

图数据库

图的特征：

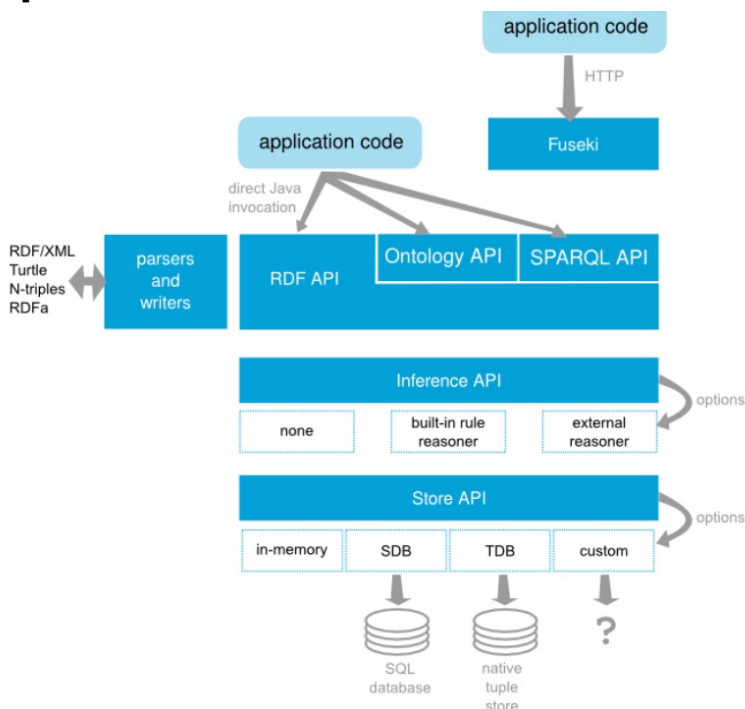
包含节点和边

节点上有属性 (键值对)

边有名字和方向，并总是有一个开始节点和一个结束节点

边也可以有属性

Apache Jena



最底层的是数据库，包含SQL数据库和原生数据库，其中SDB用来导入SQL数据库，TDB导入RDF三元组。

数据库之上的是内建的和外联的推理接口。

在往上的就是SPARQL查询接口。通过直接使用SPARQL语言或通过Refo等模块转换成SPARQL语言进行查询。

上方Fuseki模块，它相当于一个服务器端，我们的操作就是在它提供的端口上进行。

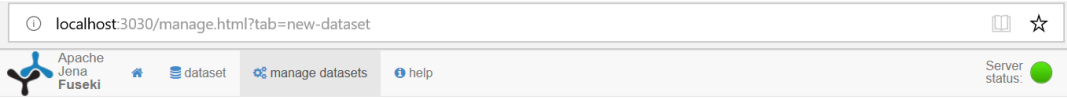
TDB是Jena用于存储RDF的组件，是属于存储层面的技术。在单机情况下，它能够提供非常高的RDF存储性能。目前TDB的最新版本是TDB2，且与TDB1不兼容。

Jena提供了RDFS、OWL和通用规则推理机。其实Jena的RDFS和OWL推理机也是通过Jena自身的通用规则推理机实现的。

Fuseki是Jena提供的SPARQL服务器，也就是SPARQL endpoint。其提供了四种运行模式：单机运行、作为系统的一个服务运行、作为web应用运行或者作为一个嵌入式服务器运行。

数据导入

1.通过Fuseki的手动导入



Manage datasets

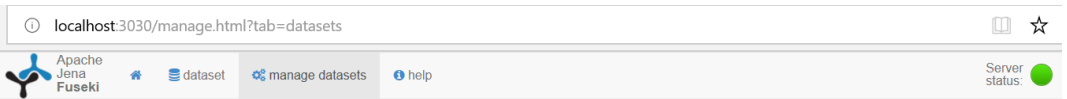
Perform management actions on existing datasets, including backup, or add a new dataset.

existing datasets add new dataset

Dataset name

Dataset type

- ☐ In-memory – dataset will be recreated when Fuseki restarts, but contents will be lost
- ☐ Persistent – dataset will persist across Fuseki restarts
- ☒ Persistent (TDB2) – dataset will persist across Fuseki restarts

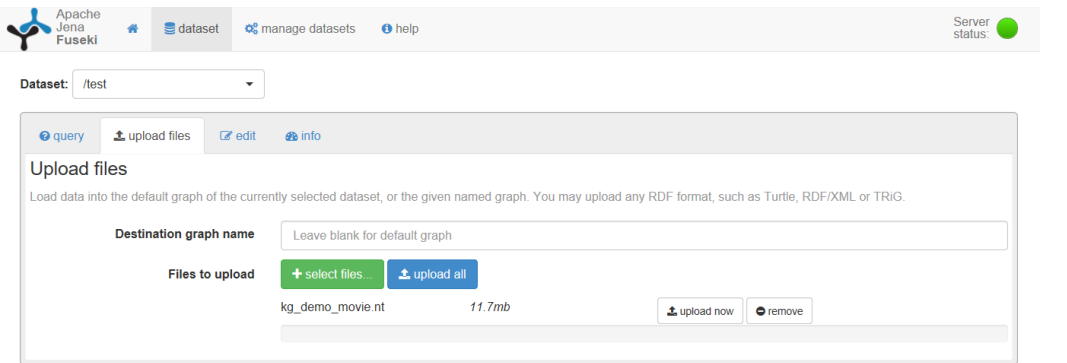


Manage datasets

Perform management actions on existing datasets, including backup, or add a new dataset.

existing datasets add new dataset

Name
/test



2.通过TDB进行导入

使用TDB导入的命令如下

```
/jena-fuseki/tdbloader --loc=/jena-fuseki/data filename
```



Fuseki启动的命令如下，需要指定tdb生成的文件路径并指定数据库名

```
/jena-fuseki/fuseki-server --loc=/jena-fuseki/data :  
--update /music |
```

查询

Fuseki界面查询

使用endpoint接口查询

Endpoint接口查询

endpoint地址

SPARQL Query: <http://localhost:3030/music/query>;

SPARQL Update: <http://localhost:3030/music/update>

Python操作Apache Jena

- 使用 Jena SPARQL endpoint 接口进行查询和更新
- 使用SPARQLWrapper包查询和更新

```
def query(query_string=None):
    sparql_query = SPARQLWrapper(query_url)
    sparql_query.setQuery(query_string)
    sparql_query.setReturnFormat(JSON)

    results = None
    try:
        results = sparql_query.query().convert()
    except:
        return False
    return results
```

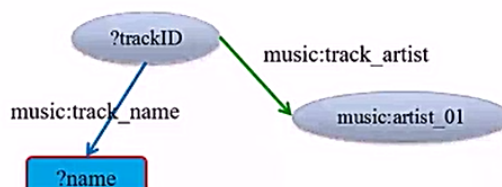
```
def update(update_string):
    sparql_update = SPARQLWrapper(update_endpoint=update_url)
    sparql_update.setQuery(update_string)
    sparql_update.method = "POST"
    sparql_update.setReturnFormat(JSON)

    results = None
    try:
        results = sparql_update.query()
    except:
        return False
    if "Success" in results.response.read():
        return True
    return False
```

<https://rdflib.github.io/sparqlwrapper/>

如果我想得到某一位歌手所有歌曲的歌曲名，应该怎么办？

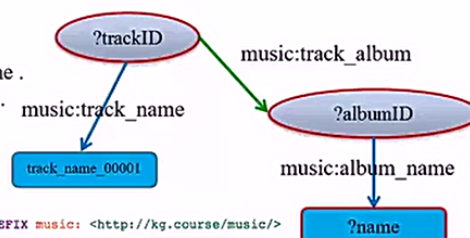
```
SELECT ?name
WHERE {
  ?trackID track_artist artistID .
  ?trackID track_name ?name
}
```



```
1 PREFIX music: <http://kg.course/music/>
2
3 SELECT ?name
4 WHERE {
5   ?trackID music:track_artist music:artist_01
6   ?trackID music:track_name ?name
7 }
```

查询某一首歌曲名对应的专辑信息

```
SELECT ?name
WHERE {
  ?trackID track_name track_name .
  ?trackID track_album ?albumID .
  ?albumID album_name ?name
}
```



```
1 PREFIX music: <http://kg.course/music/>
2
3 SELECT ?trackID ?albumID ?name
4 WHERE {
5   ?trackID music:track_name "track_name_00001" .
6   ?trackID music:track_album ?albumID .
7   ?albumID music:album_name ?name
8 }
9
```

歌曲名为歌曲的属性，需要查出歌曲id，再查询属性专辑信息

□ 我想在专辑名前面加一些描述，应该怎么办？

```
SELECT ?歌曲id ?专辑id (CONCAT("专辑
名","?专辑名) AS ?专辑信息)
WHERE {
  ?歌曲id track_name track_name .
  ?歌曲id track_album ?专辑id .
  ?专辑id album_name ?专辑名
}
```

使用CONCAT连接

```
1= PREFIX music: <http://kg.course/music/>
2
3 SELECT ?歌曲id ?专辑id (CONCAT("专辑名","?专辑名) AS ?专辑信息)
4= WHERE {
5   ?歌曲id music:track_name "track_name_00001" .
6   ?歌曲id music:track_album ?专辑id .
7   ?专辑id music:album_name ?专辑名
8 }
9
```

QUERY RESULTS

Table Raw Response

Showing 1 to 1 of 1 entries Search: Show 50 entries

歌曲id	专辑id	专辑信息
music:track_00001	music:album_0001	"专辑名album_name_0001"

```
1= PREFIX music: <http://kg.course/music/>
2
3 SELECT ?albumID ?trackID
4= WHERE {
5   ?albumID music:album_name "album_name_0002" .
6   ?trackID music:track_album ?albumID
7 }
8
9 LIMIT 2
```

使用Limit 限制查询结果的条数

计算某一个专辑的歌曲数目呢？

```
SELECT (COUNT(?trackID) AS ?num)
WHERE {
  ?albumID album_name album_name .
  ?trackID track_album ?albumID
}
```

使用COUNT进行计数

```
1= PREFIX music: <http://kg.course/music/>
2
3 SELECT (COUNT(?trackID) as ?num)
4= WHERE {
5   ?albumID music:album_name "album_name_0002" .
6   ?trackID music:track_album ?albumID
7 }
8
```

QUERY RESULTS

Table Raw Response

Showing 1 to 1 of 1 entries Search: Show 50 entries

num
"9"^^xsd:integer

"9"^^xsd:integer integer 为数据类型

□ 查询某一歌手唱过歌曲的所有标签

```
SELECT DISTINCT ?tag_name
WHERE {
  ?trackID track_artist artistID .
  ?trackID track_tag ?tag_name
}
```

```
1 PREFIX music: <http://kg.course/music/>
2
3 SELECT ?tag_name
4 WHERE {
5   ?trackID music:track_artist music:artist_001 .
6   ?trackID music:track_tag ?tag_name
7 }
8
```

直接查询所有的歌曲标签，发现标签有重复

	tag_name
1	"tag_name_02"
2	"tag_name_02"
3	"tag_name_06"
4	"tag_name_05"
5	"tag_name_02"
6	"tag_name_02"
7	"tag_name_02"
8	"tag_name_08"
9	"tag_name_03"
10	"tag_name_02"
11	"tag_name_02"
12	"tag_name_04"

Showing 1 to 12 of 12 entries

```
SELECT DISTINCT ?tag_name
WHERE {
  ?trackID track_artist artistID .
  ?trackID track_tag ?tag_name
}
```

使用ORDER BY 排序

ORDER BY DESC(?tag_name)

```
1 PREFIX music: <http://kg.course/music/>
2
3 SELECT DISTINCT ?tag_name
4 WHERE {
5   ?trackID music:track_artist music:artist_001 .
6   ?trackID music:track_tag ?tag_name
7 }
8
9 ORDER BY ?tag_name
10
```

那么ORDER BY DESC(?tag_name) 就是逆排序

	tag_name
1	"tag_name_02"
2	"tag_name_03"
3	"tag_name_04"
4	"tag_name_05"
5	"tag_name_06"
6	"tag_name_08"

Showing 1 to 6 of 6 entries

□ 查询某几类歌曲标签中的歌曲数目

```
SELECT (COUNT(?trackID) AS ?num)
WHERE {
  {
    ?trackID track_tag tag_name .
  }
  UNION
  {
    ?trackID track_tag tag_name2 .
  }
}
```

使用UNION联合查询

```
1 PREFIX music: <http://kg.course/music/>
2
3 SELECT (COUNT(?trackID) AS ?num)
4 WHERE {
5   {
6     ?trackID music:track_tag "tag_name_01" .
7   }
8   UNION
9   {
10    ?trackID music:track_tag "tag_name_02" .
11  }
12 }
13
14
```

QUERY RESULTS	
Table	Raw Response
Showing 1 to 1 of 1 entries	
num	
1	"85"^^xsd:integer

□ 查询某几类歌曲标签中的歌曲数目

```
SELECT (count(?trackID) as ?num)
WHERE {
  ?trackID track_tag ?tag_name
  FILTER (?tag_name = tag_name1 ||
    ?tag_name = tag_name2)
}
```

使用FILTER达到同样的效果

```
1- PREFIX music: <http://kg.course/music/>
2- |
3- SELECT (count(?trackID) as ?num)
4- WHERE {
5- ?trackID music:track_tag ?tag_name
6- FILTER (?tag_name = 'tag_name_01' || ?tag_name = 'tag_name_02')
7- }
8- .
```

QUERY RESULTS

num
85

□ 询问是否存在带有'xx'字符的歌曲名

```
ASK
{
  ?trackID track_name ?track_name .
  FILTER regex(?track_name,"xx")
}
```

使用ASK来询问是否存在

ASK的回答只有True或False

```
1- PREFIX music: <http://kg.course/music/>
2- |
3- ASK
4- {
5- ?trackID music:track_name ?track_name .
6- FILTER regex(?track_name,"008")
7- }
8- .
```

QUERY RESULTS

True

□ 给艺术家id新增属性艺术家名字

INSERT DATA

```
{
  artistID artist_name artist_name .
}
```

```
1- PREFIX music: <http://kg.course/music/>
2- |
3- INSERT DATA
4- {
5- music:artist_01 music:artist_name 'artist_name_01' .
6- music:artist_02 music:artist_name 'artist_name_02' .
7- music:artist_03 music:artist_name 'artist_name_03' .
8- }
9- .
```

QUERY RESULTS

```
<?xml?>
<?xml:namespace prefix='music' uri='http://kg.course/music/'?>
<update succeeded='true'/?>
</update succeeded?>
</?xml?>
```

我们使用INSERT DATA的方式，对artist_01,artist_02,artist_03新增了artist_name属性

执行结果显示更新成功

❑ 删除增加的属性艺术家名字

```
DELETE
{
  artistID artist_name ?x .
}

WHERE
{
  artistID artist_name ?x .
}
```

使用
WHERE定位
DELETE删除



开源图数据库

RDF4i

它是处理RDF数据的Java框架。使用简单可用的API来实现RDF存储。支持SPARQL 查询和两种RDF存储机制，支持所有主流的RDF格式。

aStore

aStore从图数据库角度存储和检索RDF知识图谱数据。

aStore支持W3C定义的SPARQL 1.1标准,包括含有Union,OPTIONAL,FILTER和聚集函数的查询;aStore支持有效的增删改操作。

gStore单机可以支持1Billion(十亿)三元组规模的RDF知识图谱的数据管理任务。

商业图数据库介绍

Virtuoso

智能数据，可视化与整合。

可扩展和高性能数据管理，支持Web扩展和安全

Allegrograph

AllegroGraph是一个现代的高性能的，支持永久存储的图数据库。

它基于Restful接入支持多语言编程。具有强大的加载速度、查询速度和高性能。

Stardog

原生图数据库

Neo4j

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。它是一个嵌入式的、基于磁盘的、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。内置Cypher 查询语言。

Neo4j具有以下特性：

图数据库 + Lucene索引

支持图属性

支持ACID

高可用性

支持320亿的结点,320亿的关系结点,640亿的属性

RESR API接口

注：7473端口:https前端服务；7474端口：http前端服务；
7687端口：bolt端口，后台数据库连接服务

1. Cypher CREATE语句，为每一条数据写一个CREATE
2. Cypher [LOAD CSV](#) 语句，将数据转成CSV格式，通过LOAD CSV读取数据
3. 官方提供的Java API — [Batch Inserter](#)
- 4.官方提供的 [neo4j-import](#) 工具
5. 第三方开发者编写的 [Batch Import](#) 工具

OrientDB

Titan