信息。
  {
    name: team
    type: {
      source: pulsar
      sink: client
    }
    service: "pulsar://127.0.0.1:6650"
    admin: "http://127.0.0.1:8081"
    options: {
      topics: "topic1,topic2"
    }
    fields: [name]
    nebula.fields: [name]
    vertex:{
        field:teamid
    }
    batch: 10
    partition: 10
    interval.seconds: 10
  }

]

# 处理边数据
edges: [
  # 设置边类型follow相关信息
  {
    # Nebula Graph中对应的边类型名称。
    name: follow

    type: {
      # 指定数据源文件格式，设置为Pulsar。
      source: pulsar

      # 指定边数据导入Nebula Graph的方式，
      # 指定如何将点数据导入Nebula Graph：Client或SST。
      sink: client
    }

    # Pulsar服务器地址。
    service: "pulsar://127.0.0.1:6650"
```


2021 Vesoft Inc.

```
      # 连接pulsar的admin.url。
      admin: "http://127.0.0.1:8081"
      # Pulsar的选项，可以从topic、topics和topicsPattern选择一个进行配置。
      options: {
        topics: "topic1,topic2"
      }

      # 在fields里指定follow表中的列名称，其对应的value会作为Nebula Graph中指定属性。
      # fields和nebula.fields里的配置必须一一对应。
      # 如果需要指定多个列名称，用英文逗号（,）隔开。
      fields: [degree]
      nebula.fields: [degree]

      # 在source里，将follow表中某一列作为边的起始点数据源。
      # 在target里，将follow表中某一列作为边的目的点数据源。
      source:{
          field:src_player
      }


      target:{
          field:dst_player
      }


      # 单次写入 Nebula Graph 的最大点数据量。
      batch: 10

      # Spark 分区数量
      partition: 10

      # 读取消息的间隔。单位：秒。
      interval.seconds: 10
    }

    # 设置边类型serve相关信息
    {
      name: serve
      type: {
        source: Pulsar
        sink: client
      }
      service: "pulsar://127.0.0.1:6650"
      admin: "http://127.0.0.1:8081"
      options: {
        topics: "topic1,topic2"
      }

      fields: [start_year,end_year]
      nebula.fields: [start_year,end_year]
      source:{
          field:playerid
      }

      target:{
          field:teamid
      }

      batch: 10
      partition: 10
      interval.seconds: 10
    }
  ]
}
```

步骤 3：向NEBULA GRAPH导入数据

运行如下命令将Pulsar数据导入到Nebula Graph中。关于参数的说明，请参见导入命令参数。

```
${SPARK_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange <nebula-exchange-2.0.0.jar_path> -c <pulsar_application.conf_path>
```

> **Note**
>
> JAR包有两种获取方式：自行编译或者从maven仓库下载。

示例：

```
${SPARK_HOME}/bin/spark-submit  --master "local" --class com.vesoft.nebula.exchange.Exchange  /root/nebula-spark-utils/nebula-exchange/target/nebula-exchange-2.0.0.jar  -c /root/nebula-spark-utils/nebula-exchange/target/classes/pulsar_application.conf
```

用户可以在返回信息中搜索 `batchSuccess.<tag_name/edge_name>`，确认成功的数量。例如例如 `batchSuccess.follow: 300` 。

2021 Vesoft Inc.

步骤 4：（可选）验证数据

用户可以在Nebula Graph客户端（例如Nebula Graph Studio）中执行查询语句，确认数据是否已导入。例如：

```
GO FROM "player100" OVER follow;
```

用户也可以使用命令 `SHOW STATS` 查看统计数据。

步骤 5：（如有）在NEBULA GRAPH中重建索引

导入数据后，用户可以在Nebula Graph中重新创建并重建索引。详情请参见索引介绍。

最后更新: 2021年5月14日

2021 Vesoft Inc.

16.4.10 导入Kafka数据

本文简单说明如何使用Exchange将存储在Kafka上的数据导入Nebula Graph。

环境配置

本文示例在MacOS下完成，以下是相关的环境配置信息：

- 硬件规格：
  - CPU：1.7 GHz Quad-Core Intel Core i7
  - 内存：16 GB
- Spark：2.4.7，单机版
- Nebula Graph：2.0.0。使用Docker Compose部署。

前提条件

开始导入数据之前，用户需要确认以下信息：

- 已经安装部署Nebula Graph并获取如下信息：
  - Graph服务和Meta服务的的IP地址和端口。
  - 拥有Nebula Graph写权限的用户名和密码。
- 已经编译Exchange。详情请参见编译Exchange。本示例中使用Exchange 2.0。
- 已经安装Spark。
- 了解Nebula Graph中创建Schema的信息，包括标签和边类型的名称、属性等。
- 已经安装并开启Kafka服务。

2021 Vesoft Inc.

操作步骤

步骤 1：在NEBULA GRAPH中创建SCHEMA

分析数据，按以下步骤在Nebula Graph中创建Schema：

1. 确认Schema要素。Nebula Graph中的Schema要素如下表所示。

要素	名称	属性
标签（Tag）	`player`	`name string, age int`
标签（Tag）	`team`	`name string`
边类型（Edge Type）	`follow`	`degree int`
边类型（Edge Type）	`serve`	`start_year int, end_year int`

2. 在Nebula Graph中创建一个图空间**basketballplayer**，并创建一个Schema，如下所示。

```
## 创建图空间
nebula> CREATE SPACE basketballplayer \
        (partition_num = 10, \
        replica_factor = 1, \
        vid_type = FIXED_STRING(30));

## 选择图空间basketballplayer
nebula> USE basketballplayer;

## 创建标签player
nebula> CREATE TAG player(name string, age int);

## 创建标签team
nebula> CREATE TAG team(name string);

## 创建边类型follow
nebula> CREATE EDGE follow(degree int);

## 创建边类型serve
nebula> CREATE EDGE serve(start_year int, end_year int);
```

更多信息，请参见快速开始。

步骤 2：修改配置文件

编译Exchange后，复制 `target/classes/application.conf` 文件设置Kafka数据源相关的配置。在本示例中，复制的文件名为 `kafka_application.conf` 。各个配置项的详细说明请参见配置说明。

```
{
  # Spark相关配置
  spark: {
    app: {
      name: Nebula Exchange 2.0
    }
    driver: {
      cores: 1
      maxResultSize: 1G
    }
    cores {
      max: 16
    }
  }


  # Nebula Graph相关配置
  nebula: {
    address:{
      # 以下为Nebula Graph的Graph服务和Meta服务所在机器的IP地址及端口。
      # 如果有多个地址，格式为 "ip1:port","ip2:port","ip3:port"。
      # 不同地址之间以英文逗号 (,) 隔开。
      graph:["127.0.0.1:9669"]
      meta:["127.0.0.1:9559"]
    }
    # 填写的账号必须拥有Nebula Graph相应图空间的写数据权限。
    user: root
    pswd: nebula
    # 填写Nebula Graph中需要写入数据的图空间名称。
    space: basketballplayer
    connection {
      timeout: 3000
      retry: 3
    }
    execution {
```


2021 Vesoft Inc.

```
    retry: 3
  }
  error: {
    max: 32
    output: /tmp/errors
  }
  rate: {
    limit: 1024
    timeout: 1000
  }
}
# 处理点
tags: [
  # 设置标签player相关信息。
  {
    # Nebula Graph中对应的标签名称。
    name: player
    type: {
      # 指定数据源文件格式，设置为Kafka。
      source: kafka
      # 指定如何将点数据导入Nebula Graph：Client或SST。
      sink: client
    }
    # Kafka服务器地址。
    service: "127.0.0.1:9092"
    # 消息类别。
    topic: "topic_name"

    # 在fields里指定player表中的列名称，其对应的value会作为Nebula Graph中指定属性。
    # fields和nebula.fields里的配置必须一一对应。
    # 如果需要指定多个列名称，用英文逗号（,）隔开。
    fields: [age,name]
    nebula.fields: [age,name]

    # 指定表中某一列数据为Nebula Graph中点VID的来源。
    vertex:{
        field:playerid
    }


    # 单次写入 Nebula Graph 的最大点数据量。
    batch: 10

    # Spark 分区数量
    partition: 10
    # 读取消息的间隔。单位：秒。
    interval.seconds: 10
  }
  # 设置标签team相关信息。
  {
    name: team
    type: {
      source: kafka
      sink: client
    }
    service: "127.0.0.1:9092"
    topic: "topic_name"
    fields: [name]
    nebula.fields: [name]
    vertex:{
        field:teamid
    }
    batch: 10
    partition: 10
    interval.seconds: 10
  }

]

# 处理边数据
edges: [
  # 设置边类型follow相关信息
  {
    # Nebula Graph中对应的边类型名称。
    name: follow

    type: {
      # 指定数据源文件格式，设置为Kafka。
      source: kafka

      # 指定边数据导入Nebula Graph的方式，
      # 指定如何将点数据导入Nebula Graph：Client或SST。
      sink: client
    }

    # Kafka服务器地址。
    service: "127.0.0.1:9092"
    # 消息类别。
    topic: "topic_name"

    # 在fields里指定follow表中的列名称，其对应的value会作为Nebula Graph中指定属性。
    # fields和nebula.fields里的配置必须一一对应。
    # 如果需要指定多个列名称，用英文逗号（,）隔开。
    fields: [degree]
```


2021 Vesoft Inc.

```
      nebula.fields: [degree]

      # 在source里，将follow表中某一列作为边的起始点数据源。
      # 在target里，将follow表中某一列作为边的目的点数据源。
      source:{
          field:src_player
      }


      target:{
          field:dst_player
      }


      # 单次写入 Nebula Graph 的最大点数据量。
      batch: 10

      # Spark 分区数量
      partition: 10

      # 读取消息的间隔。单位：秒。
      interval.seconds: 10
    }

    # 设置边类型serve相关信息
    {
      name: serve
      type: {
        source: kafka
        sink: client
      }
      service: "127.0.0.1:9092"
      topic: "topic_name"

      fields: [start_year,end_year]
      nebula.fields: [start_year,end_year]
      source:{
          field:playerid
      }

      target:{
          field:teamid
      }

      batch: 10
      partition: 10
      interval.seconds: 10
    }
  ]
}
```

步骤 3：向NEBULA GRAPH导入数据

运行如下命令将Kafka数据导入到Nebula Graph中。关于参数的说明，请参见导入命令参数。

```
${SPARK_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange <nebula-exchange-2.0.0.jar_path> -c <kafka_application.conf_path>
```

> **Note**
>
> JAR包有两种获取方式：自行编译或者从maven仓库下载。

示例：

```
${SPARK_HOME}/bin/spark-submit  --master "local" --class com.vesoft.nebula.exchange.Exchange  /root/nebula-spark-utils/nebula-exchange/target/nebula-exchange-2.0.0.jar  -c /root/nebula-spark-utils/nebula-exchange/target/classes/kafka_application.conf
```

用户可以在返回信息中搜索 `batchSuccess.<tag_name/edge_name>`，确认成功的数量。例如例如 `batchSuccess.follow: 300` 。

步骤 4：（可选）验证数据

用户可以在Nebula Graph客户端（例如Nebula Graph Studio）中执行查询语句，确认数据是否已导入。例如：

```
GO FROM "player100" OVER follow;
```

用户也可以使用命令 `SHOW STATS` 查看统计数据。

步骤 5：（如有）在NEBULA GRAPH中重建索引

导入数据后，用户可以在Nebula Graph中重新创建并重建索引。详情请参见索引介绍。

2021 Vesoft Inc.

最后更新: 2021年5月14日

2021 Vesoft Inc.

16.5 Exchange常见问题

16.5.1 编译问题

部分非central仓库的包下载失败，报错`Could not resolve dependencies for project xxx`

请检查Maven安装目录下 `libexec/conf/settings.xml` 文件的 `mirror` 部分：

```
<mirror>
    <id>alimaven</id>
    <mirrorOf>central</mirrorOf>
    <name>aliyun maven</name>
    <url>http://maven.aliyun.com/nexus/content/repositories/central/</url>
</mirror>
```

检查 `mirrorOf` 的值是否配置为 `*`，如果为 `*`，请修改为 `central` 或 `*,!SparkPackagesRepo,!bintray-streamnative-maven`。

原因：Exchange的 `pom.xml` 中有两个依赖包不在Maven的central仓库中，`pom.xml` 配置了这两个依赖所在的仓库地址。如果maven中配置的镜像地址对应的 `mirrorOf` 值为 `*`，那么所有依赖都会在central仓库下载，导致下载失败。

16.5.2 执行问题

报错`method name xxx not found`

一般是端口配置错误，需检查Meta服务、Graph服务、Storage服务的端口配置。

报NoSuchMethod、MethodNotFound错误（`Exception in thread "main" java.lang.NoSuchMethodError`等）

绝大多数是因为JAR包冲突和版本冲突导致的报错，请检查报错服务的版本，与Exchange中使用的版本进行对比，检查是否一致，尤其是Spark版本、Scala版本、Hive版本。

Exchange导入Hive数据时报错`Exception in thread "main" org.apache.spark.sql.AnalysisException: Table or view not found`

检查提交exchange任务的命令中是否遗漏参数 `-h`，检查table和database是否正确，在spark-sql中执行用户配置的exec语句，验证exec语句的正确性。

2021 Vesoft Inc.

运行时报错`com.facebook.thrift.protocol.TProtocolException: Expected protocol id xxx`

请检查Nebula Graph服务端口配置是否正确。

- 如果是源码、RPM或DEB安装，请配置各个服务的配置文件中`--port`对应的端口号。
- 如果是docker安装，请配置docker映射出来的端口号，查看方式如下：

  在`nebula-docker-compose`目录下执行`docker-compose ps`，例如：

```
$ docker-compose ps
              Name                            Command                  State                                                          Ports
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
nebula-docker-compose_graphd_1      /usr/local/nebula/bin/nebu ...   Up (healthy)   0.0.0.0:33205->19669/tcp, 0.0.0.0:33204->19670/tcp, 0.0.0.0:9669->9669/tcp
nebula-docker-compose_metad0_1      ./bin/nebula-metad --flagf ...   Up (healthy)   0.0.0.0:33165->19559/tcp, 0.0.0.0:33162->19560/tcp, 0.0.0.0:33167->9559/tcp, 9560/tcp
nebula-docker-compose_metad1_1      ./bin/nebula-metad --flagf ...   Up (healthy)   0.0.0.0:33166->19559/tcp, 0.0.0.0:33163->19560/tcp, 0.0.0.0:33168->9559/tcp, 9560/tcp
nebula-docker-compose_metad2_1      ./bin/nebula-metad --flagf ...   Up (healthy)   0.0.0.0:33161->19559/tcp, 0.0.0.0:33160->19560/tcp, 0.0.0.0:33164->9559/tcp, 9560/tcp
nebula-docker-compose_storaged0_1   ./bin/nebula-storaged --fl ...   Up (healthy)   0.0.0.0:33180->19779/tcp, 0.0.0.0:33178->19780/tcp, 9777/tcp, 9778/tcp, 0.0.0.0:33183->9779/tcp,
9780/tcp
nebula-docker-compose_storaged1_1   ./bin/nebula-storaged --fl ...   Up (healthy)   0.0.0.0:33175->19779/tcp, 0.0.0.0:33172->19780/tcp, 9777/tcp, 9778/tcp, 0.0.0.0:33177->9779/tcp,
9780/tcp
nebula-docker-compose_storaged2_1   ./bin/nebula-storaged --fl ...   Up (healthy)   0.0.0.0:33184->19779/tcp, 0.0.0.0:33181->19780/tcp, 9777/tcp, 9778/tcp, 0.0.0.0:33185->9779/tcp,
9780/tcp
```

  查看`Ports`列，查找docker映射的端口号，例如：

  - Graph服务可用的端口号是9669。

  - Meta服务可用的端口号有33167、33168、33164。

  - Storage服务可用的端口号有33183、33177、33185。

16.5.3 配置问题

哪些配置项影响导入性能？

- batch：每次发送给Nebula Graph服务的nGQL语句中包含的数据条数。
- partition：Spark数据的分区数，表示数据导入的并发数。
- nebula.rate：向Nebula Graph发送请求前先去令牌桶获取令牌。

  - limit：表示令牌桶的大小。

  - timeout：表示获取令牌的超时时间。

根据机器性能可适当调整这四项参数的值。如果在导入过程中，Storage服务的leader变更，可以适当调小这四项参数的值，降低导入速度。

16.5.4 其他问题

Exchange支持哪些版本的Nebula Graph？

请参见Exchange的使用限制。

Exchange与Spark Writer有什么关系？

Exchange是在Spark Writer基础上开发的Spark应用程序，二者均适用于在分布式环境中将集群的数据批量迁移到Nebula Graph中，但是后期的维护工作将集中在 Exchange上。与Spark Writer相比，Exchange有以下改进：

- 支持更丰富的数据源，如MySQL、Neo4j、Hive、HBase、Kafka、Pulsar等。
- 修复了Spark Writer的部分问题。例如Spark读取HDFS里的数据时，默认读取到的源数据均为String类型，可能与Nebula Graph定义的Schema不同，所以Exchange增加了数据类型的自动匹配和类型转换，当Nebula Graph定义的Schema中数据类型为非String类型（如double）时，Exchange会将String类型的源数据转换为对应的类型（如double）。

最后更新: 2021年5月14日

2021 Vesoft Inc.

17. 附录

17.1 常见问题

本文列出了使用Nebula Graph book 时可能遇到的常见问题。

17.1.1 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             |
+------------------+
| basketballplayer |
+------------------+
Got 1 rows (time spent 1235/1934 us)
```

- 第一个数字 `1235` 表示数据库本身执行该命令花费的时间，即查询引擎从客户端接收到一个查询，然后从存储服务器获取数据并执行一系列计算所花费的时间。
- 第二个数字 `1934` 表示从客户端角度看所花费的时间，即从客户端发送请求、接收结果，然后在屏幕上显示结果所花费的时间。

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。

如何处理错误信息`[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。

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、LOOKUP和MATCH。

2021 Vesoft Inc.

如何处理错误信息can’t solve the start vids from the sentence

查询引擎需要知道从哪些VID开始图遍历. 这些开始（图遍历）的VID，或者通过用户指定，例如

```
> GO FROM ${vids} ...
> MATCH (src) WHERE id(src) == ${vids}
# 开始图遍历的VID通过如上办法指定
```

或者通过一个(属性)索引来得到，例如

```
# CREATE TAG INDEX 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`。

如何处理错误信息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.

如何处理错误信息edge conflict或vertex conflict

Storage服务在毫秒级时间内多次收到插入或者更新同一点或边的请求时，可能返回该错误。请稍后重试。

如何处理错误信息Storage Error E_RPC_FAILURE

报错原因通常为Graph服务向Storage服务请求了过多的数据，导致Storage服务超时。请尝试以下解决方案：

- 修改配置文件: 在 `nebula-graphd.conf` 文件中修改 `--storage_client_timeout_ms` 参数的值，以增加Storage client的连接超时时间。该值的单位为毫秒（ms）。例如，设置 `--storage_client_timeout_ms=60000`。如果 `nebula-graphd.conf` 文件中未配置该参数，请手动增加。提示：请在配置文件开头添加--local_config=true再重启服务.
- 优化查询语句，减少全库扫描型的查询（包括含有 `LIMIT` 的该类语句）。
- 检查Storaged是否发生的 OOM。(`dmesg |grep nebula`)
- 为Storage服务器提供性能更好的SSD或者内存。
- 重试请求。

如何处理错误信息The leader has changed. Try again later

已知问题，通常需要重试1-N次(N==partition数量)。原因为 meta client 更新leader缓存需要1-2个心跳或者通过错误触发强制更新。

是否支持停止或者中断慢查询

不支持。即使关闭了客户端，服务端仍会尽力把该查询执行完毕，无法中断。

17.1.2 关于运维

日志文件过大时如何回收日志？

Nebula Graph 的日志默认在 `/usr/local/nebula/logs/` 下，正常INFO级别日志文件为 `nebula-graphd.INFO`, `nebula-storaged.INFO`, `nebula-metad.INFO`，报警和错误级别后缀为 `.WARNING` 和 `.ERROR`.

Nebula Graph使用 glog 打印日志。glog 没有日志回收的功能，用户可以使用 crontab 设置定期任务回收日志文件，详情请参见Glog should delete old log files automatically。

2021 Vesoft Inc.

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页的版本发布时间，即可确定Nebula Graph的版本。

如何扩缩容

Nebula Graph book 未提供运维命令以实现自动扩缩容，参考以下步骤：

- metad 的扩容和缩容： metad 不支持扩缩容，也不支持迁移到新机器，也不要增加新的 metad 进程。
- graphd 的缩容： 将该graphd 的 ip 从 client 的代码中移除，关闭该 graphd 进程。
- graphd 的扩容： 在新机器上准备 graphd 二进制文件和配置文件，在配置文件中修改或增加已在运行的 metad 地址，启动 graphd 进程。
- storaged 的缩容：（副本数都必须大于1），参考缩容命令。完成后关闭 storaged 进程。
- storaged 的扩容：（副本数都必须大于1） 在新机器上准备 storaged 二进制文件和配置文件，在配置文件中修改或增加已在运行的 metad 地址，启动 storaged 进程。

storaged扩缩容之后，还需要运行Balance Data 和 Balance Leader 命令。

17.1.3 关于连接

防火墙中需要开放哪些端口

如果没有修改过配置文件中预设的端口，请在防火墙中开放如下端口：

服务类型	端口
Meta	9559, 9560, 19559, 19560
Graph	9669, 19669, 19670
Storage	9777 ~ 9780, 19779, 19780

如果修改过配置文件中预设的端口，请找出实际使用的端口并在防火墙中开放它们。

如何测试端口是否已开放

用户可以使用如下telnet命令检查端口状态：

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

> **Note**
>
> 如果无法使用telnet命令，请先检查主机中是否安装并启动了telnet。

示例：

```
// 如果端口已开放：
$ telnet 192.168.1.10 9669
Trying 192.168.1.10...
Connected to 192.168.1.10.
```


2021 Vesoft Inc.

```
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
```

最后更新: 2021年5月24日

2021 Vesoft Inc.

17.2 图建模与系统设计

本文介绍在Nebula Graph项目中成功应用的一些图建模和系统设计的通用建议。

> **Note**
>
> 本文建议是通用的，在特定领域有例外，请结合实际业务情况进行图建模。

17.2.1 以性能为目标进行建模

目前 Nebula Graph 没有完美的建模方法，如何建模取决于想从数据中挖掘的内容。分析数据并根据业务模型创建方便直观的数据模型，测试模型并优化，逐渐适应业务。为了更好的性能，用户可以多次更改或重新设计模型。

合理设置边属性

- Nebula Graph 基于图拓扑结构进行深度图遍历的性能较低，但是广度优先遍历以及获取属性的性能较好。因此为了减少遍历深度，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：虽然 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，注意写入放大问题（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

使用属性索引可以通过属性查找到 VID。但是索引会导致写性能下降90%甚至更多，只有在根据点或边的属性定位点或边时才使用索引。

合理设计VID

参考点VID一节。

最后更新: 2021年5月18日

2021 Vesoft Inc.

17.3 系统设计建议

17.3.1 选择吞吐量优先或时延优先

- Nebula Graph book 更擅长处理（互联网式的）有大量并发的小请求。也即：虽然全图很大（万亿点边），但是每个请求要访问到的子图本身并不大（几百万个点边）——单个请求时延不大；但这类请求的并发数量特别多——吞吐量大。
- 但对于一些相互分析型的场景，并发请求的数量不多，而每个请求要访问的子图本身特别大（亿以上）。为降低时延，可以在应用程序中将一个大的请求，拆分为多个小请求，并发发送给多个graphd。这样可以降低单个请求的时延和graphd的内存占用。

17.3.2 水平扩展或垂直扩展

Nebula Graph book 支持水平扩展：

- Storaged 的水平扩展：

  - 增加storaged的机器数量，可以大体线性增加集群的整体能力，包括增加整体吞吐量和降低时延。

  - 但由于 partition 数量在 CREATE SPACE 时已固定，因此单个 partition 的服务能力只由单服务器决定——例如：获取单个点的属性（`FETCH`）、单个点开始的广度优先遍历（`GO`）

- Graphd 的水平扩展：

  - 来自客户端的每个请求，都由且仅由一个 graphd 处理，其他 graphd 不会参与处理该请求。

  - 因此增加 graphd 机器数量，可以增加集群整体吞吐量，但不能降低单个请求时延。

- Metad 不支持水平扩展。

垂直扩展通常硬件成本更高，但运维操作相对简单。Nebula Graph book 也可以直接垂直扩展。

17.3.3 数据传输与优化

- 读写平衡。Nebula Graph 适合读写平衡性的在线场景，也即“并发的写入与读取”；而非数仓型的“一次写入多次读取”。
- 选择不同的写入方式。大批量的数据写入可以使用sst加载的方式；小批量的写入使用 `INSERT` 语句。
- 选择合适的时间运行 COMPACTION 和 BALANCE，来分别优化数据格式和存储分布。
- Nebula Graph book 不支持关系型数据库意义上的事务和隔离性，更接近 NoSQL。

17.3.4 查询预热与数据预热

- Graphd 不支持预编译查询及相应生成查询计划，也不支持缓存之前的查询结果；
- Storaged 不支持预热数据，只有 RocksDB 自身的 LSM-tree 和 BloomFilter 会启动时加载到内存中。
- 点和边被访问过后，会各自缓存在 storaged 的两种 (LRU) Cache 中。

因此只能应用端来进行预热。

最后更新: 2021年5月26日

2021 Vesoft Inc.

17.4 关于 Raft 的简单介绍

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

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

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

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

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

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

Listener：这是一种特殊的 Raft 角色，并不参与投票，也不能用于多副本的数据一致性。在Nebula Graph中，其唯一作用是从 Leader 读取 Raft-wal，并向ElasticSearch集群同步。

最后更新: 2021年5月26日

2021 Vesoft Inc.

17.5 分片ID

点和边分布在不同的分片，分片分布在不同的机器。分片数量在 CREATE SPACE 语句中指定，此后不可更改。

如果需要将某些点放置在相同的分片（例如在一台机器上），可以参考公式或代码。

下文用简单代码说明VID和分片的关系。

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

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

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

示例的部分参数说明如下。

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

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

最后更新: 2021年5月18日

2021 Vesoft Inc.

17.6 生态工具概览

不同版本的生态工具支持的Nebula Graph内核版本不同，本文介绍Nebula Graph内核版本 book 和生态工具版本的对应关系。

> **Note**
>
> 1.x版本的生态工具不支持在Nebula Graph 2.x版本中使用。

> **Caution**
>
> 大多数工具和客户端**未发布**相对内核 book 版本的适配。但通常2.0.0的客户端前后向兼容 book 的内核版本。

17.6.1 Nebula Graph Studio

Nebula Graph Studio（简称 Studio）是一款可以通过Web访问的图数据库可视化工具，搭配Nebula Graph DBMS使用，提供构图、数据导入、编写nGQL查询、图探索等一站式服务。详情请参见什么是Nebula Graph Studio。

Nebula Graph版本	Studio版本
book	2.2.0

17.6.2 Nebula Exchange

Nebula Exchange（简称Exchange）是一款Apache Spark™应用，用于在分布式环境中将集群中的数据批量迁移到Nebula Graph中，能支持多种不同格式的批式数据和流式数据的迁移。详情请参见什么是Nebula Exchange。

Nebula Graph版本	Exchange版本（commit id）
book	2.0.1（cfc15bd）、2.0.0（af3fdf4）

17.6.3 Nebula Importer

Nebula Importer（简称Importer）是一款Nebula Graph的CSV文件导入工具。Importer可以读取本地的CSV文件，然后导入数据至Nebula Graph图数据库中。详情请参见什么是Nebula Importer。

Nebula Graph版本	Importer版本（commit id）
book	2.0.0（1d87c7b）

17.6.4 Nebula Spark Connector

Nebula Spark Connector是一个Spark连接器，提供通过Spark标准形式读写Nebula Graph数据的能力。Nebula Spark Connector由Reader和Writer两部分组成。详情请参见什么是Nebula Spark Connector。

Nebula Graph版本	Spark Connector版本（commit id）
book	2.0.1（cfc15bd）、2.0.0（af3fdf4）

2021 Vesoft Inc.

17.6.5 Nebula Flink Connector

Nebula Flink Connector是一款帮助Flink用户快速访问Nebula Graph的连接器，支持从Nebula Graph图数据库中读取数据，或者将其他外部数据源读取的数据写入Nebula Graph图数据库。详情请参见什么是Nebula Flink Connector。

Nebula Graph版本	Flink Connector版本（commit id）
book	2.0.0（34dc606）

17.6.6 Nebula Algorithm

Nebula Algorithm（简称Algorithm）是一款基于GraphX的Spark应用程序，通过提交Spark任务的形式使用完整的算法工具对Nebula Graph数据库中的数据执行图计算，也可以通过编程形式调用lib库下的算法针对DataFrame执行图计算。详情请参见什么是Nebula Algorithm。

Nebula Graph版本	Algorithm版本（commit id）
book	2.0.0（52d2b3d）

17.6.7 Nebula Console

Nebula Console是Nebula Graph的原生CLI客户端。如何使用请参见连接Nebula Graph。

Nebula Graph版本	Console版本（commit id）
book	2.0.0（1f32236）

17.6.8 Nebula Docker Compose

Docker Compose可以快速部署Nebula Graph集群。如何使用请参见Docker Compose部署Nebula Graph。

Nebula Graph版本	Docker Compose版本（commit id）
book	2.0.0（2c2549a）

17.6.9 API

Nebula Graph版本	语言（commit id）
book	C++（7305c72）
book	Go（542ed24）
book	Python（cb48e8a）
book	Java Client（923bc04）

2021 Vesoft Inc.

17.6.10 未发布

- API
  - Rust Client
  - Node.js Client
  - HTTP Client
- 监控
  - Promethus connector
  - Dashboard
  - [Graph Computing]
- 性能、测试、备份与恢复
  - Benchmark
  - Chaos Test
  - Backup&Restore

最后更新: 2021年5月25日

2021 Vesoft Inc.

17.7 点VID

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

17.7.1 VID 的特点

- VID数据类型只可以为定长字符串 `FIXED_STRING(<N>)` 或64位整数；一个图空间只能选用其中一种VID类型。
- VID在一个图空间中必须唯一，其作用类似于关系型数据库中的主键（索引+唯一约束）。但不同图空间中的VID是完全独立无关的。
- 点VID的生成方式必须由用户自行指定，系统不提供自增ID或者UUID。
- VID相同的点，会被认为是同一个点。例如：
  - VID相当于一个实体的唯一标号，例如一个人的身份证号。标签（Tag）相当于实体所拥有的类型，例如滴滴司机和老板。不同的Tag又相应定义了几组不同的属性，例如工号、薪水、商务手机号、债务额度。
  - 同时操作相同VID并且相同Tag的两条 `INSERT` 语句，晚写入的 `INSERT` 会覆盖先写入的。
  - 同时操作包含相同VID但是两个不同 `TAG A` 和 `TAG B` 的两条 `INSERT` 语句，对 `TAG A` 的操作不会影响 `TAG B` 。
- VID通常会被索引（LSM-tree方式）并缓存在内存中，因此直接访问VID的性能最高。（属性本身则需通过属性索引功能)

17.7.2 VID使用建议

- Nebula Graph 1.x只支持VID为int64。2.x默认使用 `FIXED_STRING(<N>)` 为VID类型；但在 `CREATE SPACE` 中通过参数 `vid_type = int64` ，也可设置为64位整数。
- 可以使用 `id()` 函数，指定或引用该点的VID；
- 可以使用 `LOOKUP` 或者 `MATCH` 语句，来通过属性索引查找对应的VID;
- 性能上，直接通过VID找到点的语句性能最高，例如 `DELETE xxx WHERE id(xxx) == "player100"` , 或者 `GO FROM "player100"` 等语句。通过属性先查找VID，再进行图操作的性能会变差，例如 `LOOKUP | GO FROM $-.ids` 等语句，相比前者多了一次内存或硬盘的随机读（ `LOOKUP` ）以及一次序列化（ `|` ）。

17.7.3 VID生成建议

VID的生成工作完全交给应用端，有一些通用的建议：

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

最后更新: 2021年5月20日

2021 Vesoft Inc.

17.8 注释

本文介绍nGQL中的注释方式。

17.8.1 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。

17.8.2 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。

17.8.3 OpenCypher兼容性

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

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

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

最后更新: 2021年5月27日

2021 Vesoft Inc.

17.9 大小写区分

17.9.1 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:
```

17.9.2 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;
```

最后更新: 2021年5月18日

2021 Vesoft Inc.

17.10 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`。

> **Note**
>
> 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，用户必须使用反引号（`）括起来。
- `SPACE` 是非保留关键字，可以直接作为标识符使用。

17.10.1 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
```


2021 Vesoft Inc.

```
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
```

17.10.2 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
```


2021 Vesoft Inc.

```
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
```

最后更新: 2021年5月18日

2021 Vesoft Inc.

17.11 Nebula Graph 年表

1. 2018.9.5 由 @dutor 提交了第一行代码。

1. 2019.5 发布了 v0.1.0 alpha 版本，并在 HBaseCon Asia2019 第一次公开亮相。此后一年内陆续发布 v1.0.0-beta, v1.0.0-rc1, v1.0.0-rc2, v1.0.0-rc3, v1.0.0-rc4

2021 Vesoft Inc.

Pre-release

v0.1.0

b0d817f

Compare

Nebula Graph v0.1.0

darionyaphet released this on May 14, 2019 · 1075 commits to master since this release

This is the first release of *Nebula Graph*, a brand new, fast and distributed graph database.

Available Features

- Physical data isolation with Graph Space
- Strongly typed schema support
- Vertices and edges insertion
- Graph traversal(the `GO` statement)
- Variable definition and reference
- Piping query result between statements
- Client API in C++, Golang and Java

Features Coming Soon

- Raft support
- Query based on secondary index(the `LOOKUP` statement)
- Sub-graph retrieval(the `MATCH` statement)
- User defined function call
- User management

Try Out

A Docker image is available for trial purpose. You can get it by following the guide here.

Assets 2

Source code (zip)

Source code (tar.gz)

1. 2020.3 在v1.0 开发的收尾阶段，启动了 v2.0 项目的研发
2. 2020.6 发布了第一个正式大版本 v1.0.0 GA

2021 Vesoft Inc.

v1.0.0
06a5db4
Verified
Compare

V1.0.0 GA

jude-zhu released this on Jun 10, 2020 · 146 commits to master since this release

Basic Features

- Online DDL & DML. Support updating schemas and data without stopping or affecting your ongoing operations.
- Graph traversal. `GO` statement supports forward/reverse and bidirectional graph traversal. `GO minHops TO maxHops` is supported to get variable hops relationships.
- Aggregate. Support aggregation functions such as `GROUP BY`, `ORDER BY`, and `LIMIT`.
- Composite query. Support composite clauses: `UNION`, `UNION DISTINCT`, `INTERSECT`, and `MINUS`.
- PIPE statements. The result yielded from the previous statement could be piped to the next statement as input.
- Use defined variables. Support user-defined variables to pass the result of a query to another.
- Index. Both the single-property index and composite index are supported to make searches of related data more efficient. `LOOKUP ON` statement is to query on the index.

Advanced Features

- Privilege Management. Support user authentication and role-based access control. Nebula Graph can easily integrate with third-party authentication systems. There are five built-in roles in Nebula Graph: `GOD`, `ADMIN`, `DBA`, `USER`, and `GUEST`. Each role has its corresponding privileges.
- Support Reservoir Sampling, which will retrieve k elements randomly for the sampling of the supernode at the complexity of `O(n)`.
- Cluster snapshot. Support creating snapshots for the cluster as an online backup strategy.
- TTL. Support TTL to expire items after a certain amount of time automatically.
- Operation & Maintenance
  - Scale in/out. Support online scale in/out and load balance for storage
  - `HOSTS` clause to manage storage hosts
  - `CONFIGS` clause to manage configuration options
- Job Manager & Scheduler. A tool for job managing and scheduling. Currently, `COMPACT` and `FLUSH` jobs are supported.
- Graph Algorithms. Support finding the full path and the shortest path between vertices.
- Provide OLAP interfaces to integrate with third-party graph analytics platforms.
- Support multiple character sets and collations. The default `CHARSET` and `COLLATE` are `utf8` and `utf8_bin`.

Clients

- Java Client. Support source code building and downloading from the MVN repository, see Java Client for more details.
- Python Client. Support source code building and installation with pip, see Python Client for more details.
- Golang Client. Install the client with the command `go get -u -v github.com/vesoft-inc/nebula-go`, see Go Client for more details.

Nebula Graph Studio

A graphical user interface for working with Nebula Graph. Support querying, designing schema, data loading, and graph exploring. See Nebula Graph Studio for more details.

1. 2021.3 发布了第二个大版本 v2.0 GA

2021 Vesoft Inc.

v2.0.0
91639db
Verified
Compare

Nebula Graph v2.0 GA

jude-zhu released this on Mar 23

New Features

- `vertexID` supports both `Integer` and `String`.
- New data types:
  - NULL: the property can be set to `NULL`. `NOT NULL` constraint is also supported
  - Composite types: LIST, SET, and MAP(Cannot be set as property types)
  - Temporal types: DATE and DATETIME.
  - FIXED_STRING: a fixed size `String`
- Full-text indexes are supported to do prefix, wildcard, regex, and fuzzy search on a string property.
- Explain & Profile outputs the execution plan of an nGQL statement and execution profile.
- Subgraph to retrieve vertices and edges reachable from the start vertices.
- Support to collect statistics of the graph space.
- OpenCypher compatibility
  - Partially support the `MATCH` clause
  - Support `RETURN`, `WITH`, `UNWIND`, `LIMIT` & `SKIP` clauses
- More built-in functions
  - Predicate functions
  - Scalar functions
  - List functions
  - Aggregating functions
  - Mathematical functions
  - String functions
  - Temporal functions

Improvements

- Optimize the performance of inserting, updating, and deleting data with indexes.
- `LOOKUP ON` filtering data supports `OR` and `AND` operators.
- `FIND PATH` supports finding paths with or without regard to direction, and also supports excluding cycles in paths.
- `SHOW HOSTS graph/meta/storage` supports to retrieve the basic information of graphd/metad/storaged hosts.

Changelog

- The data type of `vertexID` must be specified when creating a graph space.
- `FETCH PROP ON` returns a composite object if not specify the result set.
- Changed the default port numbers of `metad`, `graphd`, and `storaged`.
- Refactor metrics counters.

Nebula-graph Console

Supports local commands mode. `:set csv` outputs the query results to the console and the specified CSV file. For more information, please refer to https://github.com/vesoft-inc/nebula-console.

Clients

Support connection pool and load balance.

- cpp client https://github.com/vesoft-inc/nebula-cpp
- java client https://github.com/vesoft-inc/nebula-java
- python client https://github.com/vesoft-inc/nebula-python
- go client https://github.com/vesoft-inc/nebula-go

Nebula Graph Studio

With Studio, you can create a graph schema, load data, execute nGQL statements, and explore graphs in one stop. For more information, please refer to https://github.com/vesoft-inc/nebula-web-docker.

Known Issues

- #860

最后更新: 2021年5月20日

2021 Vesoft Inc.

2021 Vesoft Inc.