# 史上超全面的Neo4j使用指南

[像风一样](https://cloud.tencent.com/developer/user/1483357)发表于[你不就像风一样](https://cloud.tencent.com/developer/column/4942)订阅

10.3K

## 在这篇文章中：

* [第一章：介绍](https://cloud.tencent.com/developer/article/javascript:;" \o "第一章：介绍)
  + [Neo4j是什么](https://cloud.tencent.com/developer/article/javascript:;" \o "Neo4j是什么)
  + [Neo4j的特点](https://cloud.tencent.com/developer/article/javascript:;" \o "Neo4j的特点)
  + [Neo4j的优点](https://cloud.tencent.com/developer/article/javascript:;" \o "Neo4j的优点)
* [第二章：安装](https://cloud.tencent.com/developer/article/javascript:;" \o "第二章：安装)
  + [1.环境](https://cloud.tencent.com/developer/article/javascript:;" \o "1.环境)
  + [2.下载](https://cloud.tencent.com/developer/article/javascript:;" \o "2.下载)
  + [3.开启远程访问](https://cloud.tencent.com/developer/article/javascript:;" \o "3.开启远程访问)
  + [4.测试](https://cloud.tencent.com/developer/article/javascript:;" \o "4.测试)
* [第三章：CQL](https://cloud.tencent.com/developer/article/javascript:;" \o "第三章：CQL)
  + [1.CQL简介](https://cloud.tencent.com/developer/article/javascript:;" \o "1.CQL简介)
  + [2.Neo4j CQL命令/条款](https://cloud.tencent.com/developer/article/javascript:;" \o "2.Neo4j CQL命令/条款)
  + [3.Neo4j CQL 函数](https://cloud.tencent.com/developer/article/javascript:;" \o "3.Neo4j CQL 函数)
  + [4.Neo4j CQL数据类型](https://cloud.tencent.com/developer/article/javascript:;" \o "4.Neo4j CQL数据类型)
* [第四章：命令](https://cloud.tencent.com/developer/article/javascript:;" \o "第四章：命令)
  + [1.CREATE创建](https://cloud.tencent.com/developer/article/javascript:;" \o "1.CREATE创建)
  + [2.MATCH查询](https://cloud.tencent.com/developer/article/javascript:;" \o "2.MATCH查询)
  + [3.RETURN返回](https://cloud.tencent.com/developer/article/javascript:;" \o "3.RETURN返回)
  + [4.关系基础](https://cloud.tencent.com/developer/article/javascript:;" \o "4.关系基础)
  + [5.WHERE子句](https://cloud.tencent.com/developer/article/javascript:;" \o "5.WHERE子句)
  + [6.DELETE删除](https://cloud.tencent.com/developer/article/javascript:;" \o "6.DELETE删除)
  + [7.REMOVE删除](https://cloud.tencent.com/developer/article/javascript:;" \o "7.REMOVE删除)
  + [8.SET子句](https://cloud.tencent.com/developer/article/javascript:;" \o "8.SET子句)
  + [9.ORDER BY排序](https://cloud.tencent.com/developer/article/javascript:;" \o "9.ORDER BY排序)
  + [10.UNION子句](https://cloud.tencent.com/developer/article/javascript:;" \o "10.UNION子句)
  + [11.LIMIT和SKIP子句](https://cloud.tencent.com/developer/article/javascript:;" \o "11.LIMIT和SKIP子句)
  + [12.MERGE命令](https://cloud.tencent.com/developer/article/javascript:;" \o "12.MERGE命令)
  + [13.NULL值](https://cloud.tencent.com/developer/article/javascript:;" \o "13.NULL值)
  + [14.IN操作符](https://cloud.tencent.com/developer/article/javascript:;" \o "14.IN操作符)
  + [15.INDEX索引](https://cloud.tencent.com/developer/article/javascript:;" \o "15.INDEX索引)
  + [16.UNIQUE约束](https://cloud.tencent.com/developer/article/javascript:;" \o "16.UNIQUE约束)
  + [17.DISTINCT独特](https://cloud.tencent.com/developer/article/javascript:;" \o "17.DISTINCT独特)
* [第五章：解释](https://cloud.tencent.com/developer/article/javascript:;" \o "第五章：解释)
  + [1.图形字体](https://cloud.tencent.com/developer/article/javascript:;" \o "1.图形字体)
  + [2.ID属性](https://cloud.tencent.com/developer/article/javascript:;" \o "2.ID属性)
  + [3.Caption标题](https://cloud.tencent.com/developer/article/javascript:;" \o "3.Caption标题)
* [第六章：函数](https://cloud.tencent.com/developer/article/javascript:;" \o "第六章：函数)
  + [1.字符串函数](https://cloud.tencent.com/developer/article/javascript:;" \o "1.字符串函数)
  + [2.AGGREGATION聚合](https://cloud.tencent.com/developer/article/javascript:;" \o "2.AGGREGATION聚合)
  + [3.关系函数](https://cloud.tencent.com/developer/article/javascript:;" \o "3.关系函数)
* [第七章：管理员](https://cloud.tencent.com/developer/article/javascript:;" \o "第七章：管理员)
  + [1.数据库备份](https://cloud.tencent.com/developer/article/javascript:;" \o "1.数据库备份)
  + [2.数据库恢复](https://cloud.tencent.com/developer/article/javascript:;" \o "2.数据库恢复)
* [第八章：Spring Data Neo4j](https://cloud.tencent.com/developer/article/javascript:;" \o "第八章：Spring Data Neo4j)
  + [1.简单介绍](https://cloud.tencent.com/developer/article/javascript:;" \o "1.简单介绍)
  + [2.新建项目](https://cloud.tencent.com/developer/article/javascript:;" \o "2.新建项目)
  + [3.节点与关系](https://cloud.tencent.com/developer/article/javascript:;" \o "3.节点与关系)
  + [4.Repository](https://cloud.tencent.com/developer/article/javascript:;" \o "4.Repository)
  + [5.单元测试](https://cloud.tencent.com/developer/article/javascript:;" \o "5.单元测试)
* [附录 neo4j.conf汉化版](https://cloud.tencent.com/developer/article/javascript:;" \o "附录 neo4j.conf汉化版)
* Create by yster@foxmail.com 2018-7-10
* 我的博客：[https://blog.csdn.net/yueshutong123](https://blog.csdn.net/yueshutong123" \t "https://cloud.tencent.com/developer/article/_blank)
* W3Cschool文档：[https://www.w3cschool.cn/neo4j/neo4j\_features\_advantages.html](https://www.w3cschool.cn/neo4j/neo4j_features_advantages.html" \t "https://cloud.tencent.com/developer/article/_blank)
* neo4j-examples：[https://github.com/neo4j-examples/](https://github.com/neo4j-examples/" \t "https://cloud.tencent.com/developer/article/_blank)

## **第一章：介绍**

### **Neo4j是什么**

[Neo4j](https://baike.baidu.com/item/Neo4j" \t "https://cloud.tencent.com/developer/article/_blank)是一个高性能的,NOSQL图形数据库，它将结构化数据存储在网络上而不是表中。它是一个[嵌入式](https://baike.baidu.com/item/%E5%B5%8C%E5%85%A5%E5%BC%8F" \t "https://cloud.tencent.com/developer/article/_blank)的、基于[磁盘](https://baike.baidu.com/item/%E7%A3%81%E7%9B%98" \t "https://cloud.tencent.com/developer/article/_blank)的、具备完全的事务特性的Java持久化引擎，但是它将结构化数据存储在网络(从数学角度叫做图)上而不是表中。Neo4j也可以被看作是一个高性能的图引擎，该引擎具有成熟数据库的所有特性。程序员工作在一个面向对象的、灵活的网络结构下而不是严格、静态的表中——但是他们可以享受到具备完全的事务特性、企业级的数据库的所有好处。

### **Neo4j的特点**

* SQL就像简单的查询语言Neo4j CQL
* 它遵循属性图数据模型
* 它通过使用Apache Lucence支持索引
* 它支持UNIQUE约束
* 它它包含一个用于执行CQL命令的UI：Neo4j数据浏览器
* 它支持完整的ACID（原子性，一致性，隔离性和持久性）规则
* 它采用原生图形库与本地GPE（图形处理引擎）
* 它支持查询的数据导出到JSON和XLS格式
* 它提供了REST API，可以被任何编程语言（如Java，Spring，Scala等）访问
* 它提供了可以通过任何UI MVC框架（如Node JS）访问的Java脚本
* 它支持两种Java API：Cypher API和Native Java API来开发Java应用程序

### **Neo4j的优点**

* 它很容易表示连接的数据
* 检索/遍历/导航更多的连接数据是非常容易和快速的
* 它非常容易地表示半结构化数据
* Neo4j CQL查询语言命令是人性化的可读格式，非常容易学习
* 它使用简单而强大的数据模型
* 它不需要复杂的连接来检索连接的/相关的数据，因为它很容易检索它的相邻节点或关系细节没有连接或索引

## **第二章：安装**

### **1.环境**

Centos 7.4

neo4j-community-3.4.1.tar.gz

### **2.下载**

下载地址 [https://neo4j.com/download/other-releases/](https://neo4j.com/download/other-releases/" \t "https://cloud.tencent.com/developer/article/_blank)

下载

wget https://neo4j.com/artifact.php?name=neo4j-community-3.4.1-unix.tar.gz

解压

tar -zxvf neo4j-community-3.4.1.tar.gz

### **3.开启远程访问**

****一、对于3.0以前的版本****

在安装目录的 $NEO4J\_HOME/conf/neo4j.conf 文件内，找到下面一行，将注释#号去掉就可以了 #dbms.connector.https.address=localhost:7473 改为 dbms.connector.https.address=0.0.0.0:7473 这样，远程其他电脑可以用本机的IP或者域名后面跟上7474 端口就能打开web界面了 如： https://:7473

当然，你的操作系统的防火墙也要确保开放了7474端口才行，防火墙怎样开放请自行针对自己的操作系统查找文档

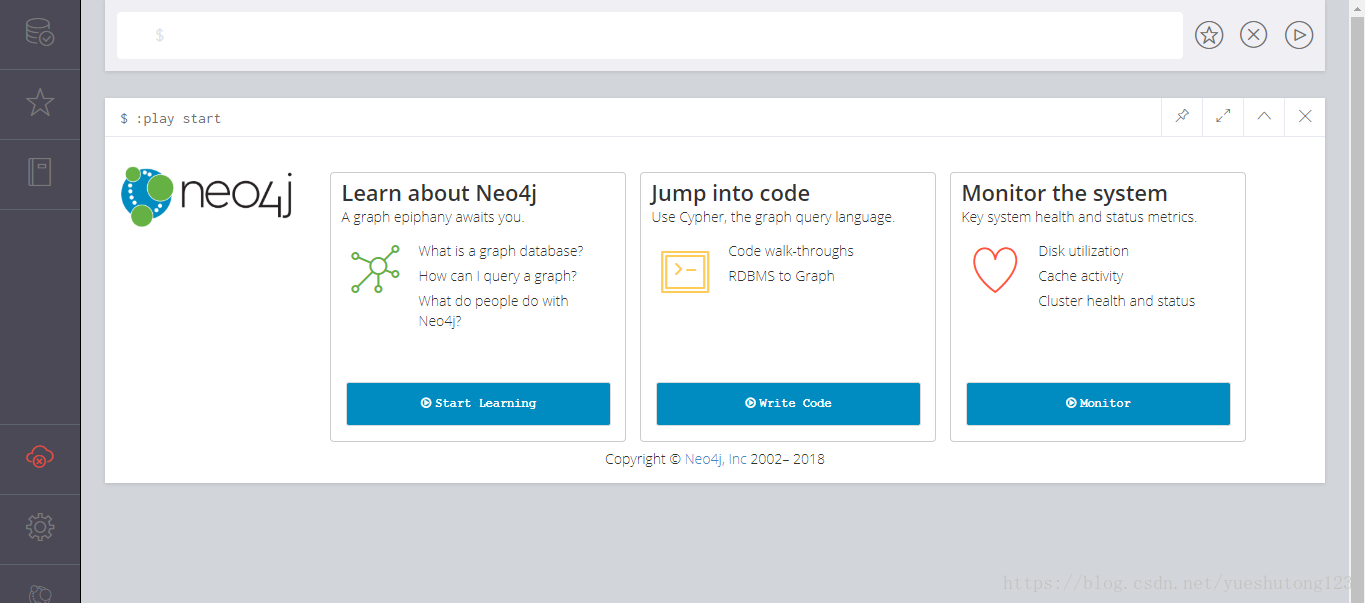
****二、对于3.1及以后的版本****

在安装目录的 $NEO4J\_HOME/conf/neo4j.conf 文件内，找到下面一行，将注释#号去掉就可以了 dbms.connectors.default\_listen\_address=0.0.0.0

### **4.测试**

在bin目录下，执行命令：./neo4j start启动，其他命令 { console | start | stop | restart | status }

访问http://IP地址:7474/, 出现下图即代表安装成功，顶部的$输入框用来执行下面的CQL语句。



## **第三章：CQL**

### **1.CQL简介**

CQL代表Cypher查询语言。 像Oracle数据库具有查询语言SQL，Neo4j具有CQL作为查询语言。

****Neo4j CQL -****

* 它是Neo4j图形数据库的查询语言。
* 它是一种声明性模式匹配语言
* 它遵循SQL语法。
* 它的语法是非常简单且人性化、可读的格式。

****如Oracle SQL -****

* Neo4j CQL 已命令来执行数据库操作。
* Neo4j CQL 支持多个子句像在哪里，顺序等，以非常简单的方式编写非常复杂的查询。
* NNeo4j CQL 支持一些功能，如字符串，Aggregation.In 加入他们，它还支持一些关系功能。

### **2.Neo4j CQL命令/条款**

常用的Neo4j CQL命令/条款如下：

| S.No. | CQL命令/条 | 用法 |
| --- | --- | --- |
| 1。 | CREATE 创建 | 创建节点，关系和属性 |
| 2。 | MATCH 匹配 | 检索有关节点，关系和属性数据 |
| 3。 | RETURN 返回 | 返回查询结果 |
| 4。 | WHERE 哪里 | 提供条件过滤检索数据 |
| 5。 | DELETE 删除 | 删除节点和关系 |
| 6。 | REMOVE 移除 | 删除节点和关系的属性 |
| 7。 | ORDER BY以…排序 | 排序检索数据 |
| 8。 | SET 组 | 添加或更新标签 |

### **3.Neo4j CQL 函数**

以下是常用的Neo4j CQL函数：

| S.No. | 定制列表功能 | 用法 |
| --- | --- | --- |
| 1。 | String 字符串 | 它们用于使用String字面量。 |
| 2。 | Aggregation 聚合 | 它们用于对CQL查询结果执行一些聚合操作。 |
| 3。 | Relationship 关系 | 他们用于获取关系的细节，如startnode，endnode等。 |

我们将在后面的章节中详细讨论所有Neo4j CQL命令，子句和函数语法，用法和示例。

### **4.Neo4j CQL数据类型**

这些数据类型与Java语言类似。 它们用于定义节点或关系的属性

Neo4j CQL支持以下数据类型：

| S.No. | CQL数据类型 | 用法 |
| --- | --- | --- |
| 1. | boolean | 用于表示布尔文字：true，false。 |
| 2. | byte | 用于表示8位整数。 |
| 3. | short | 用于表示16位整数。 |
| 4. | int | 用于表示32位整数。 |
| 5. | long | 用于表示64位整数。 |
| 6. | float | I用于表示32位浮点数。 |
| 7. | double | 用于表示64位浮点数。 |
| 8. | char | 用于表示16位字符。 |
| 9. | String | 用于表示字符串。 |

## **第四章：命令**

### **1.CREATE创建**

****Neo4j CQL创建一个没有属性的节点****

CREATE (<node-name>:<label-name>)

语法说明

规范说法是节点标签名称，其实相当于Mysql数据库中的表名，而是节点名称，其实代指创建的此行数据。

****示例****

CREATE (emp:Employee)

或者

CREATE (:Employee)

****Neo4j CQL创建具有属性的节点****

Neo4j CQL“CREATE”命令用于创建带有属性的节点。 它创建一个具有一些属性（键值对）的节点来存储数据。

CREATE (

<node-name>:<label-name>

{

<key>:<Value>

........

<n-key>:<n-Value>

})

****示例****

CREATE (dept:Dept { deptno:10,dname:"Accounting",location:"Hyderabad" })

****创建多个标签到节点****

语法：

CREATE (<node-name>:<label-name1>:<label-name2>.....:<label-namen>)

****示例****

CREATE (m:Movie:Cinema:Film:Picture)

### **2.MATCH查询**

Neo4j CQL MATCH命令用于

* 从数据库获取有关节点和属性的数据
* 从数据库获取有关节点，关系和属性的数据

MATCH命令语法：

MATCH (

<node-name>:<label-name>)

****示例****

MATCH (dept:Dept)

但是执行后会报错：

Neo.ClientError.Statement.SyntaxError:

Query cannot conclude with MATCH (must be RETURN or an update clause) (line 1, column 1 (offset: 0))

如果你观察到错误消息，它告诉我们，我们可以使用MATCH命令与RETURN子句或UPDATA子句。

### **3.RETURN返回**

Neo4j CQL RETURN子句用于 -

* 检索节点的某些属性
* 检索节点的所有属性
* 检索节点和关联关系的某些属性
* 检索节点和关联关系的所有属性

RETURN命令语法：

RETURN

<node-name>.<property1-name>,

........

<node-name>.<propertyn-name>

****示例****

MATCH (e:Employee) RETURN e

或

MATCH (dept: Dept)

RETURN dept.deptno,dept.dname,dept.location

### **4.关系基础**

Neo4j图数据库遵循属性图模型来存储和管理其数据。

根据属性图模型，关系应该是定向的。 否则，Neo4j将抛出一个错误消息。

基于方向性，Neo4j关系被分为两种主要类型。

* 单向关系
* 双向关系

****使用新节点创建关系****

****示例****

CREATE (e:Employee)-[r:DemoRelation]->(c:Employee)

这句会创建节点e，节点c，以及e -> c的关系r，这里需要注意方向，比如双向是

CREATE (e:Employee)<-[r:DemoRelation]->(c:Employee)

****使用已知节点创建带属性的关系：****

MATCH (<node1-label-name>:<node1-name>),(<node2-label-name>:<node2-name>)CREATE

(<node1-label-name>)-[<relationship-label-name>:<relationship-name>

{<define-properties-list>}]->(<node2-label-name>)

RETURN <relationship-label-name>

还是一系列键值对

****示例****

MATCH (cust:Customer),(cc:CreditCard) CREATE (cust)-[r:DO\_SHOPPING\_WITH{shopdate:"12/12/2014",price:55000}]->(cc)

RETURN r

****检索关系节点的详细信息：****

MATCH (<node1-label-name>)-[<relationship-label-name>:<relationship-name>]->(<node2-label-name>)

RETURN <relationship-label-name>

****示例****

MATCH (cust)-[r:DO\_SHOPPING\_WITH]->(cc)

RETURN cust,cc

### **5.WHERE子句**

像SQL一样，Neo4j CQL在CQL MATCH命令中提供了WHERE子句来过滤MATCH查询的结果。

****简单WHERE子句语法****

WHERE <property-name> <comparison-operator> <value>

语法说明：

| S.No. | 语法元素 | 描述 |
| --- | --- | --- |
| 1 | WHERE | 它是一个Neo4j CQL关键字。 |
| 2 | <属性名称> | 它是节点或关系的属性名称。 |
| 3 | <比较运算符> | 它是Neo4j CQL比较运算符之一。 |
| 4 | <值> | 它是一个字面值，如数字文字，字符串文字等。 |

****Neo4j CQL中的比较运算符****

Neo4j 支持以下的比较运算符，在 Neo4j CQL WHERE 子句中使用来支持条件

| S.No. | 布尔运算符 | 描述 |
| --- | --- | --- |
| 1. | = | 它是Neo4j CQL“等于”运算符。 |
| 2. | <> | 它是一个Neo4j CQL“不等于”运算符。 |
| 3. | < | 它是一个Neo4j CQL“小于”运算符。 |
| 4. | > | 它是一个Neo4j CQL“大于”运算符。 |
| 5. | <= | 它是一个Neo4j CQL“小于或等于”运算符。 |
| 6. | = | 它是一个Neo4j CQL“大于或等于”运算符。 |

我们可以使用布尔运算符在同一命令上放置多个条件。

****Neo4j CQL中的布尔运算符****

Neo4j支持以下布尔运算符在Neo4j CQL WHERE子句中使用以支持多个条件。

| S.No. | 布尔运算符 | 描述 |
| --- | --- | --- |
| 1 | AND | 它是一个支持AND操作的Neo4j CQL关键字。 |
| 2 | OR | 它是一个Neo4j CQL关键字来支持OR操作。 |
| 3 | NOT | 它是一个Neo4j CQL关键字支持NOT操作。 |
| 4 | XOR | 它是一个支持XOR操作的Neo4j CQL关键字。 |

****示例****

MATCH (emp:Employee)

WHERE emp.name = 'Abc' OR emp.name = 'Xyz'

RETURN emp

利用WHERE创建指定关系节点：

MATCH (cust:Customer),(cc:CreditCard)

WHERE cust.id = "1001" AND cc.id= "5001" CREATE (cust)-[r:DO\_SHOPPING\_WITH{shopdate:"12/12/2014",price:55000}]->(cc)

RETURN r

有必要补充一下，可以不使用WHERE达到WHERE的一些效果，比如

MATCH p=(m:Bot{id:123})<-[:BotRelation]->(:Bot) RETURN p

### **6.DELETE删除**

Neo4j使用CQL DELETE子句

* 删除节点。
* 删除节点及相关节点和关系。

****DELETE节点子句语法****

DELETE <node-name-list>

****示例****

MATCH (e: Employee) DELETE e

****DELETE节点和关系子句语法****

DELETE <node1-name>,<node2-name>,<relationship-name>

****示例****

MATCH (cc: CreditCard)-[rel]-(c:Customer)

DELETE cc,c,rel

### **7.REMOVE删除**

有时基于我们的客户端要求，我们需要向现有节点或关系添加或删除属性。

我们使用Neo4j CQL SET子句向现有节点或关系添加新属性。

我们使用Neo4j CQL REMOVE子句来删除节点或关系的现有属性。

Neo4j CQL REMOVE命令用于

* 删除节点或关系的标签
* 删除节点或关系的属性

Neo4j CQL DELETE和REMOVE命令之间的主要区别 -

* DELETE操作用于删除节点和关联关系。
* REMOVE操作用于删除标签和属性。

Neo4j CQL DELETE和REMOVE命令之间的相似性 -

* 这两个命令不应单独使用。
* 两个命令都应该与MATCH命令一起使用。

****1.REMOVE属性子句语法****

REMOVE <node-name>.<property1-name>,<node-name>.<property2-name>

语法说明：

| S.No. | 语法元素 | 描述 |
| --- | --- | --- |
| 1。 |  | 它是节点的名称。 |
| 2。 |  | 它是节点的属性名称。 |

****示例****



这里我们可以观察到DebitCard节点包含6个属性。

在数据浏览器上键入以下命令删除cvv属性

MATCH (dc:DebitCard)

REMOVE dc.cvv

RETURN dc

****2.REMOVE一个Label子句语法：****

REMOVE <label-name-list>

| S.No. | 语法元素 | 描述 |
| --- | --- | --- |
| 1. | REMOVE | 它是一个Neo4j CQL关键字。 |
| 2. |  | 它是一个标签列表，用于永久性地从节点或关系中删除它。 |

语法

<node-name>:<label2-name>, .... <node-name>:<labeln-name>

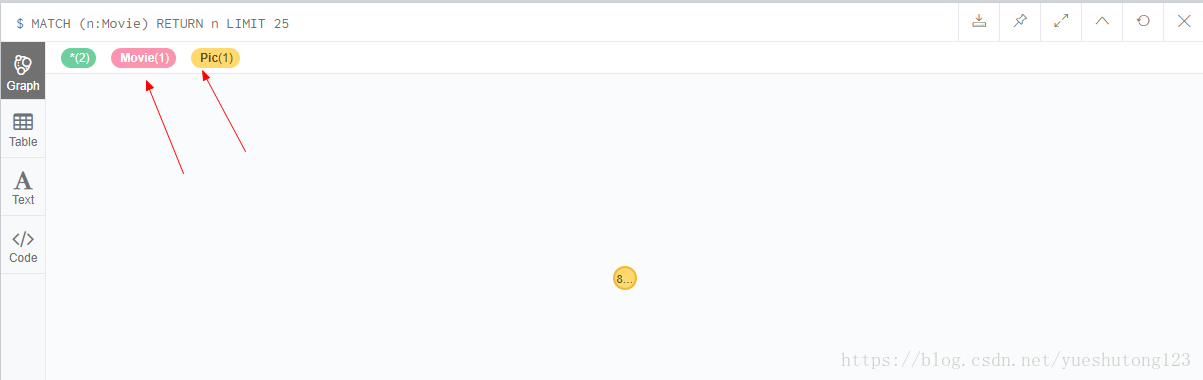
****示例****

1.我们创建一个含有两个标签的节点：

CREATE (m:Movie:Pic)

2.查询该节点

MATCH (n:Movie) RETURN n

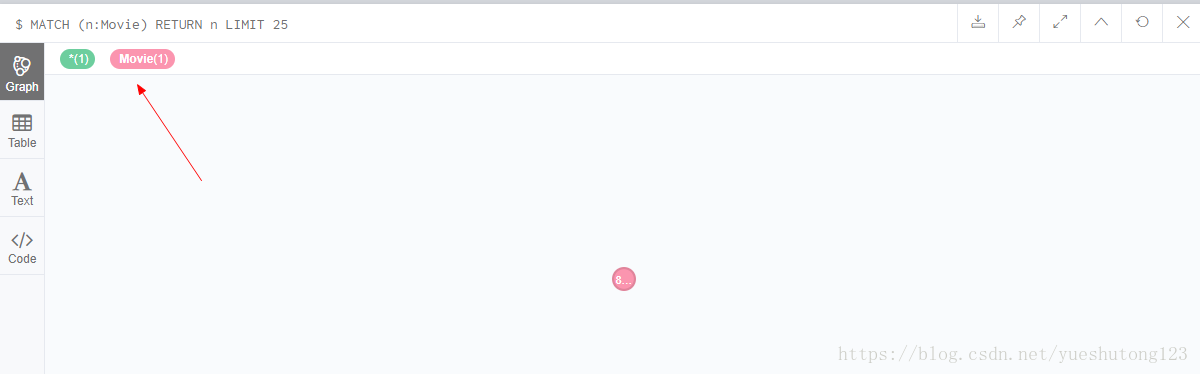


3.删除标签

MATCH (m:Movie)

REMOVE m:Pic

4.再次查询



### **8.SET子句**

有时，根据我们的客户端要求，我们需要向现有节点或关系添加新属性。

要做到这一点，Neo4j CQL提供了一个SET子句。

Neo4j CQL已提供SET子句来执行以下操作。

* 向现有节点或关系添加新属性
* 添加或更新属性值

****SET子句语法****

SET <node-label-name>.<property1-name>,...<node-laben-name>.<propertyn-name>

语法说明：

| S.No. | 语法元素 | 描述 |
| --- | --- | --- |
| 1 | <节点标签名称> | 这是一个节点的标签名称。 |
| 2 | <属性名称> | 它是一个节点的属性名。 |

****示例****

MATCH (dc:DebitCard)

SET dc.atm\_pin = 3456

RETURN dc

### **9.ORDER BY排序**

****Neo4j CQL ORDER BY子句****

Neo4j CQL在MATCH命令中提供了“ORDER BY”子句，对MATCH查询返回的结果进行排序。

我们可以按升序或降序对行进行排序。

默认情况下，它按升序对行进行排序。 如果我们要按降序对它们进行排序，我们需要使用DESC子句。

****ORDER BY子句语法****

ORDER BY <property-name-list> [DESC]

****语法：****

<node-label-name>.<property1-name>,<node-label-name>.<property2-name>, .... <node-label-name>.<propertyn-name>

| S.No. | 语法元素 | 描述 |
| --- | --- | --- |
| 1。 |  | 它是节点的标签名称。 |
| 2。 |  | 它是节点的属性名称。 |

****示例****

MATCH (emp:Employee)

RETURN emp.empid,emp.name,emp.salary,emp.deptno

ORDER BY emp.name

### **10.UNION子句**

与SQL一样，Neo4j CQL有两个子句，将两个不同的结果合并成一组结果

* UNION
* UNION ALL

****UNION子句****

它将两组结果中的公共行组合并返回到一组结果中。 它不从两个节点返回重复的行。

限制：

结果列类型和来自两组结果的名称必须匹配，这意味着列名称应该相同，列的数据类型应该相同。

****UNION子句语法****

<MATCH Command1>

UNION<MATCH Command2>

| S.No. | 语法元素 | 描述 |
| --- | --- | --- |
| 1。 |  | 它是CQL MATCH命令，由UNION子句使用。 |
| 2。 |  | 它是CQL MATCH命令两个由UNION子句使用。 |
| 3。 | UNION | 它是UNION子句的Neo4j CQL关键字。 |

****注意 -****

如果这两个查询不返回相同的列名和数据类型，那么它抛出一个错误。

****示例****

MATCH (cc:CreditCard) RETURN cc.id,cc.number

UNIONMATCH (dc:DebitCard) RETURN dc.id,dc.number

****UNION ALL子句****

它结合并返回两个结果集的所有行成一个单一的结果集。它还返回由两个节点重复行。

****限制****

结果列类型，并从两个结果集的名字必须匹配，这意味着列名称应该是相同的，列的数据类型应该是相同的。

****UNION ALL子句语法****

<MATCH Command1>

UNION ALL<MATCH Command2>

****示例****

MATCH (cc:CreditCard) RETURN cc.id,cc.number

UNION ALLMATCH (dc:DebitCard) RETURN dc.id,dc.number

### **11.LIMIT和SKIP子句**

Neo4j CQL已提供LIMIT子句和SKIP来过滤或限制查询返回的行数。

****简单来说：**LIMIT**返回前几行，**SKIP**返回后几行。****

LIMIT ****示例****

MATCH (emp:Employee)

RETURN emp

LIMIT 2

它只返回Top的两个结果，因为我们定义了limit = 2。这意味着前两行。

SKIP****示例****

MATCH (emp:Employee)

RETURN emp

SKIP 2

它只返回来自Bottom的两个结果，因为我们定义了skip = 2。这意味着最后两行。

### **12.MERGE命令**

Neo4j使用CQL MERGE命令 -

* 创建节点，关系和属性
* 为从数据库检索数据

MERGE命令是CREATE命令和MATCH命令的组合。

MERGE = CREATE + MATCH

Neo4j CQL MERGE命令在图中搜索给定模式，如果存在，则返回结果

如果它不存在于图中，则它创建新的节点/关系并返回结果。

****Neo4j CQL MERGE语法****

MERGE (<node-name>:<label-name>{

<key>:<1-Value>

.....

<n-key>:<n-Value>})

****注意 -****

Neo4j CQL MERGE命令语法与CQL CREATE命令类似。

我们将使用这两个命令执行以下操作 -

* 创建具有一个属性的配置文件节点：Id，名称
* 创建具有相同属性的同一个Profile节点：Id，Name
* 检索所有Profile节点详细信息并观察结果

我们将使用CREATE命令执行这些操作：

MERGE (gp2:GoogleProfile2{ Id: 201402,Name:"Nokia"})

MERGE (gp2:GoogleProfile2{ Id: 201402,Name:"Nokia"})

MATCH (gp1:GoogleProfile1)

RETURN gp1.Id,gp1.Name

如果我们观察上面的查询结果，它只显示一行，因为CQL MERGE命令检查该节点在数据库中是否可用。 如果它不存在，它创建新节点。 否则，它不创建新的。

通过观察这些结果，我们可以说，CQL MERGE命令将新的节点添加到数据库，只有当它不存在。

### **13.NULL值**

Neo4j CQL将空值视为对节点或关系的属性的缺失值或未定义值。

当我们创建一个具有现有节点标签名称但未指定其属性值的节点时，它将创建一个具有NULL属性值的新节点。

让我们用一个例子来看这个。

MATCH (e:Employee)

WHERE e.id IS NOT NULL

RETURN e.id,e.name,e.sal,e.deptno

提供了一个WHERE子句来过滤该行，即Id属性不应该包含NULL值。

MATCH (e:Employee)

WHERE e.id IS NULL

RETURN e.id,e.name,e.sal,e.deptno

这里我们使用IS操作符来仅返回NULL行。

### **14.IN操作符**

与SQL一样，Neo4j CQL提供了一个IN运算符，以便为CQL命令提供值的集合。

****IN操作符语法****

IN[<Collection-of-values>]

它是由逗号运算符分隔的值的集合。

****示例****

MATCH (e:Employee)

WHERE e.id IN [123,124]

RETURN e.id,e.name,e.sal,e.deptno

### **15.INDEX索引**

Neo4j SQL支持节点或关系属性上的索引，以提高应用程序的性能。

我们可以为具有相同标签名称的所有节点的属性创建索引。

我们可以在MATCH或WHERE或IN运算符上使用这些索引列来改进CQL Command的执行。

****Neo4J索引操作****

* Create Index 创建索引
* Drop Index 丢弃索引

我们将在本章中用示例来讨论这些操作。

****创建索引的语法：****

CREATE INDEX ON :<label\_name> (<property\_name>)

****注意：-****

冒号（:)运算符用于引用节点或关系标签名称。

上述语法描述它在节点或关系的的上创建一个新索引。

****示例****

CREATE INDEX ON :Customer (name)

****删除索引的语法：****

DROP INDEX ON :<label\_name> (<property\_name>)

****示例****

DROP INDEX ON :Customer (name)

### **16.UNIQUE约束**

在Neo4j数据库中，CQL CREATE命令始终创建新的节点或关系，这意味着即使您使用相同的值，它也会插入一个新行。 根据我们对某些节点或关系的应用需求，我们必须避免这种重复。 然后我们不能直接得到这个。 我们应该使用一些数据库约束来创建节点或关系的一个或多个属性的规则。

像SQL一样，Neo4j数据库也支持对NODE或Relationship的属性的UNIQUE约束

****UNIQUE约束的优点****

* 避免重复记录。
* 强制执行数据完整性规则

****创建唯一约束语法****

CREATE CONSTRAINT ON (<label\_name>)

ASSERT <property\_name> IS UNIQUE

语法说明：

| S.No. | 语法元素 | 描述 |
| --- | --- | --- |
| 1。 | CREATE CONSTRAINT ON | 它是一个Neo4j CQL关键字。 |
| 2。 |  | 它是节点或关系的标签名称。 |
| 3。 | ASSERT | 它是一个Neo4j CQL关键字。 |
| 4。 |  | 它是节点或关系的属性名称。 |
| 5。 | IS UNIQUE | 它是一个Neo4j CQL关键字，通知Neo4j数据库服务器创建一个唯一约束。 |

****注意：-****

上述语法描述了只需要 节点或关系就可以创造一个独特的约束。

****示例****

CREATE CONSTRAINT ON (cc:CreditCard)

ASSERT cc.number IS UNIQUE

****注意****

如果创建约束时节点属性有重复值，Neo4j DB服务器将会抛出一个错误，表示无法创建。

****删除UNIQUE约束语法：****

DROP CONSTRAINT ON (<label\_name>)

ASSERT <property\_name> IS UNIQUE

****示例****

DROP CONSTRAINT ON (cc:CreditCard)

ASSERT cc.number IS UNIQUE

### **17.DISTINCT独特**

这个函数的用法就像SQL中的distinct关键字，返回的是所有不同值。

****示例****



MATCH (n:Movie) RETURN Distinct(n.name)

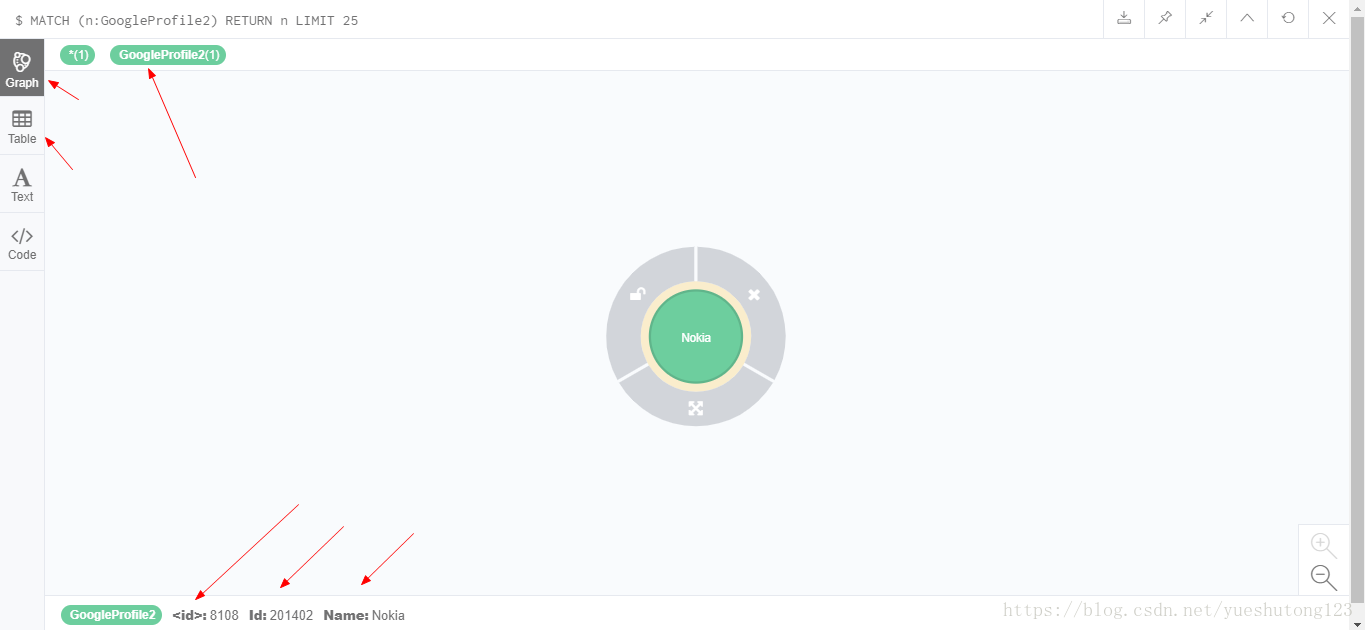
返回的是

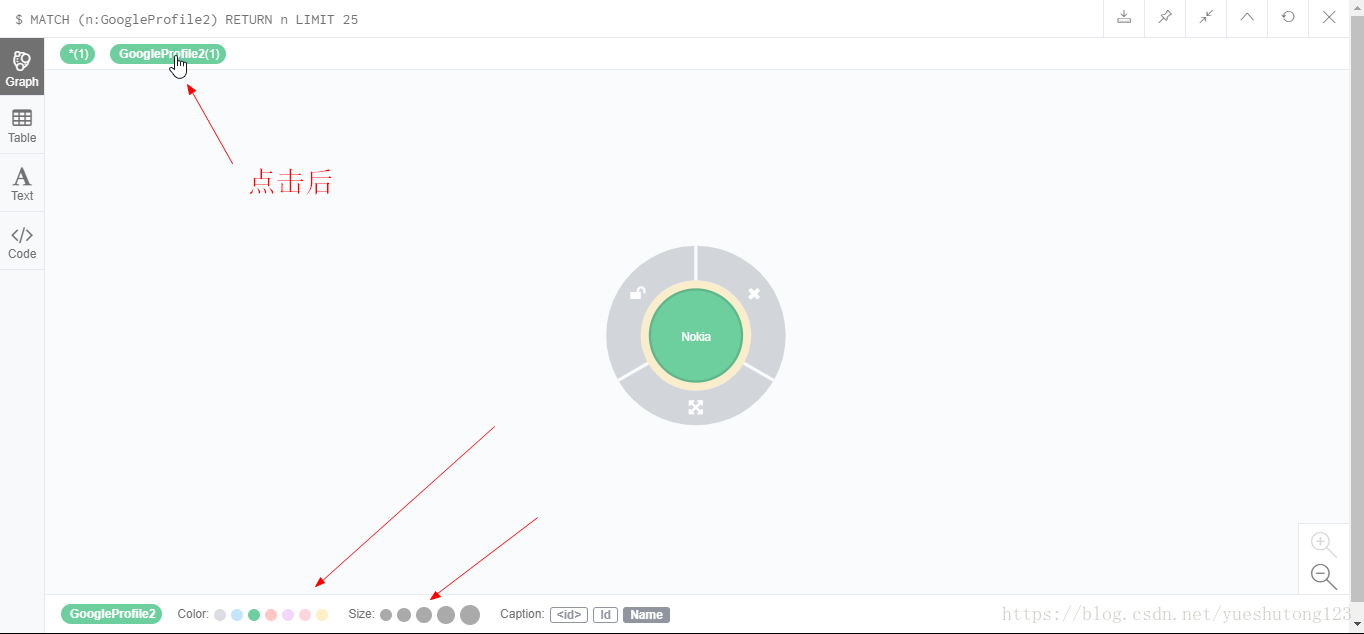


## **第五章：解释**

### **1.图形字体**

关于Neo4j提供的图形浏览器，我们可以从其中查看节点的属性，或者改变其中的节点的大小颜色。





### **2.ID属性**

在Neo4j中，“Id”是节点和关系的默认内部属性。 这意味着，当我们创建一个新的节点或关系时，Neo4j数据库服务器将为内部使用分配一个数字。 它会自动递增。

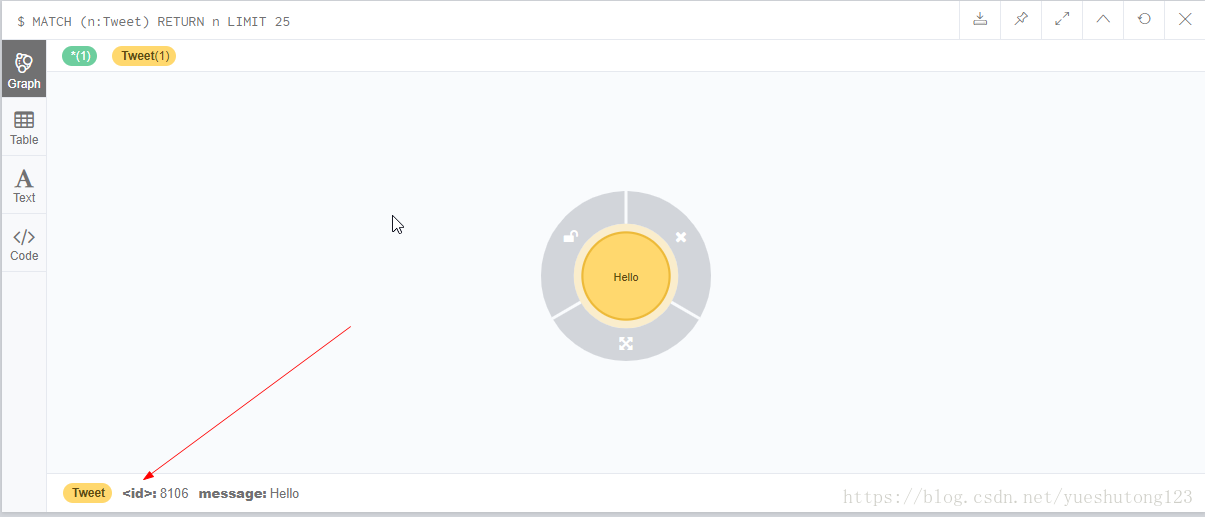
我们从一个例子去看：

新增一个节点

CREATE (tweet:Tweet{message:"Hello"})

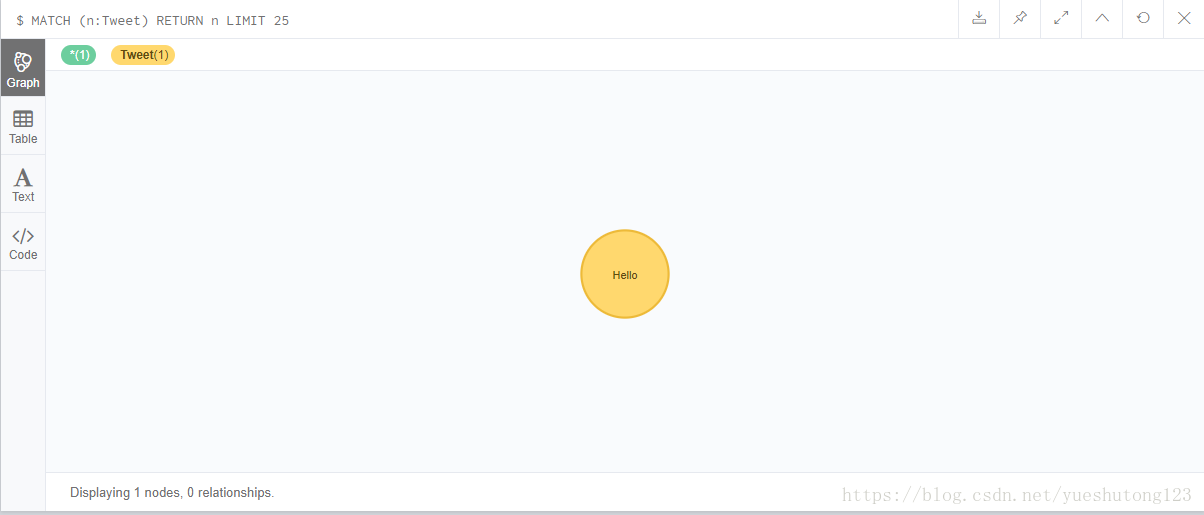
查看该节点

MATCH (n:Tweet) RETURN n

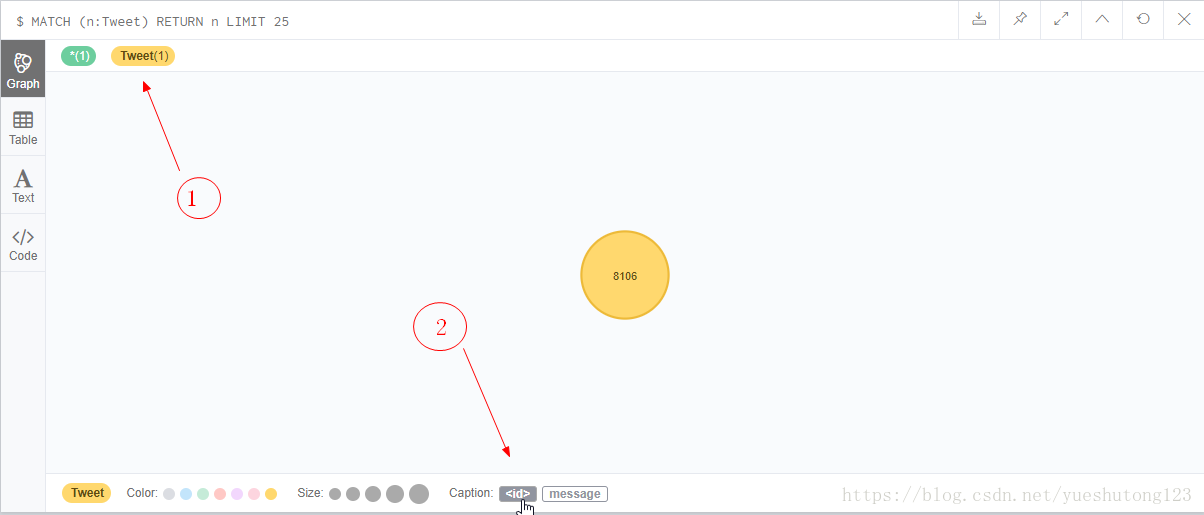


### **3.Caption标题**

所谓的Caption标题，就是更改Neo4j浏览器的节点显示的文字（圆圈内部）。比如



我们点击下图所示：



圆圈内部变为了id值。

## **第六章：函数**

### **1.字符串函数**

与SQL一样，Neo4J CQL提供了一组String函数，用于在CQL查询中获取所需的结果。

这里我们将讨论一些重要的和经常使用的功能。

****字符串函数列表****

| S.No. | 功能 | 描述 |
| --- | --- | --- |
| 1。 | UPPER | 它用于将所有字母更改为大写字母。 |
| 2。 | LOWER | 它用于将所有字母改为小写字母。 |
| 3。 | SUBSTRING | 它用于获取给定String的子字符串。 |
| 4。 | REPLACE | 它用于替换一个字符串的子字符串。 |

****注意：****所有CQL函数应使用“（）”括号。

现在我们将通过示例详细讨论每个Neo4J CQL字符串函数

1.****UPPER****

它需要一个字符串作为输入并转换为大写字母。 所有CQL函数应使用“（）”括号。

****函数语法****

UPPER (<input-string>)

****注意：-****

可以是来自Neo4J数据库的节点或关系的属性名称。

****示例****

MATCH (e:Employee)

RETURN e.id,UPPER(e.name),e.sal,e.deptno

2.****LOWER****

它需要一个字符串作为输入并转换为小写字母。 所有CQL函数应使用“（）”括号。

****函数语法****

LOWER (<input-string>)

****注意：-****

可以是来自Neo4J数据库的节点或关系的属性名称

MATCH (e:Employee)

RETURN e.id,LOWER(e.name),e.sal,e.deptno

3.****SUBSTRING****

它接受一个字符串作为输入和两个索引：一个是索引的开始，另一个是索引的结束，并返回从StartInded到EndIndex-1的子字符串。 所有CQL函数应使用“（）”括号。

****函数的语法****

SUBSTRING(<input-string>,<startIndex> ,<endIndex>)

****注意：-****

在Neo4J CQL中，如果一个字符串包含n个字母，则它的长度为n，索引从0开始，到n-1结束。

是SUBSTRING函数的索引值。

是可选的。 如果我们省略它，那么它返回给定字符串的子串从startIndex到字符串的结尾。

****示例****

MATCH (e:Employee)

RETURN e.id,SUBSTRING(e.name,0,2),e.sal,e.deptno

### **2.AGGREGATION聚合**

和SQL一样，Neo4j CQL提供了一些在RETURN子句中使用的聚合函数。 它类似于SQL中的GROUP BY子句。

我们可以使用MATCH命令中的RETURN +聚合函数来处理一组节点并返回一些聚合值。

****聚合函数列表****

| S.No. | 聚集功能 | 描述 |
| --- | --- | --- |
| 1。 | COUNT | 它返回由MATCH命令返回的行数。 |
| 2。 | MAX | 它从MATCH命令返回的一组行返回最大值。 |
| 3。 | MIN | 它返回由MATCH命令返回的一组行的最小值。 |
| 4。 | SUM | 它返回由MATCH命令返回的所有行的求和值。 |
| 5。 | AVG | 它返回由MATCH命令返回的所有行的平均值。 |

现在我们将通过示例详细讨论每个Neo4j CQL AGGREGATION函数

****计数****

它从MATCH子句获取结果，并计算结果中出现的行数，并返回该计数值。 所有CQL函数应使用“（）”括号。

****函数语法****

COUNT(<value>)

****注意 -****

可以是\*，节点或关系标签名称或属性名称。

****示例****

MATCH (e:Employee) RETURN COUNT(\*)

****MAX****

它采用一组行和节点或关系的作为输入，并从给定行的give 列中查找最小值。

****函数语法****

MAX(<property-name> )

****MIN****

它采用一组行和节点或关系的作为输入，并从给定行的give 列中查找最小值。

****函数语法****

MIN(<property-name> )

****注意 -****

应该是节点或关系的名称。

让我们用一个例子看看MAX和MIN的功能。

****示例****

MATCH (e:Employee)

RETURN MAX(e.sal),MIN(e.sal)

****AVG****

它采用一组行和节点或关系的作为输入，并从给定行的give 列中查找平均值。

****函数的语法****

AVG(<property-name> )

****SUM****

它采用一组行和节点或关系的作为输入，并从给定行的give 列中查找求和值。

****函数的语法****

SUM(<property-name> )

让我们用一个例子来检查SUM和AVG函数。

MATCH (e:Employee)

RETURN SUM(e.sal),AVG(e.sal)

此命令从数据库中可用的所有Employee节点查找总和平均值.

### **3.关系函数**

Neo4j CQL提供了一组关系函数，以在获取开始节点，结束节点等细节时知道关系的细节。

****关系函数列表****

| S.No. | 功能 | 描述 |
| --- | --- | --- |
| 1。 | STARTNODE | 它用于知道关系的开始节点。 |
| 2。 | ENDNODE | 它用于知道关系的结束节点。 |
| 3。 | ID | 它用于知道关系的ID。 |
| 4。 | TYPE | 它用于知道字符串表示中的一个关系的TYPE。 |

现在我们将通过示例详细讨论每个Neo4j CQL关系函数

****STARTNODE****

它需要一个字符串作为输入并转换为大写字母。 所有CQL函数应使用“（）”括号。

****函数语法****

STARTNODE (<relationship-label-name>)

****注意：-****

可以是来自Neo4j数据库的节点或关系的属性名称。

****示例****

MATCH (a)-[movie:ACTION\_MOVIES]->(b)

RETURN STARTNODE(movie)

****ENDNODE****

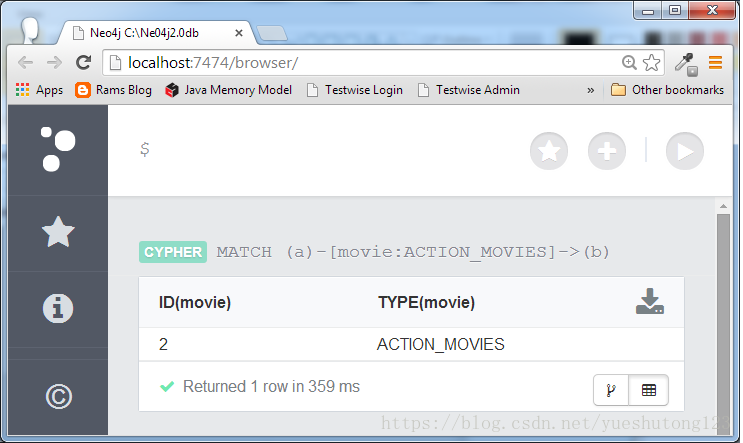
MATCH (a)-[movie:ACTION\_MOVIES]->(b)

RETURN ENDNODE(movie)

****ID TYPE****

MATCH (a)-[movie:ACTION\_MOVIES]->(b)

RETURN ID(movie),TYPE(movie)



## **第七章：管理员**

### **1.数据库备份**

在对Neo4j数据进行备份、还原、迁移的操作时，首先要关闭neo4j;

cd %NEO4J\_HOME%/bin

./neo4j stop

数据备份到文件

./neo4j-admin dump --database=graph.db --to=/home/2018.dump

之后，进行数据还原，将生成的存储文件拷贝到另一个相同版本的环境中。

### **2.数据库恢复**

还原、迁移之前 ，关闭neo4j服务。操作同上；

数据导入：

./neo4j-admin load --from=/home/2016-10-02.dump --database=graph.db --force

重启服务：

./neo4j start

## **第八章：Spring Data Neo4j**

### **1.简单介绍**

Neo4j提供JAVA API以编程方式执行所有数据库操作。

具体Neo4j如何在原生Java程序编程，以及与Spring的集成，本章暂不讨论。

****Spring数据模块的优点：****

* 消除DAO层中的boiler plate代码
* DAO层中的工件少
* 易于开发和维护
* 改进开发过程

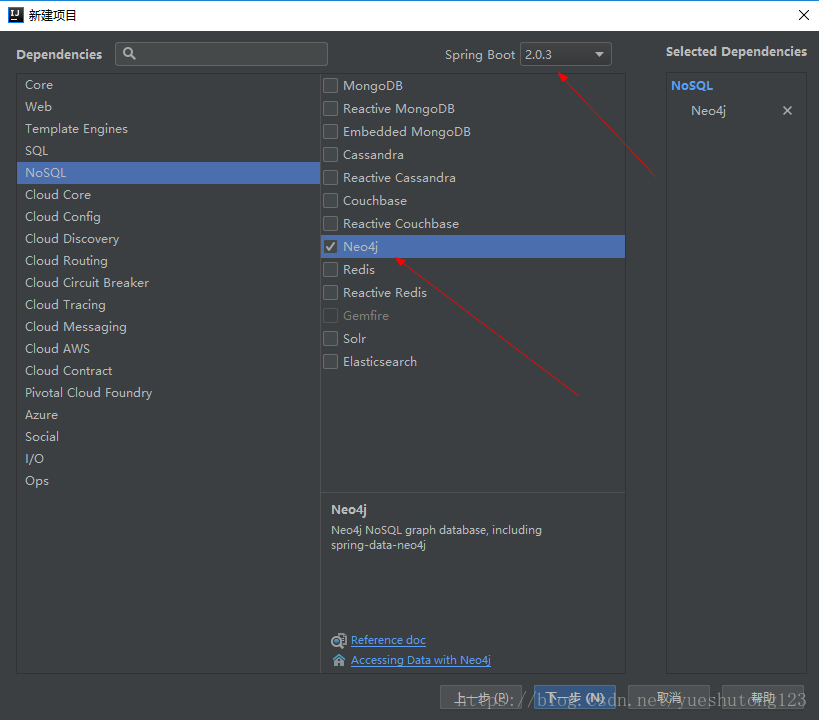
****Spring数据模块功能：****

* 支持基于XML的实体映射
* 支持基于注释的实体映射
* 支持分页
* 支持事务
* 更少的DAO层工件 - 实现存储库

Spring DATA Neo4j模块具有与上述相同的优点和特点。

****接下来，我们将基于Spring Boot在IDEA上开发Neo4j应用程序，需要注意的是Springboot的版本****

### **2.新建项目**



我们选择web和Neo4j两个依赖即可，这里有必要说一下，如果你是使用Spring boot2.0以上，在你创建项目完成后，启动程序会报错：

Caused by: java.lang.ClassNotFoundException: org.neo4j.ogm.drivers.http.driver.HttpDriver

at java.net.URLClassLoader.findClass(URLClassLoader.java:381) ~[na:1.8.0\_111]

at java.lang.ClassLoader.loadClass(ClassLoader.java:424) ~[na:1.8.0\_111]

at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:331) ~[na:1.8.0\_111]

at java.lang.ClassLoader.loadClass(ClassLoader.java:357) ~[na:1.8.0\_111]

at java.lang.Class.forName0(Native Method) ~[na:1.8.0\_111]

at java.lang.Class.forName(Class.java:264) ~[na:1.8.0\_111]

at org.neo4j.ogm.session.SessionFactory.newDriverInstance(SessionFactory.java:92) ~[neo4j-ogm-core-3.1.0.jar:3.1.0]

... 45 common frames omitted

原因是缺少依赖，解决方法是导入缺少的依赖：

<dependency>

<groupId>org.neo4j</groupId>

<artifactId>neo4j-ogm-http-driver</artifactId>

</dependency>

如果你的Spring boot版本为1.5.x，那么你只需要spring-data-neo4j即可：

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-neo4j</artifactId>

</dependency>

### **3.节点与关系**

****新建节点类****，id的属性为Long而不能为long，还需要注意的是在Spring boot1.5中修饰id属性的注释为@GraphId，org.neo4j.ogm.annotation.Id不存在，效果一样，都是Neo4j数据库自动创建的ID值。

@NodeEntity(label = "Bot")public class BotNode {

@Id

@GeneratedValue

private Long id; //id

@Property(name = "name")

private String name;//名

@Property(name = "kind")

private String kind;//类

@Property(name = "weight")

private long weight;//权重

public BotNode() {

}

public BotNode(Long id, String name, String kind, long weight) {

this.id = id;

this.name = name;

this.kind = kind;

this.weight = weight;

}

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getKind() {

return kind;

}

public void setKind(String kind) {

this.kind = kind;

}

public long getWeight() {

return weight;

}

public void setWeight(long weight) {

this.weight = weight;

}

@Override

public String toString() {

return "BotNode{" +

"id=" + id +

", name='" + name + '\'' +

", kind='" + kind + '\'' +

", weight=" + weight +

'}';

}

}

****新建节点关系类****

有必要说明一下， @StartNode 和@EndNode注释的类可以不是同一个类。

@RelationshipEntity(type = "BotRelation")public class BotRelation {

@Id

@GeneratedValue

private Long id;

@StartNode

private BotNode startNode;

@EndNode

private BotNode endNode;

@Property

private String relation;

public BotRelation() {

}

public BotRelation(Long id, BotNode startNode, BotNode endNode, String relation) {

this.id = id;

this.startNode = startNode;

this.endNode = endNode;

this.relation = relation;

}

public String getRelation() {

return relation;

}

public void setRelation(String relation) {

this.relation = relation;

}

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public BotNode getStartNode() {

return startNode;

}

public void setStartNode(BotNode startNode) {

this.startNode = startNode;

}

public BotNode getEndNode() {

return endNode;

}

public void setEndNode(BotNode endNode) {

this.endNode = endNode;

}

@Override

public String toString() {

return "BotRelation{" +

"id=" + id +

", startNode=" + startNode +

", endNode=" + endNode +

", relation='" + relation + '\'' +

'}';

}}

### **4.Repository**

我们只需要使接口继承Neo4jRepository就可以使用该接口提供的一些基础的增删改查方法。

@Repositorypublic interface BotRepository extends Neo4jRepository<BotNode,Long> {

BotNode findAllByName(String name);

}

对于复杂的查询我们可以参照上面讲到的CQL语句执行。

@Repositorypublic interface BotRelationRepository extends Neo4jRepository<BotRelation,Long> {

//返回节点n以及n指向的所有节点与关系

@Query("MATCH p=(n:Bot)-[r:BotRelation]->(m:Bot) WHERE id(n)={0} RETURN p")

List<BotRelation> findAllByBotNode(BotNode botNode);

//返回节点n以及n指向或指向n的所有节点与关系

@Query("MATCH p=(n:Bot)<-[r:BotRelation]->(m:Bot) WHERE m.name={name} RETURN p")

List<BotRelation> findAllBySymptom(@Param("name") String name);

//返回节点n以及n指向或指向n的所有节点以及这些节点间的所有关系

@Query("MATCH p=(n:Bot)<-[r:BotRelation]->(m:Bot)<-[:BotRelation]->(:Bot)<-[:BotRelation]->(n:Bot) WHERE n.name={name} RETURN p")

List<BotRelation> findAllByStartNode(@Param("name") String name);

}

### **5.单元测试**

****保存****

@RunWith(SpringRunner.class)

@SpringBootTestpublic class Neo4jApplicationTests {

@Autowired

MovieRepository movieRepository;

@Test

public void contextLoads() {

movieRepository.save(new Movie("《奥特曼》"));

System.out.println(movieRepository.findAll());

}

}

查看打印：

[Movie{id=8183, name='《奥特曼》'}]

保存成功！

****补充****

如果想保存关系的话

MedicalNode node = new MedicalNode(-1l,"节点","测试");

medicalNodeRepository.save(node);

MedicalNode node1 = new MedicalNode(-1l,"节点","测试");

medicalNodeRepository.save(node1);

medicalRelationRepository.save(new MedicalRelation(-1l,node,node1,"关系"));

****更新****

接下来我们测试更新数据：

@Test

public void updata(){

Movie movie = movieRepository.findAllById(8183l);

movie.setName("《迪迦》");

movieRepository.save(movie);

System.out.println(movieRepository.findAll());

}

执行程序，报错：

java.lang.NullPointerException

我们看到程序执行的CQL语句为：

MATCH (n:`Movie`) WHERE n.`id` = { `id\_0` } WITH n RETURN n, ID(n)

然后我们在Neo4j浏览器控制台执行查询语句：



这是为什么呢？在Neo4j中，根据Id查询节点的语句为：

MATCH (n:Movie) where id(n)=8183 RETURN n

我们修改Repository层的查询方法：

@Repositorypublic interface MovieRepository extends Neo4jRepository<Movie, Long> {

@Query("MATCH (n:Movie) where id(n)={id} RETURN n")

Movie findAllById(@Param("id") Long id);}

再次执行更新程序，结果为：

[Movie{id=8183, name='《迪迦》'}]

更新成功！

换句话说，只要掌握了CQL语句，就基本啥都会了~！ 还有，本书并不代表全部的Neo4j知识。

## **附录 neo4j.conf汉化版**

For more details and a complete list of settings, please see https://neo4j.com/docs/operations-manual/current/reference/configuration-settings/

# 如果想自定义neo4j数据库数据的存储路径，要同时修改dbms.active\_database 和 dbms.directories.data 两项配置，

# 修改配置后，数据会存放在${dbms.directories.data}/databases/${dbms.active\_database} 目录下

# 安装的数据库的名称，默认使用${NEO4J\_HOME}/data/databases/graph.db目录

# The name of the database to mount

#dbms.active\_database=graph.db

#安装Neo4j数据库的各个配置路径，默认使用$NEO4J\_HOME下的路径

#Paths of directories in the installation.

# 数据路径

#dbms.directories.data=data

# 插件路径

#dbms.directories.plugins=plugins

#dbms.directories.certificates=certificates 证书路径

#dbms.directories.logs=logs 日志路径

#dbms.directories.lib=lib jar包路径

#dbms.directories.run=run 运行路径

#默认情况下想load csv文件，只能把csv文件放到${NEO4J\_HOME}/import目录下，把下面的#删除后，可以在load csv时使用绝对路径，这样可能不安全

#This setting constrains all `LOAD CSV` import files to be under the `import` directory. Remove or comment it out to allow files to be loaded from anywhere in the filesystem; this introduces possible security problems. See the `LOAD CSV` section of the manual for details.

#此设置将所有“LOAD CSV”导入文件限制在`import`目录下。删除注释允许从文件系统的任何地方加载文件;这引入了可能的安全问题。

dbms.directories.import=import

#把下面这行的#删掉后，连接neo4j数据库时就不用输密码了

#Whether requests to Neo4j are authenticated. 是否对Neo4j的请求进行了身份验证。

#To disable authentication, uncomment this line 要禁用身份验证，请取消注释此行。

#dbms.security.auth\_enabled=false

#Enable this to be able to upgrade a store from an older version. 是否兼容以前版本的数据

dbms.allow\_format\_migration=true

#Java Heap Size: by default the Java heap size is dynamically calculated based on available system resources. Java堆大小：默认情况下，Java堆大小是动态地根据可用的系统资源计算。

#Uncomment these lines to set specific initial and maximum heap size. 取消注释这些行以设置特定的初始值和最大值

#dbms.memory.heap.initial\_size=512m

#dbms.memory.heap.max\_size=512m

#The amount of memory to use for mapping the store files, in bytes (or kilobytes with the 'k' suffix, megabytes with 'm' and gigabytes with 'g'). 用于映射存储文件的内存量（以字节为单位）千字节带有'k'后缀，兆字节带有'm'，千兆字节带有'g'）。

#If Neo4j is running on a dedicated server, then it is generally recommended to leave about 2-4 gigabytes for the operating system, give the JVM enough heap to hold all your transaction state and query context, and then leave the rest for the page cache. 如果Neo4j在专用服务器上运行，那么通常建议为操作系统保留大约2-4千兆字节，为JVM提供足够的堆来保存所有的事务状态和查询上下文，然后保留其余的页面缓存 。

#The default page cache memory assumes the machine is dedicated to running Neo4j, and is heuristically set to 50% of RAM minus the max Java heap size. 默认页面缓存存储器假定机器专用于运行Neo4j，并且试探性地设置为RAM的50％减去最大Java堆大小。

#dbms.memory.pagecache.size=10g

### Network connector configuration

#With default configuration Neo4j only accepts local connections. Neo4j默认只接受本地连接(localhost)

#To accept non-local connections, uncomment this line: 要接受非本地连接，请取消注释此行

dbms.connectors.default\_listen\_address=0.0.0.0 (这是删除#后的配置，可以通过ip访问)

#You can also choose a specific network interface, and configure a non-default port for each connector, by setting their individual listen\_address. 还可以选择特定的网络接口，并配置非默认值端口，设置它们各自的listen\_address

#The address at which this server can be reached by its clients. This may be the server's IP address or DNS name, or it may be the address of a reverse proxy which sits in front of the server. This setting may be overridden for individual connectors below. 客户端可以访问此服务器的地址。这可以是服务器的IP地址或DNS名称，或者可以是位于服务器前面的反向代理的地址。此设置可能会覆盖以下各个连接器。

#dbms.connectors.default\_advertised\_address=localhost

#You can also choose a specific advertised hostname or IP address, and configure an advertised port for each connector, by setting their individual advertised\_address. 您还可以选择特定广播主机名或IP地址，

为每个连接器配置通告的端口，通过设置它们独特的advertised\_address。

#Bolt connector 使用Bolt协议

dbms.connector.bolt.enabled=true

dbms.connector.bolt.tls\_level=OPTIONAL

dbms.connector.bolt.listen\_address=:7687

#HTTP Connector. There must be exactly one HTTP connector. 使用http协议

dbms.connector.http.enabled=true

dbms.connector.http.listen\_address=:7474

#HTTPS Connector. There can be zero or one HTTPS connectors. 使用https协议

dbms.connector.https.enabled=true

dbms.connector.https.listen\_address=:7473

#Number of Neo4j worker threads. Neo4j线程数

#dbms.threads.worker\_count=

#Logging configuration 日志配置

#To enable HTTP logging, uncomment this line 要启用HTTP日志记录，请取消注释此行

dbms.logs.http.enabled=true

#Number of HTTP logs to keep. 要保留的HTTP日志数

#dbms.logs.http.rotation.keep\_number=5

#Size of each HTTP log that is kept. 每个HTTP日志文件的大小

dbms.logs.http.rotation.size=20m

#To enable GC Logging, uncomment this line 要启用GC日志记录，请取消注释此行

#dbms.logs.gc.enabled=true

#GC Logging Options see http://docs.oracle.com/cd/E19957-01/819-0084-10/pt\_tuningjava.html#wp57013 for more information. GC日志记录选项 有关详细信息，请参见http://docs.oracle.com/cd/E19957-01/819-0084-10/pt\_tuningjava.html#wp57013

#dbms.logs.gc.options=-XX:+PrintGCDetails -XX:+PrintGCDateStamps -XX:+PrintGCApplicationStoppedTime -XX:+PrintPromotionFailure -XX:+PrintTenuringDistribution

#Number of GC logs to keep. 要保留的GC日志数

#dbms.logs.gc.rotation.keep\_number=5

#Size of each GC log that is kept. 保留的每个GC日志文件的大小

#dbms.logs.gc.rotation.size=20m

#Size threshold for rotation of the debug log. If set to zero then no rotation will occur. Accepts a binary suffix "k", "m" or "g". 调试日志旋转的大小阈值。如果设置为零，则不会发生滚动(达到指定大小后切割日志文件)。接受二进制后缀“k”，“m”或“g”。

#dbms.logs.debug.rotation.size=20m

#Maximum number of history files for the internal log. 最多保存几个日志文件

#dbms.logs.debug.rotation.keep\_number=7

### Miscellaneous configuration 其他配置

#Enable this to specify a parser other than the default one. 启用此选项可指定除默认解析器之外的解析器

#cypher.default\_language\_version=3.0

#Determines if Cypher will allow using file URLs when loading data using `LOAD CSV`. Setting this value to `false` will cause Neo4j to fail `LOAD CSV` clauses that load data from the file system. 确定当使用加载数据时，Cypher是否允许使用文件URL `LOAD CSV`。将此值设置为`false`将导致Neo4j不能通过互联网上的URL导入数据，`LOAD CSV` 会从文件系统加载数据。

dbms.security.allow\_csv\_import\_from\_file\_urls=true

#Retention policy for transaction logs needed to perform recovery and backups. 执行恢复和备份所需的事务日志的保留策略

#dbms.tx\_log.rotation.retention\_policy=7 days

#Enable a remote shell server which Neo4j Shell clients can log in to. 启用Neo4j Shell客户端可以登录的远程shell服务器

dbms.shell.enabled=true

#The network interface IP the shell will listen on (use 0.0.0.0 for all interfaces).

dbms.shell.host=127.0.0.1

#The port the shell will listen on, default is 1337.

dbms.shell.port=1337

#Only allow read operations from this Neo4j instance. This mode still requires write access to the directory for lock purposes. 只允许从Neo4j实例读取操作。此模式仍然需要对目录的写访问以用于锁定目的。

#dbms.read\_only=false

#Comma separated list of JAX-RS packages containing JAX-RS resources, one package name for each mountpoint. The listed package names will be loaded under the mountpoints specified. Uncomment this line to mount the org.neo4j.examples.server.unmanaged.HelloWorldResource.java from neo4j-server-examples under /examples/unmanaged, resulting in a final URL of http://localhost:7474/examples/unmanaged/helloworld/{nodeId} 包含JAX-RS资源的JAX-RS软件包的逗号分隔列表，每个安装点一个软件包名称。所列出的软件包名称将在指定的安装点下加载。取消注释此行以装载org.neo4j.examples.server.unmanaged.HelloWorldResource.java neo4j-server-examples下/ examples / unmanaged，最终的URL为http//localhost7474/examples/unmanaged/helloworld/{nodeId}

#dbms.unmanaged\_extension\_classes=org.neo4j.examples.server.unmanaged=/examples/unmanaged

#JVM Parameters JVM参数

#G1GC generally strikes a good balance between throughput and tail latency, without too much tuning. G1GC通常在吞吐量和尾部延迟之间达到很好的平衡，而没有太多的调整。

dbms.jvm.additional=-XX:+UseG1GC

#Have common exceptions keep producing stack traces, so they can be debugged regardless of how often logs are rotated. 有共同的异常保持生成堆栈跟踪，所以他们可以被调试，无论日志被旋转的频率

dbms.jvm.additional=-XX:-OmitStackTraceInFastThrow

#Make sure that `initmemory` is not only allocated, but committed to the process, before starting the database. This reduces memory fragmentation, increasing the effectiveness of transparent huge pages. It also reduces the possibility of seeing performance drop due to heap-growing GC events, where a decrease in available page cache leads to an increase in mean IO response time. Try reducing the heap memory, if this flag degrades performance. 确保在启动数据库之前，“initmemory”不仅被分配，而且被提交到进程。这减少了内存碎片，增加了透明大页面的有效性。它还减少了由于堆增长的GC事件而导致性能下降的可能性，其中可用页面缓存的减少导致平均IO响应时间的增加。如果此标志降低性能，请减少堆内存。

dbms.jvm.additional=-XX:+AlwaysPreTouch

#Trust that non-static final fields are really final. This allows more optimizations and improves overall performance. NOTE: Disable this if you use embedded mode, or have extensions or dependencies that may use reflection or serialization to change the value of final fields! 信任非静态final字段真的是final。这允许更多的优化和提高整体性能。注意：如果使用嵌入模式，或者有可能使用反射或序列化更改最终字段的值的扩展或依赖关系，请禁用此选项！

dbms.jvm.additional=-XX:+UnlockExperimentalVMOptions

dbms.jvm.additional=-XX:+TrustFinalNonStaticFields

#Disable explicit garbage collection, which is occasionally invoked by the JDK itself. 禁用显式垃圾回收，这是偶尔由JDK本身调用。

dbms.jvm.additional=-XX:+DisableExplicitGC

#Remote JMX monitoring, uncomment and adjust the following lines as needed. Absolute paths to jmx.access and jmx.password files are required. 远程JMX监视，取消注释并根据需要调整以下行。需要jmx.access和jmx.password文件的绝对路径。

#Also make sure to update the jmx.access and jmx.password files with appropriate permission roles and passwords, the shipped configuration contains only a read only role called 'monitor' with password 'Neo4j'. 还要确保使用适当的权限角色和密码更新jmx.access和jmx.password文件，所配置的配置只包含名为“monitor”的只读角色，密码为“Neo4j”。

#For more details, see: http://download.oracle.com/javase/8/docs/technotes/guides/management/agent.html On Unix based systems the jmx.password file needs to be owned by the user that will run the server, and have permissions set to 0600. Unix系统，有关详情，请参阅：http：//download.oracle.com/javase/8/docs/technotes/guides/management/agent.html，jmx.password文件需要由运行服务器的用户拥有，并且权限设置为0600。

#For details on setting these file permissions on Windows see: http://docs.oracle.com/javase/8/docs/technotes/guides/management/security-windows.html Windows系统 有关在设置这些文件权限的详细信息，请参阅：http://docs.oracle.com/javase/8/docs/technotes/guides/management/security-windows.html

#dbms.jvm.additional=-Dcom.sun.management.jmxremote.port=3637

#dbms.jvm.additional=-Dcom.sun.management.jmxremote.authenticate=true

#dbms.jvm.additional=-Dcom.sun.management.jmxremote.ssl=false

#dbms.jvm.additional=-Dcom.sun.management.jmxremote.password.file=/absolute/path/to/conf/jmx.password

#dbms.jvm.additional=-Dcom.sun.management.jmxremote.access.file=/absolute/path/to/conf/jmx.access

#Some systems cannot discover host name automatically, and need this line configured: 某些系统无法自动发现主机名，需要配置以下行：

#dbms.jvm.additional=-Djava.rmi.server.hostname=$THE\_NEO4J\_SERVER\_HOSTNAME

#Expand Diffie Hellman (DH) key size from default 1024 to 2048 for DH-RSA cipher suites used in server TLS handshakes. 对于服务器TLS握手中使用的DH-RSA密码套件，将Diffie Hellman（DH）密钥大小从默认1024展开到2048。

#This is to protect the server from any potential passive eavesdropping. 这是为了保护服务器免受任何潜在的被动窃听。

dbms.jvm.additional=-Djdk.tls.ephemeralDHKeySize=2048

### Wrapper Windows NT/2000/XP Service Properties 包装器Windows NT / 2000 / XP服务属性包装器Windows NT / 2000 / XP服务属性

#WARNING - Do not modify any of these properties when an application using this configuration file has been installed as a service. WARNING - 当使用此配置文件的应用程序已作为服务安装时，不要修改任何这些属性。

#Please uninstall the service before modifying this section. The service can then be reinstalled. 请在修改此部分之前卸载服务。 然后可以重新安装该服务。

#Name of the service 服务的名称

dbms.windows\_service\_name=neo4j

### Other Neo4j system properties 其他Neo4j系统属性

dbms.jvm.additional=-Dunsupported.dbms.udc.source=zip