





APOC Java存储过程库

实现复杂和高性能的图遍历

Fanghua(Joshua) Yu

Field Engineering, APAC.

 joshua.yu@neotechnology.com

 <https://www.linkedin.com/in/joshuayu/>

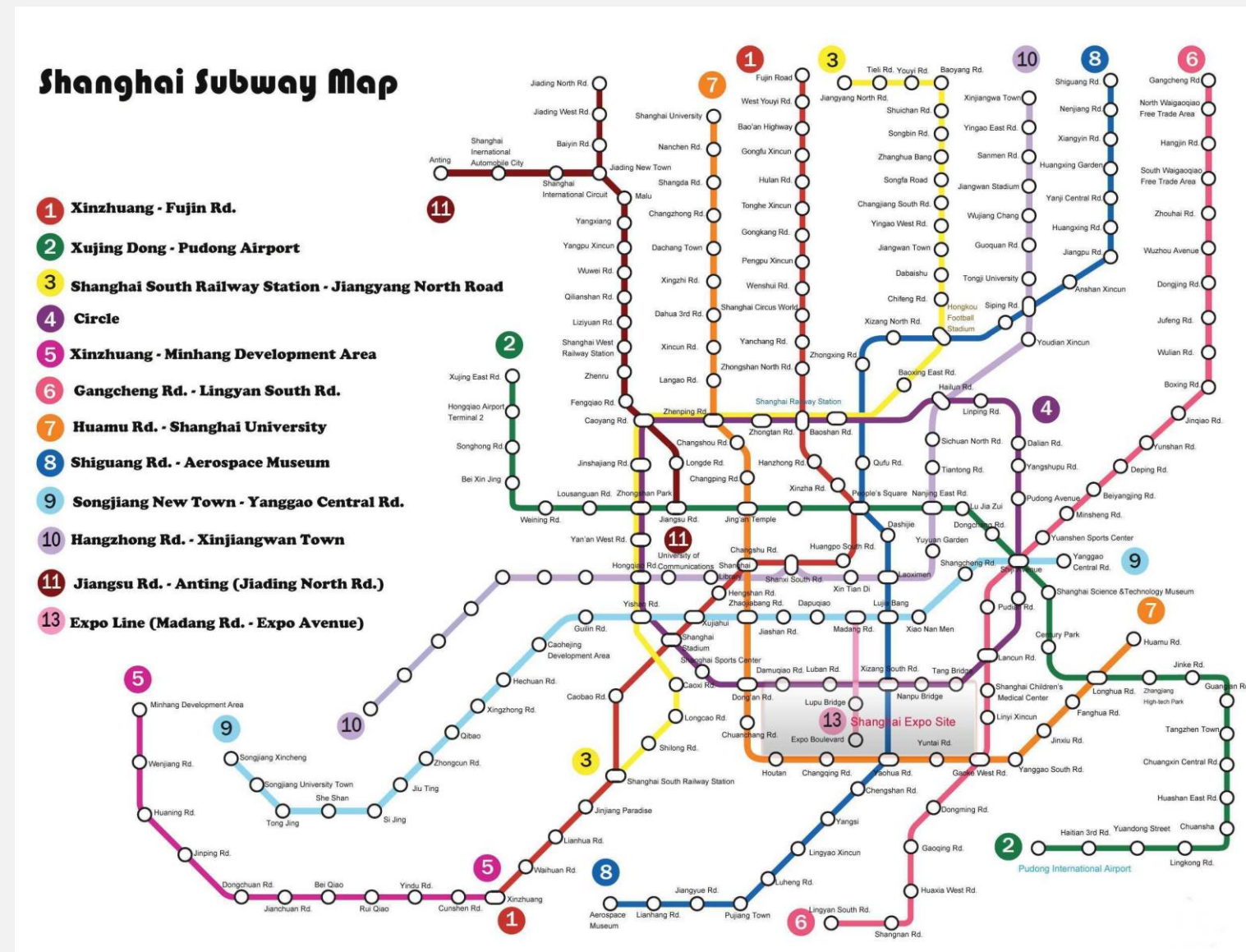
俞方桦, 博士



2730625048, QQ群 : Neo4j中文社区

Neo4j中文社区 : neo4j.com.cn, 用户: GraphWay

2、节点扩展过程：expand()



上海城市轨道交通线路图：世界上距离最长的城市轨道交通系统。



数据准备

本篇中所有实例均基于stackoverflow开放的在线技术文章导出数据。

✓ 介绍（包括数据下载链接）：

<https://neo4j.com/blog/import-10m-stackoverflow-questions/>



✓ 数据类型：用户，帖子，分类标签；已及相互之间的关联关系

✓ 数据规模：~3100万节点，7800万关系，2.6亿属性



导入数据

1. 导入数据，使用import工具*

```
bin\neo4j-admin import --database=stackoverflow.graphdb --id-type=STRING --
nodes:Post="C:\Users\Joshua\Work\Neo4j\data\stackoverflow\csv\posts.csv" --
nodes:User="C:\Users\Joshua\Work\Neo4j\data\stackoverflow\csv\users.csv" --
nodes:Tag="C:\Users\Joshua\Work\Neo4j\data\stackoverflow\csv\tags.csv" --
relationships:PARENT_OF="C:\Users\Joshua\Work\Neo4j\data\stackoverflow\csv\posts_rel.csv" --
relationships:ANSWER="C:\Users\Joshua\Work\Neo4j\data\stackoverflow\csv\posts_answers.csv" --
relationships:HAS_TAG="C:\Users\Joshua\Work\Neo4j\data\stackoverflow\csv\tags_posts_rel.csv" --
relationships:POSTED="C:\Users\Joshua\Work\Neo4j\data\stackoverflow\csv\users_posts_rel.csv"
--ignore-duplicate-nodes=true and --ignore-missing-nodes=true
```

IMPORT DONE in 16m 45s 97ms.

Imported:

31138559 nodes

77930024 relationships

260665346 properties

====>>>> 55.8k tps for node creation

====>>>> 689k tps for relationship creation

====>>>> database size: 16.2GB

====>>>> 261 missing nodes for relationships

Available resources:

Total machine memory: 11.90 GB

Free machine memory: 5.96 GB

Max heap memory : 2.65 GB

Processors: 4

Configured max memory: 2.98 GB

* 企业版特性

- ✓ 适合大量数据一次性导入
- ✓ 忽略重复节点
- ✓ 忽略未能匹配的节点
- ✓ 无法插入的数据会保存在报告文件供事后分析



下载和安装APOC存储过程包

✓ 参见本系列文章的第一部分【1】概述：

Neo4j中文社区链接：<http://neo4j.com.cn/topic/5a484be7e4d9330d4b2f584f>

CSDN链接：<http://blog.csdn.net/GraphWay/article/details/78957415>

！在Neo4j安装目录下的`neo4j.conf`中加入：

```
dbms.security.procedures.unrestricted=apoc.*
```

建议增加页缓存到至少4G，推荐20G（如果你有足够的内存）：

```
dbms.memory.pagecache.size=4g
```

JVM堆保留内存从1G起，最大4G：

```
dbms.memory.heap.initial_size=1g
```

```
dbms.memory.heap.max_size=4g
```



一切就绪，现在就要看apoc的了！





理解数据模型(1)

❖ 在neo4j浏览器中输入并运行：

```
CALL apoc.meta.graph()
```

- ❖ 熟悉要处理的数据模型；
- ❖ 依次点击每个节点**标签**：Post / Tag / User，定义它们的颜色和大小，以及用做显示标题的属性；
- ❖ 依次点击每个关系**类型**：HAS_TAG / POSTED / ANSWER / PARENT_OF，定义它们的颜色和大小，以及显示标题；
- ❖ 所有的设置在同一数据库、同一网页浏览器中会被保存

The screenshot shows the Neo4j browser interface. At the top, the command `$ CALL apoc.meta.graph()` is entered. Below the command, there are several buttons representing nodes and relationships: `*(3)`, `Post(1)` (circled in red), `Tag(1)`, `User(1)`, `*(4)`, `HAS_TAG(1)`, `POSTED(1)`, `ANSWER(1)`, and `PARENT_OF(1)`. The graph visualization shows a central blue node labeled 'Post' with a self-loop labeled 'PARENT_OF', a yellow node labeled 'User' connected to 'Post' by a relationship labeled 'POSTED', and a green node connected to 'Post' by a relationship labeled 'HAS_TAG'. There is also a self-loop on 'Post' labeled 'ANSWER'. A red arrow points to the 'Post(1)' button. At the bottom, the configuration panel for the selected 'Post' node is visible, showing 'Color' set to blue and 'Caption' set to '<id> name count'.



理解数据模型(2)

- ❖ 点击图上的节点，能够在下面的状态栏中看到：
 - 内部id(其实没什么用)；
 - 标签名称(label)，代表节点类型；
 - 节点总数(啊哈！)
- ❖ 点击图上的关系，同样能看到：
 - 内部id
 - 关系类型名
 - 入度：IN
 - 出度：OUT

\$ CALL apoc.meta.graph()

Graph: *(3) Post(1) Tag(1) User(1)

Table: *(4) HAS_TAG(1) POSTED(1) ANSWER(1) PARENT_OF(1)

Text: A

Code: </>

Post <id>: -2 name: Post **count: 26545725**

有点意思了吧？接着看下去。



理解数据模型(3)

实在喜欢Excel的朋友，或者对关系型数据库念念不忘的程序员们，好吧，试试这个命令：

```
CALL apoc.meta.data()
```

知道你会忍不住点这个按钮的。。。

\$ CALL apoc.meta.data()

"Post"	"createdAt"	0	false	true	false	"STRING"	false	(empty)	0	0	0	0	0	0	0
"Post"	"score"	0	false	true	false	"INTEGER"	false	(empty)	0	0	0	0	0	0	0
"Post"	"views"	0	false	true	false	"INTEGER"	false	(empty)	0	0	0	0	0	0	0
"Post"	"answers"	0	false	true	false	"INTEGER"	false	(empty)	0	0	0	0	0	0	0
"Post"	"comments"	0	false	false	false	"INTEGER"	false	(empty)	0	0	0	0	0	0	0
"Post"	"favorites"	0	false	true	false	"INTEGER"	false	(empty)	0	0	0	0	0	0	0
"Post"	"updatedAt"	0	false	false	false	"STRING"	false	(empty)	0	0	0	0	0	0	0
"Post"	"body"	0	false	false	false	"STRING"	false	(empty)	0	0	0	0	0	0	0
"User"	"POSTED"	26647	false	false	false	"RELATIONSHIP"	true	(empty)	117010793	26647	4391	1	["Post"]	0	
"User"	"userId"	0	true	true	false	"STRING"	false	(empty)	0	0	0	0	0	0	0
"User"	"name"	0	false	true	false	"STRING"	false	(empty)	0	0	0	0	0	0	0
"User"	"reputation"	0	false	true	false	"INTEGER"	false	(empty)	0	0	0	0	0	0	0
"User"	"createdAt"	0	false	true	false	"STRING"	false	(empty)	0	0	0	0	0	0	0
"User"	"accessedAt"	0	false	false	false	"STRING"	false	(empty)	0	0	0	0	0	0	0
"User"	"url"	0	false	false	false	"STRING"	false	(empty)	0	0	0	0	0	0	0
"User"	"location"	0	false	false	false	"STRING"	false	(empty)	0	0	0	0	0	0	0
"User"	"views"	0	false	false	false	"INTEGER"	false	(empty)	0	0	0	0	0	0	0

现在满足了吧？



节点扩展(1)

节点扩展过程`apoc.path.expand()`可以从给定节点或节点列表开始，沿着指定的关系类型进行遍历，直到特定结束条件满足时停止，并返回路径或节点。

```
CALL apoc.path.expand(startNode <id>|Node, relationshipFilter,  
labelFilter, minLevel, maxLevel ) YIELD path
```

- `startNode` : 起始节点，可以是节点的id或者节点变量
- `relationshipFilter` : 遍历关系的过滤条件，用'|'分隔
- `labelFilter` : 遍历节点的过滤条件，用'|'分隔(见下页)
- `minLevel` : 最小遍历层级
- `maxLevel` : 最大遍历层级
- `path` : 返回的路径列表

在线文档：https://neo4j-contrib.github.io/neo4j-apoc-procedures/#_expand_paths



节点扩展(2)

labelFilter：遍历节点的过滤条件可以有四种操作类型(以下均用Post作为节点例子)：

格式	含义	说明
-Post	排除	Post节点不被遍历，也不被包括在返回的路径中。
+Post	包含	缺省。Post节点将被遍历，也被包括在返回的路径中。
/Post	终止且返回	遍历路径直到遇见Post类型的节点，然后仅返回Post节点。
>Post	终止但是继续	遍历路径只返回到达Post类型的节点(含)之前的部分，在Post节点之后的部分会继续被遍历，但是不会被返回。



节点扩展(3)

先来小试牛刀一下。运行下面的查询：

```
MATCH (p:Post) WHERE id(p) = 3
WITH p
CALL apoc.path.expand(p, 'PARENT_OF>|ANSWER', '+Post', 0, 10) YIELD path as paths
RETURN paths;
```

- p：起始节点，是id为3的帖子
- ‘PARENT_OF>|ANSWER’：遍历仅沿着这两个关系进行，其中PARENT_OF是有向的(从Post离开的)，ANSWER是双向的
- ‘+Post’：包括帖子类型的节点
- 0：最小遍历层级
- 10：最大遍历层级
- YIELD path: 返回结果存放在path变量中。

注意：这里的返回变量名称**必须**和存储过程接口定义中的一致，否则会报错。

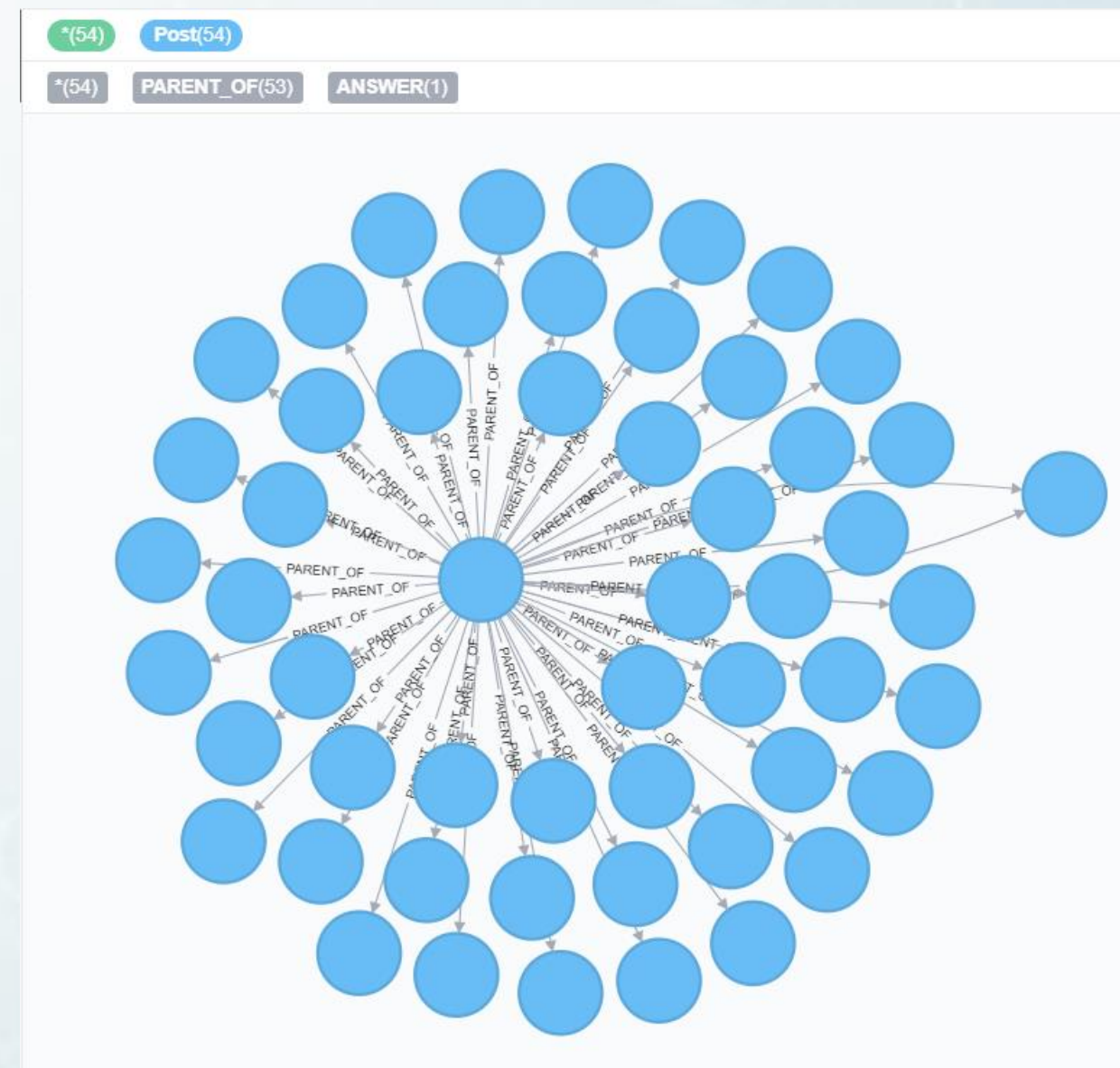


节点扩展(4)

结果如图。基本上得到的是某文章的所有子文章(PARENT_OF)和它被引用来回答(ANSWER)的其他文章。

由于数据导入时系统指定的内部id可能不一样，在你的电脑上返回的结果会略有不同。

这个，你懂的。





节点扩展(5)

来点复杂的。

我们统计一下关于'neo4j'的每个帖子的跟贴和回帖总数，并保存在一个新的属性threads中。

```
MATCH (p:Post) -[:HAS_TAG]-> (t:Tag{tagId:'neo4j'})
WITH p
CALL apoc.path.expand(p, 'PARENT_OF>|ANSWER>', '+Post', 0, 10) YIELD path as paths
WITH p, count(paths) AS numberOfThreads
SET p.threads = numberOfThreads
RETURN id(p), p.threads ORDER BY p.threads;
```

为每个帖子统计存在的路径数目

结果保存在新的属性中。

! Neo4j是Schemaless(无模式)数据库，添加新属性不需要事先修改节点定义，直接创建就行。



节点扩展(6)

然后就可以根据跟贴和回帖数排序：最热门文章。

```
MATCH (p:Post) -[HAS_TAG]-> (t:Tag{tagId:'neo4j'})
RETURN p ORDER BY p.threads DESC LIMIT 10;
```

\$ MATCH (p:Post) -[HAS_TAG]-> (t:Tag{tagId:'neo4j'}) RETURN p ORDER BY p.threads DESC LIMIT 10;

*(10) Post(10)

Graph

Table

Text

Code

Post <id>: 1396610 favorites: 22 createdAt: 2009-11-18T09:06:25.823 score: 27 comments: 5 postType: 1 answers: 7 threads: 9 postId: 1754628

点击这里可以展开长文本属性的内容。



节点扩展(7)

遍历有向关系和双向关系。在Neo4j中，关系在存储时**必须**是有向的，在查询时可以有向也可以无向/双向。遍历时，如果不指定关系的方向，会对同一对节点中的同一个关系重复遍历，这时expand()会认为是不同的路径，即“节点1-节点2”和“节点2-节点1”。

试试下面的查询：

```
MATCH (p:Post) -[:HAS_TAG]-> (t:Tag{tagId:'neo4j'})
WITH p
CALL apoc.path.expand(p, 'PARENT_OF>|ANSWER!', '+Post', 0, 10) YIELD path as paths
WITH p, count(paths) AS numberOfThreads
SET p.threads = numberOfThreads
RETURN id(p), p.threads ORDER BY p.threads;
```

这个查询和第(5)部分的完全一样，除了这里的方向符号'>'被去掉了，结果是返回的路径数几乎翻倍。



可配置节点扩展(1)

可配置节点扩展过程`apoc.path.expandConfig()`提供了更多的配置选择。

```
CALL apoc.path.expandConfig(startNode <id>Node/list, {config})  
YIELD path
```

配置选项格式如下：

```
{minLevel: -1|number,  
maxLevel: -1|number,  
relationshipFilter: '[<]RELATIONSHIP_TYPE1[>][<]RELATIONSHIP_TYPE2[>]|...',  
labelFilter: '[+/-/>]LABEL1|LABEL2|...',  
uniqueness: 见下页  
bfs: true|false,  
filterStartNode: true|false,  
limit: -1|number,  
optional: true|false}
```

注：

- **粗体**是缺省值。
- filterStartNode：节点过滤规则是否包括起始节点
- bfs: 宽度优先搜索。



可配置节点扩展(2)

uniqueness(唯一性属性)决定在遍历过程中如何判断重复路径，包括关系和节点。

值	说明
RELATIONSHIP_PATH	缺省值。关系序列唯一。
NODE_GLOBAL	全局节点唯一。
NODE_LEVEL	在同一层级上的节点唯一。
NODE_PATH	全路径唯一。

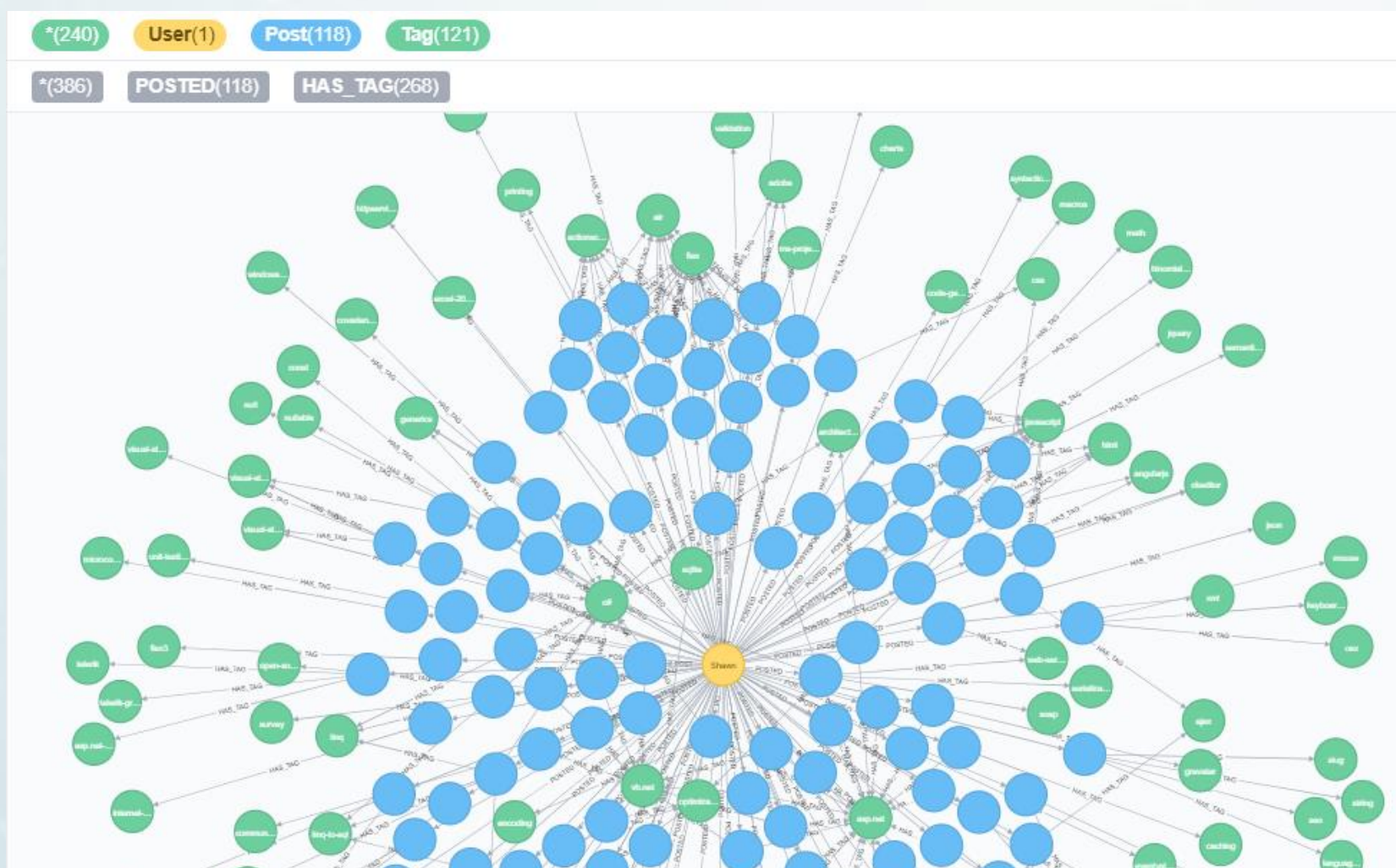
值	说明
NODE_RECENT	最近访问过的节点中唯一。
RELATIONSHIP_GLOBAL	全局关系唯一。
RELATIONSHIP_LEVEL	在同一层级上的关系唯一。
RELATIONSHIP_RECENT	最近访问过的关系中唯一。
NONE	不限制。(电脑烧坏了不要叫!)



可配置节点扩展(3)

查找某个用户的所有发表文章，及其主题类别：

```
MATCH (u:User{userId:'26'})
WITH u
CALL apoc.path.expandConfig(u,{relationshipFilter:'POSTED>|HAS_TAG>',labelFilter:'+Post|/Tag'}) yield path
RETURN path
```



- 不指定唯一性条件时，缺省是 RELATIONSHIP_PATH，即“关系序列唯一”。

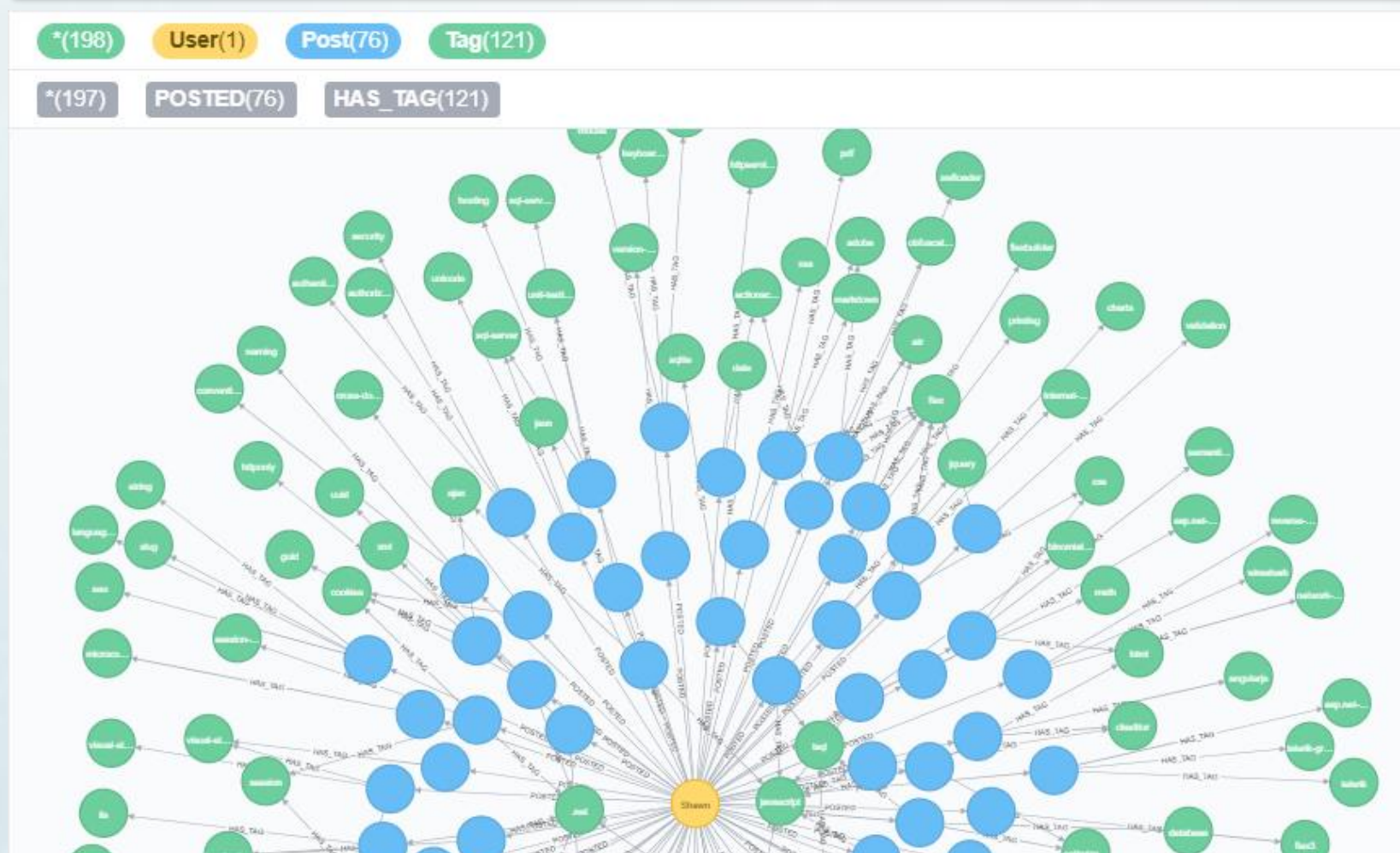
- 返回118个帖子(Post)
- 121主题类别(Tag)
- 118个POSTED关系
- 268个HAS_TAG关系
- 共240节点，386个关系



可配置节点扩展(4)

查找某个用户的所有发表文章，及其主题类别，不访问重复节点：

```
MATCH (u:User{userId:'26'})
WITH u
CALL apoc.path.expandConfig(u,
{relationshipFilter:'POSTED>|HAS_TAG>',labelFilter:'+Post|/Tag',uniqueness:'NODE_GLOBAL'}) yield path
RETURN path
```



- 指定节点全局唯一，即如果节点访问过则不返回路径。

- 返回76个帖子(Post)
- 121主题类别(Tag)
- 76个POSTED关系
- 121个HAS_TAG关系
- 共198节点，197个关系



可配置的员工扩展(5)

Ummmmm....

避免重复遍历是提高查询效率的关键。根据具体情况的不同，可以指定判断重复的条件，从而减少需要遍历和返回的路径。这里，两种方式都达到了终点，121个主题类别，但是第二种方式所返回的节点减少了接近18%，关系则减少近一半！

使用Cypher的例子：和第一种方式返回的节点和关系数目相同。



```
MATCH (u:User{userId:'26'})
WITH u
MATCH (u) -[a:POSTED]-> (p:Post) -[h:HAS_TAG]-> (t:Tag)
RETURN *
```



进一步探讨(1)

下面我们尝试更多的一些查询：寻找与neo4j主题类别相关的其他主题标签。

```
MATCH (t:Tag{tagId:'neo4j'})
WITH t
CALL apoc.path.expandConfig(t,{relationshipFilter:'HAS_TAG',labelFilter:'+Post|/Tag',uniqueness:'NODE_GLOBAL'}) YIELD path
RETURN path
```

- 从'neo4j'主题标签出发，沿着HAS_TAG找到帖子，再找到该帖子引用的其他主题标签，同样是沿着HAS_TAG关系
- 返回所有的路径到path变量中
- path变量是一个路径的列表

接下来，用filter()来提取路径中不是'neo4j'的其他主题标签节点，再用extract()来提取标签节点的名称属性，并统计这些标签名称的出现次数

```
WITH filter(n IN nodes(path) WHERE n.tagId <> 'neo4j' AND n.tagId IS NOT NULL) AS tags
RETURN extract(n IN tags | n.tagId) AS tag,
count(tags)
```



进一步探讨(2)

我们将曾经在同一个帖子里被引用的主题标签间建立一个新的关系：ASSOCIATED_WITH。在正式写入数据库之前，可以使用apoc的“虚拟关系”先在屏幕上显示出来。

```
MATCH (t:Tag{tagId:'neo4j'})
WITH t
CALL apoc.path.expandConfig(t,{relationshipFilter:'HAS_TAG',labelFilter:'+Post|/Tag',uniqueness:'RELATIONSHIP_PATH'}) YIELD path
WITH t,filter(n IN nodes(path) WHERE n.tagId <> 'neo4j' AND n.tagId IS NOT NULL) AS tags
WITH t,extract(n IN tags | n.tagId) AS tag, count(tags) AS counts
MATCH (t2:Tag{tagId:tag[0]})
CALL apoc.create.vRelationship(t, 'ASSOCIATED_WITH',{count:counts},t2) YIELD rel
RETURN t,t2,rel
```

- apoc.create.vRelationship()可以创建虚拟关系，即不真正写入到数据库中的关系。虚拟节点/关系/图便于：
 - 在屏幕上显示数据库中不存在的关系
 - 在写入数据库前直观地验证结果

虚拟关系ASSOCIATED_WITH连接2个主题标签，并有一个count属性，记录共同出现过的次数。



进一步探讨(3)

虚拟关系仅仅用来作为结果显示在Neo4j浏览器，因此是不会显示在数据库标签和关系列表中，也不会在MATCH中被返回，并且不能被排序。下面我们实际地创建关系。

```
MATCH (t:Tag{tagId:'neo4j'})
WITH t
CALL apoc.path.expandConfig(t,{relationshipFilter:'HAS_TAG',labelFilter:'+Post|/Tag',uniqueness:'RELATIONSHIP_PATH'}) YIELD path
WITH t,filter(n IN nodes(path) WHERE n.tagId <> 'neo4j' AND n.tagId IS NOT NULL) AS tags
WITH t,extract(n IN tags | n.tagId) AS tag, count(tags) AS counts
MATCH (t2:Tag{tagId:tag[0]})
SET t.processed = true
MERGE (t) -[:ASSOCIATED_WITH{count:counts}]-> (t2)
```

- MERGE可以先确定数据库中，指定的两个节点间是否已经存在要创建的关系：如果没有则创建；否则忽略。



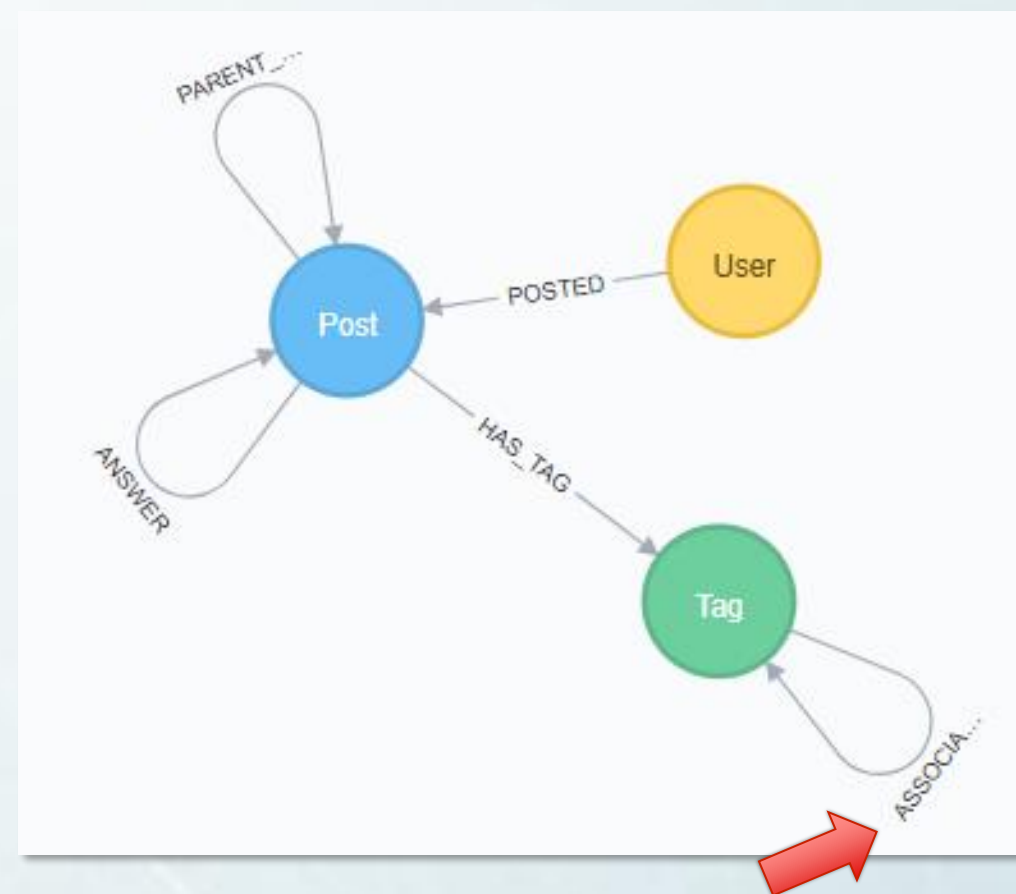
进一步探讨(4)

如果还记得怎样显示数据库模型，现在可以注意到在Tag节点上多了一指向自己的关系ASSOCIATED_WITH。

在Neo4j中，同样，新建关系不需要修改元数据定义。对地！



不过，Schemaless模式并不意味着数据就是任意存放、无法/无需管理的！其主要目的是提供处理多样化数据的高度灵活性。在Neo4j中仍旧可以定义和使用限制 (constraints)来确保完整性。





进一步探讨(5)

最后，让我们看看哪些主题和neo4j相关度最高。

```
MATCH (t:Tag{tagId:'neo4j'}) -[a:ASSOCIATED_WITH]-> (t2)
RETURN t2.tagId,a.count, toFloat(10000 * a.count/t2.count) AS percentileX100
ORDER BY a.count DESC LIMIT 10
```

- 主题标签/Tag有一个count字段，保存的是所有拥有该标签的帖子总数。这里我们用它来计算和'neo4j'相关的帖子占总数的万分比
- 显然，'cypher'是最相关的：与'neo4j'同时出现的比率是97.28%，接下来是'py2neo' – 88.97%，然后是'spring-data-neo4j' – 80.02%。
- 'neo4j'与'java'同时出现的次数第二多(1118)，但是显然Java有太广泛的问题领域，比率反而很低，只有0.12%。
- 恭喜你！你已经实现了一个“推荐引擎”了。

```
$ MATCH (t:Tag{tagId:'neo4j'}) -[a:ASSOCIATED_WITH]->
```

"t2.tagId"	"a.count"	"percentileX100"
"cypher"	2724	9728
"java"	1118	12
"spring-data-neo4j"	725	8002
"graph-databases"	535	6279
"graph"	345	281
"spring"	269	38
"neo4jclient"	258	7186
"python"	243	5
"py2neo"	218	8897
"database"	194	19



总结

- 了解图数据库中已有数据的模型apoc.meta.*
- apoc的路径扩展函数expand()和expandConfig()
- 虚拟关系和路径：apoc.create.vRelationship()
- 即时创建属性和关系
- 处理列表型数据的Cypher函数：
 - filter()
 - extract()
- 计算相关性的简单方法
- 一个推荐引擎！





感谢阅读！

欢迎提出问题、意见和建议。

Neo4j中文社区：<http://neo4j.com.cn>

QQ群：Neo4j中文社区 / 547190638

个人QQ号：Neo4j-APAC技术支持 / 2730625048