

数据集导入

1. Neo4j安装

根据课上所学知识我们可以知道 Neo4j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。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。在一个图中包含两种基本的数据类型：Nodes（节点）和 Relationships（关系）。Nodes和 Relationships 包含key/value形式的属性。Nodes通过Relationships所定义的关系相连起来，形成关系型网络结构。因此我们本次项目选择采用此方式进行数据集的存储和管理。具体操作流程如下

1. 安装java11

Neo4j 4.x以上版本需要java11，与服务器环境不符，因此我们下载安装了java11并更改环境变量来进行安装。

2. 安装Neo4j

我们选择了智慧树上的neo4j-community-4.4.11-windows进行安装，配置环境变量NEO4J_HOME，并在环境变量Path中添加%NEO4J_HOME%\bin，至此安装完成。

2. 数据集导入

作业要求从COVID-19知识图谱中选取RDF数据集下载，导入知识图谱数据库进行存储管理，要求导入后台数据库的知识图谱规模总量大于一千万条（10 million）三元组，我们选择了String Human PPI数据集，数据集包含了71,787,524个三元组，并进行导入和解析。

1. neosemantics安装

Neo4j 本身是不支持导入 RDF 文件的。但我们可以借助 neosemantics 来进行导入。neosemantics 简称 n10s，4.0之前的叫 semantics。由于我们使用了最新的neo4j-community-4.4.11版本，因此我们下载了最新的4.4.0.2版本的插件，并将其复制到 neo4j 的安装目录下的 plugins 文件夹中，在 neo4j/neo4j.conf 文件中添加以下内容：

```
dbms.unmanaged_extension_classes=n10s.endpoint=/rdf
```

之后进入neo4j浏览器中进行初始化：

```
CALL n10s.graphconfig.init();
```

将数据持久化到[Neo4j](#)中的所有方法都有一个模式级的先决条件：这是在带有标签资源的节点的属性URI上存在唯一性约束。如果约束还没有出现，需要在neo4j上运行：

```
CREATE CONSTRAINT n10s_unique_uri ON (r:Resource) ASSERT r.uri IS  
UNIQUE;
```

否则rdf导入会报错。这个约束的目的是通过URI保证资源的唯一性，并通过将资源添加到索引来加速获取过程。此时大功已经告成了。我们可以导入 RDF文件了。

2. 导入数据

运行：

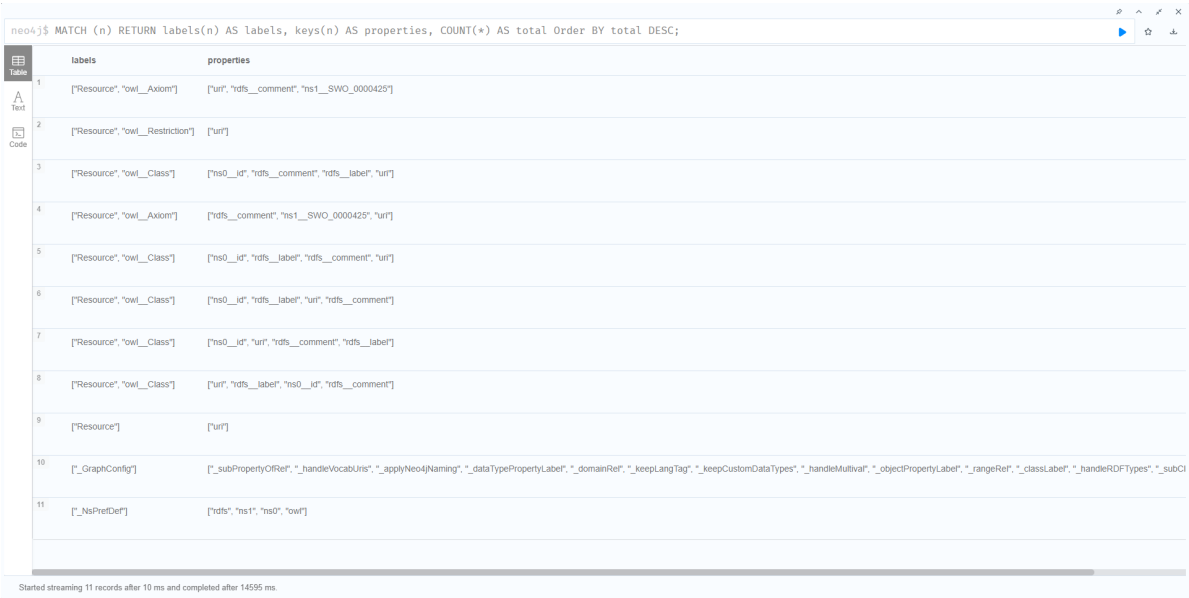
```
call n10s.rdf.import.fetch("file://%NEO4J_HOME%\import\string-  
human.rdf", "Turtle");
```

导入成功。

数据集说明

1. 通过执行下列命令来获得所有顶点聚类后的分类和属性：

```
MATCH (n) RETURN labels(n) AS labels, keys(n) AS properties, COUNT(*) AS total
Order BY total DESC;
```



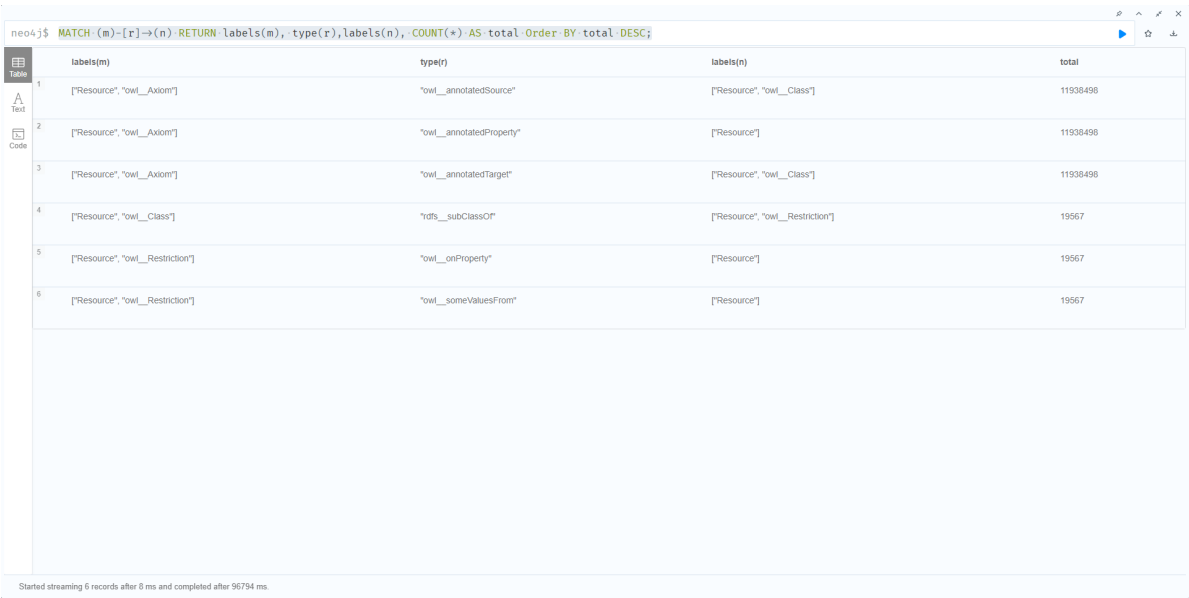
The screenshot shows the Neo4j query interface with the following query: `neo4j$ MATCH (n) RETURN labels(n) AS labels, keys(n) AS properties, COUNT(*) AS total Order BY total DESC;`

	labels	properties
1	["Resource", "owl_Axiom"]	["uri", "rdfs_comment", "ns1_SWO_0000429"]
2	["Resource", "owl_Restriction"]	["uri"]
3	["Resource", "owl_Class"]	["rso_id", "rdfs_comment", "rdfs_label", "uri"]
4	["Resource", "owl_Axiom"]	["rdfs_comment", "ns1_SWO_0000429", "uri"]
5	["Resource", "owl_Class"]	["rso_id", "rdfs_label", "rdfs_comment", "uri"]
6	["Resource", "owl_Class"]	["rso_id", "rdfs_label", "uri", "rdfs_comment"]
7	["Resource", "owl_Class"]	["rso_id", "uri", "rdfs_comment", "rdfs_label"]
8	["Resource", "owl_Class"]	["uri", "rdfs_label", "rso_id", "rdfs_comment"]
9	["Resource"]	["uri"]
10	["_GraphConfig"]	["_subPropertyOrRef", "_handleVocabUri", "_applyNeo4jNaming", "_dataTypePropertyLabel", "_domainRef", "_keepLangTag", "_keepCustomDataTypes", "_handleMultival", "_objectPropertyLabel", "_rangeRef", "_classLabel", "_handleRDFTypes", "_subC"]
11	["_NsPretDef"]	["rdfs", "ns1", "rso", "owl"]

Started streaming 11 records after 10 ms and completed after 14595 ms.

2. 通过执行下列命令来获得顶点和属性之间的关系类型：

```
MATCH (m)-[r]->(n) RETURN labels(m), type(r), labels(n), COUNT(*) AS total Order BY total DESC;
```



The screenshot shows the Neo4j query interface with the following query: `neo4j$ MATCH (m)-[r]->(n) RETURN labels(m), type(r), labels(n), COUNT(*) AS total Order BY total DESC;`

	labels(m)	type(r)	labels(n)	total
1	["Resource", "owl_Axiom"]	"owl_annotatedSource"	["Resource", "owl_Class"]	11938498
2	["Resource", "owl_Axiom"]	"owl_annotatedProperty"	["Resource"]	11938498
3	["Resource", "owl_Axiom"]	"owl_annotatedTarget"	["Resource", "owl_Class"]	11938498
4	["Resource", "owl_Class"]	"rdfs_subClassOf"	["Resource", "owl_Restriction"]	19567
5	["Resource", "owl_Restriction"]	"owl_onProperty"	["Resource"]	19567
6	["Resource", "owl_Restriction"]	"owl_someValuesFrom"	["Resource"]	19567

Started streaming 6 records after 8 ms and completed after 96794 ms.