**第十七章 Neo4j图数据库应用开发**

大数据时代，现实世界的数字映像中，实体之间的关系是非常重要的一面，它们构成了巨大的网络，例如社交网络，网页之间的链接网络，互联网的路由网络，市场主体之间商品与劳务的交易网络等等，所以基于图论的网络分析已经成为大数据基础设施与落地应用的一个重要的领域。

在小型网络分析中，比如发票交易网络数据，[igraph](https://igraph.org/)等是很好用的分析工具，但它们有比较大的限制：无法分析巨型网络，也无法并行处理，并且要自己写各种图分析算法。从原理上看这个任务显然是传统RDBMS不能胜任的，而基于图论的图数据库则可以有效克服这些问题，并且提供了简易的类SQL操作语言与各种编程语言的API，简化了各种网络的分析。

[Neo4j](https://neo4j.com/)是这个领域的全球领先者，用Java语言编写，提供了类SQL的[Cypher](https://neo4j.com/docs/cypher-manual/current/)脚本语言操作，提供Java/Python/R/C++等各种语言的API，并且有[APOC](https://github.com/neo4j-contrib/neo4j-apoc-procedures)与[GDS](https://github.com/neo4j/graph-data-science)扩展库提供各种图算法。Noe4j有收费的商用企业版，提供了并行处理的能力，开源免费的社区版不支持并行处理，单机开发版Neo4j Desktop有企业版的所有功能，但只支持单机开发运行，它的免费授权是一年。同时它支持Linux与Windows等各个平台的部署，非常全面。国内已经出版了一些Neo4j的专著，如[《Neo4j权威指南》](http://www.tup.tsinghua.edu.cn/booksCenter/book_07389801.html)、[《精通Neo4j》](https://baike.baidu.com/item/%E7%B2%BE%E9%80%9ANeo4j/62286844?fr=ge_ala)等，已有不少行业引进应用，并且[汉化版](https://we-yun.com/)也作了一些扩充（http://we-yun.com/）。国内各大云厂商也在开发自己的图数据库产品，不过与Neo4j可能还有一点距离，毕竟人家已经持续投入研发二十多年，全球领先是有道理的，全球很多大公司都在用。所以在研究阶段先用领先的开源产品探索出可行的解决方案，梳理清楚落地应用的需求，以促进国内产品的研发，仍然是条可行之道。毕竟首先是要解决问题，然后才是国产化与能者居之。

图数据库的优点是规范简易的分析巨型网络，igraph的优点是非常灵活，一切尽在掌握，所以用图数据库找出目标子网后，再用igraph作深入的分析，是个不错的组合。

第十一章《Ubuntu18 Java语言环境安装配置》第二节《Neo4j安装配置》已经介绍了它的安装配置，本章将通过一个机场航线网络实例介绍一下它的开发与应用。

**第一节 Neo4j数据加载**

一、数据加载

本例的数据集来自 [Neo4j GDS的教学案例](https://github.com/neo4j-graph-examples/airport-routes)，它记录了世界各城市机场之间的航线网络，下面是其中的节点（实体）类型和关系模型图，关系HAS\_ROUTE记录了机场之间的航线，边的权重就是机场之间的距离，所以它和发票交易网络一样，是个加权有向图。一般通过[Neo4j Browser](https://neo4j.com/docs/browser-manual/current/)来操作Neo4j图数据库。

本章用的是Neo4j 5.10.0-CHS及APOC 5.11.0、GDS 2.5.5，注意不同版本的语法可能会稍有不同，比如GDS不同版本的图算法，名称和参数可能会有不同，读者可能要根据文档修订相关的Cypher脚本。

数十万行以上的数据，用CREATE OR REPLACE DATABASE建立一个空的数据库后，再用neo4j-admin import批量导入比较快，因为不涉及事务的处理。小批量的数据可以用LOAD CSV命令从本地文件系统导入，但它涉及事务操作，速度比较慢，本例属于小数据集，用LOAD CSV加载。

// https://github.com/neo4j-graph-examples

// https://github.com/neo4j-graph-examples/airport-routes

// https://github.com/neo4j-graph-examples/airport-routes/blob/main/README.adoc

//装入机场航线网络数据

CREATE CONSTRAINT airports IF NOT EXISTS FOR (a:Airport) REQUIRE a.iata IS UNIQUE;

CREATE CONSTRAINT cities IF NOT EXISTS FOR (c:City) REQUIRE c.name IS UNIQUE;

CREATE CONSTRAINT regions IF NOT EXISTS FOR (r:Region) REQUIRE r.name IS UNIQUE;

CREATE CONSTRAINT countries IF NOT EXISTS FOR (c:Country) REQUIRE c.code IS UNIQUE;

CREATE CONSTRAINT continents IF NOT EXISTS FOR (c:Continent) REQUIRE c.code IS UNIQUE;

WITH

'https://raw.githubusercontent.com/neo4j-graph-examples/graph-data-science2/main/data/airport-node-list.csv'

AS url

LOAD CSV WITH HEADERS FROM url AS row

MERGE (a:Airport {iata: row.iata})

MERGE (ci:City {name: row.city})

MERGE (r:Region {name: row.region})

MERGE (co:Country {code: row.country})

MERGE (con:Continent {name: row.continent})

MERGE (a)-[:IN\_CITY]->(ci)

MERGE (a)-[:IN\_COUNTRY]->(co)

MERGE (ci)-[:IN\_COUNTRY]->(co)

MERGE (r)-[:IN\_COUNTRY]->(co)

MERGE (a)-[:IN\_REGION]->(r)

MERGE (ci)-[:IN\_REGION]->(r)

MERGE (a)-[:ON\_CONTINENT]->(con)

MERGE (ci)-[:ON\_CONTINENT]->(con)

MERGE (co)-[:ON\_CONTINENT]->(con)

MERGE (r)-[:ON\_CONTINENT]->(con)

SET a.id = row.id,

a.icao = row.icao,

a.city = row.city,

a.descr = row.descr,

a.runways = toInteger(row.runways),

a.longest = toInteger(row.longest),

a.altitude = toInteger(row.altitude),

a.location = point({latitude: toFloat(row.lat), longitude: toFloat(row.lon)});

create index for (n:Airport) on (n.location);

LOAD CSV WITH HEADERS FROM 'https://raw.githubusercontent.com/neo4j-graph-examples/graph-data-science2/main/data/iroutes-edges.csv' AS row

MATCH (source:Airport {iata: row.src})

MATCH (target:Airport {iata: row.dest})

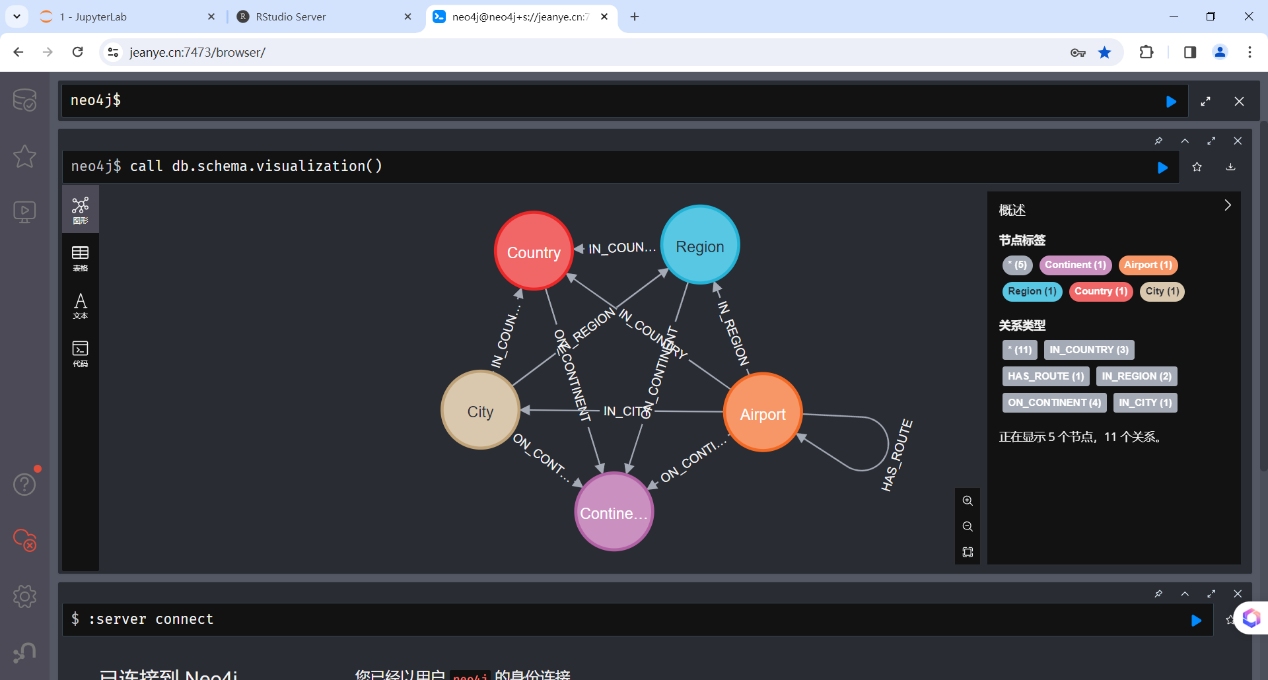
MERGE (source)-[r:HAS\_ROUTE]->(target)

ON CREATE SET r.distance = toInteger(row.dist);

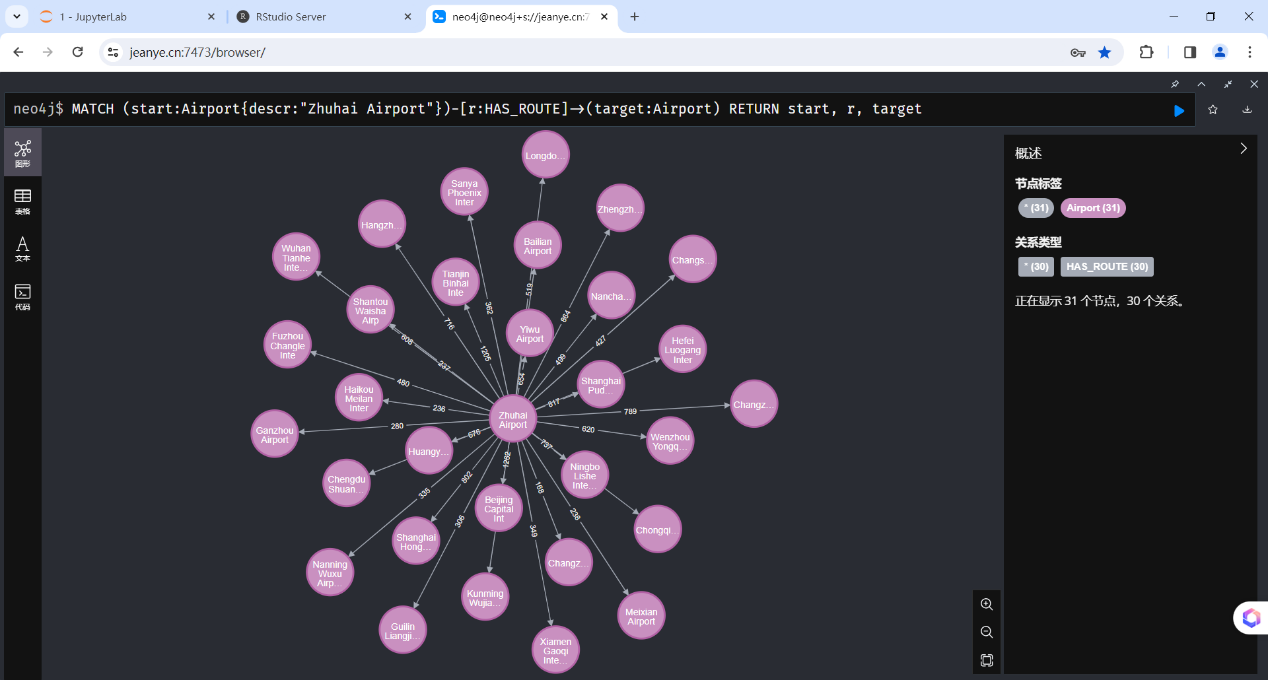
二、数据模型

在Neo4j Browser中输入命令，查看一下实体-关系的数据模型图：

call db.schema.visualization()



查看一下珠海机场出发的航班，验证数据正确加载：



MATCH (start:Airport{descr:"Zhuhai Airport"})-[r:HAS\_ROUTE]->(target:Airport)

RETURN start, r, target

**第二节 Python API连接Neo4j**

一、用Cypher调用图算法

上一节已经用简单的Cypher语句来加载数据及了解数据，本节先编写好查询机场航线的Cypher语句，可以用外置的文本编辑器编辑，然后拷贝粘贴到Neo4j Browser中运行。在调用GDS图算法前，要先建立参与运算子图的投影，并加载到内存中。下面的语句建立了国内机场（Country节点的code=”CN”）子网的投影。

//建立子图投影，后面的GDS图算法要通过名称引用它，重启Neo4j后将自动从内存中删除。

CALL gds.graph.exists('airportsNetwork\_CN')

YIELD graphName, exists

WHERE NOT exists

CALL gds.graph.project.cypher(

'airportsNetwork\_CN',

'MATCH (n:Airport)-[:IN\_COUNTRY]->(c:Country{code:"CN"})

RETURN id(n) AS id',

'MATCH (startNode:Airport)-[r:HAS\_ROUTE]->(targetNode:Airport),

(startNode:Airport)-[:IN\_COUNTRY]->(c:Country{code:"CN"}),

(targetNode:Airport)-[:IN\_COUNTRY]->(c:Country{code:"CN"})

RETURN id(startNode) AS source, id(targetNode) AS target, r.distance AS distance'

) YIELD

graphName AS graph, nodeQuery, nodeCount AS nodes, relationshipQuery, relationshipCount AS rels

RETURN graphName, nodes, rels;

//如果不需要了，可以用这个语句删除内存中的子图投影

CALL gds.graph.drop( 'airportsNetwork\_CN');

这个子网有217个机场2699条航线：

╒════════════════════╤═══════╤══════╕

│"graphName" │"nodes" │"rels" │

╞════════════════════╪═══════╪══════╡

│"airportsNetwork\_CN" │217 │2699 │

└────────────────────┴───────┴──────┘

再调用GDS的图算法，查找从某机场出发到达其它机场的所有最短路径。

//单源最短路径算法 国内航线，length(path)<=3，3跳以内，出发机场为"Zhuhai Airport"。

CALL {

MATCH (start:Airport{descr:"Zhuhai Airport"})

CALL gds.allShortestPaths.delta.stream('airportsNetwork\_CN',

{ sourceNode:start,

relationshipWeightProperty:'distance',

delta: 3.0 })

YIELD targetNode AS nodeId, totalCost AS distance, path

RETURN nodeId, distance, path

ORDER BY distance

//跳过第一条，是自己

SKIP 1

//LIMIT 10;

}

WITH path

WHERE length(path)<=3

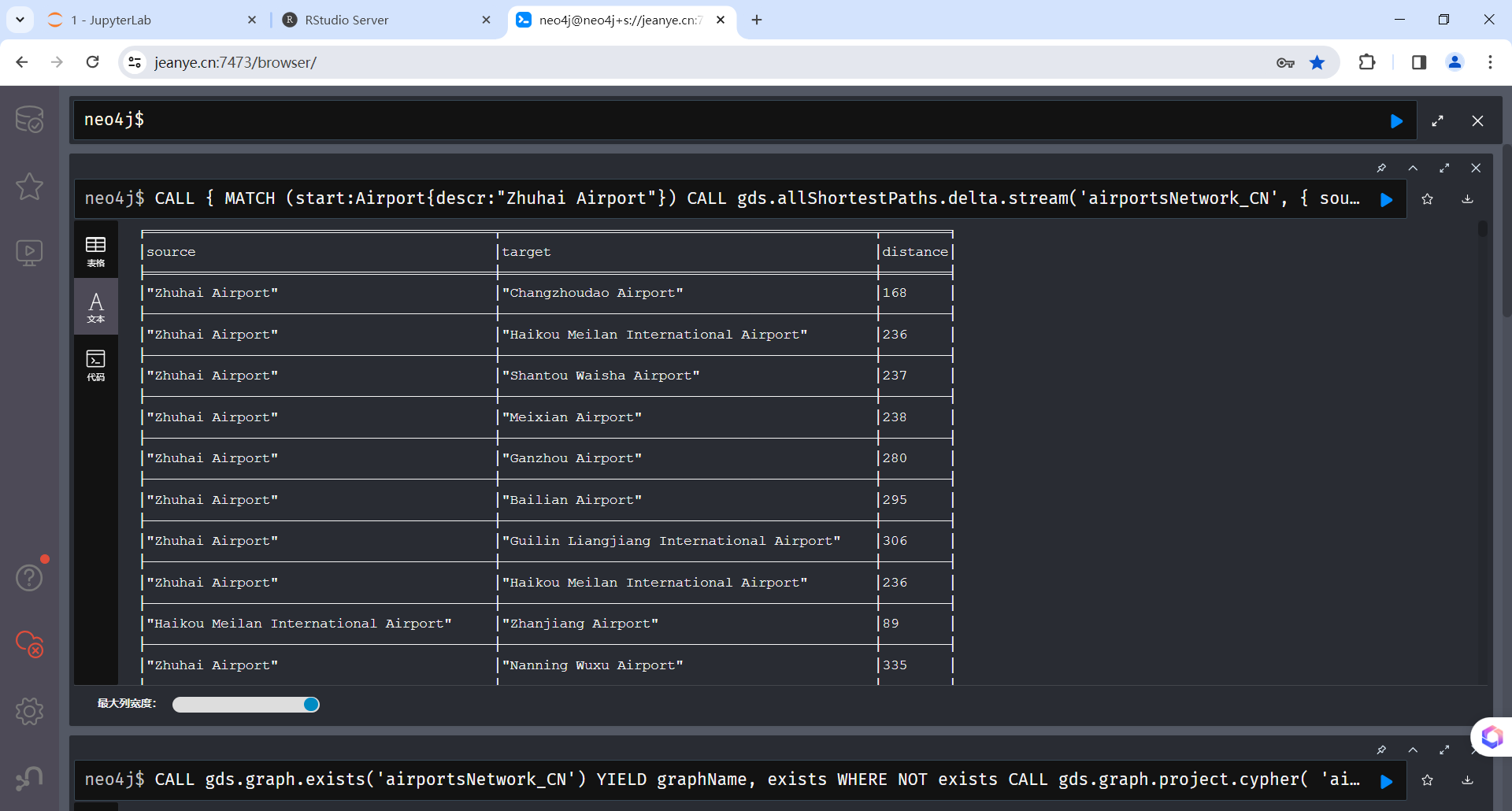
WITH nodes(path) AS pnodes

UNWIND range(0, size(pnodes)-1) AS index

WITH pnodes[index] AS current, pnodes[index+1] AS next

MATCH (current)-[r:HAS\_ROUTE]->(next)

RETURN current.descr AS source, next.descr as target, r.distance AS distance;



下面的Cypher语句查询从某机场出发的稳定环路。虚开增值税发票的一种情形是，从虚开的企业出发，资金绕了不长的一圈后会回流该企业，形成稳定的环路，其资金流量就是边的权重。这里用两个参数来过滤稳定环路，threshold表示平均值，平均值太小的不予查看。amplitude表示波幅，波幅太大的不是稳定环路，也不予查看。第3个参数是跳数，这种图路径算法要遍历所有的路径，比较耗时，在稠密的网络中需要控制环路最大的跳数，不能太大，否则算法跑不出来。这里用机场替代企业，机场间航线的距离替代资金流量，但算法与程序是通用的。机场航线网络是稠密的网络，最大跳数就小一点，发票交易网络是稀疏的网络，跳数就可以稍大一点。后面会开发一个自定义函数，用中位数替代平均值，这样更准确一点。

//从珠海机场出发3跳以内的稳定环路，环路平均距离500公里以上，波幅<50% 。

MATCH (china:Airport)-[:IN\_COUNTRY]->(c:Country{code:"CN"})

WITH COLLECT(china.id) AS targets

MATCH path = (source:Airport{descr:"Zhuhai Airport"})-[r:HAS\_ROUTE\*2..3]->(target)

WHERE target.descr = source.descr

AND ALL( node IN nodes(path) where node.id IN targets)

WITH path, [r in relationships(path)|r.distance] AS distances, 0.5 AS amplitude, 500 AS threshold

WITH path, apoc.coll.avg(distances) AS avgDist, apoc.coll.min(distances) as minDist, apoc.coll.max(distances) as maxDist

WHERE minDist/avgDist >= (1-amplitude) AND maxDist/avgDist <= (1+amplitude) AND avgDist >= threshold

//WITH path, apoc.coll.median(distances) AS medianDist, apoc.coll.min(distances) as minDist, apoc.coll.max(distances) as maxDist

//WHERE minDist/medianDist >= (1-amplitude) AND maxDist/medianDist <= (1+amplitude) AND medianDist >= threshold

WITH path, relationships(path) as flights

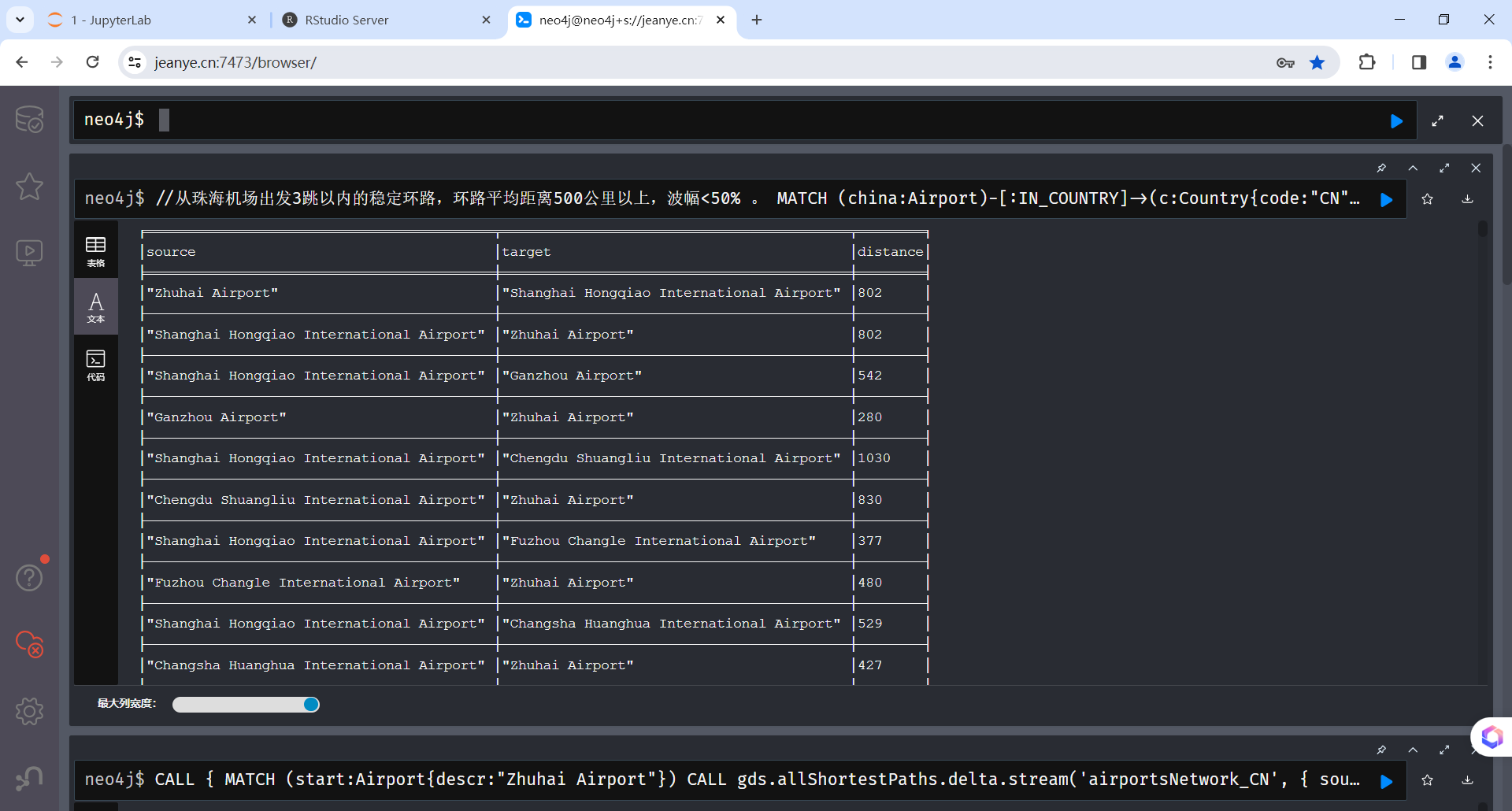
UNWIND flights AS flight

WITH DISTINCT flight

MATCH (n:Airport)-[flight]->(m:Airport)

RETURN n.descr AS source, m.descr as target, flight.distance AS distance;

运行结果，每行是一条边，当source=target时，一个环路结束：



下面这个Cypher语句返回从某机场出发指定跳数以内的航线子网。

//返回从珠海机场出发3跳以内的国内航线完全航线子网

MATCH (target:Airport)-[:IN\_COUNTRY]->(c:Country{code:'CN'})

WITH target

MATCH path = (start:Airport{descr:"Zhuhai Airport"})-[r:HAS\_ROUTE\*1..3]->(target)

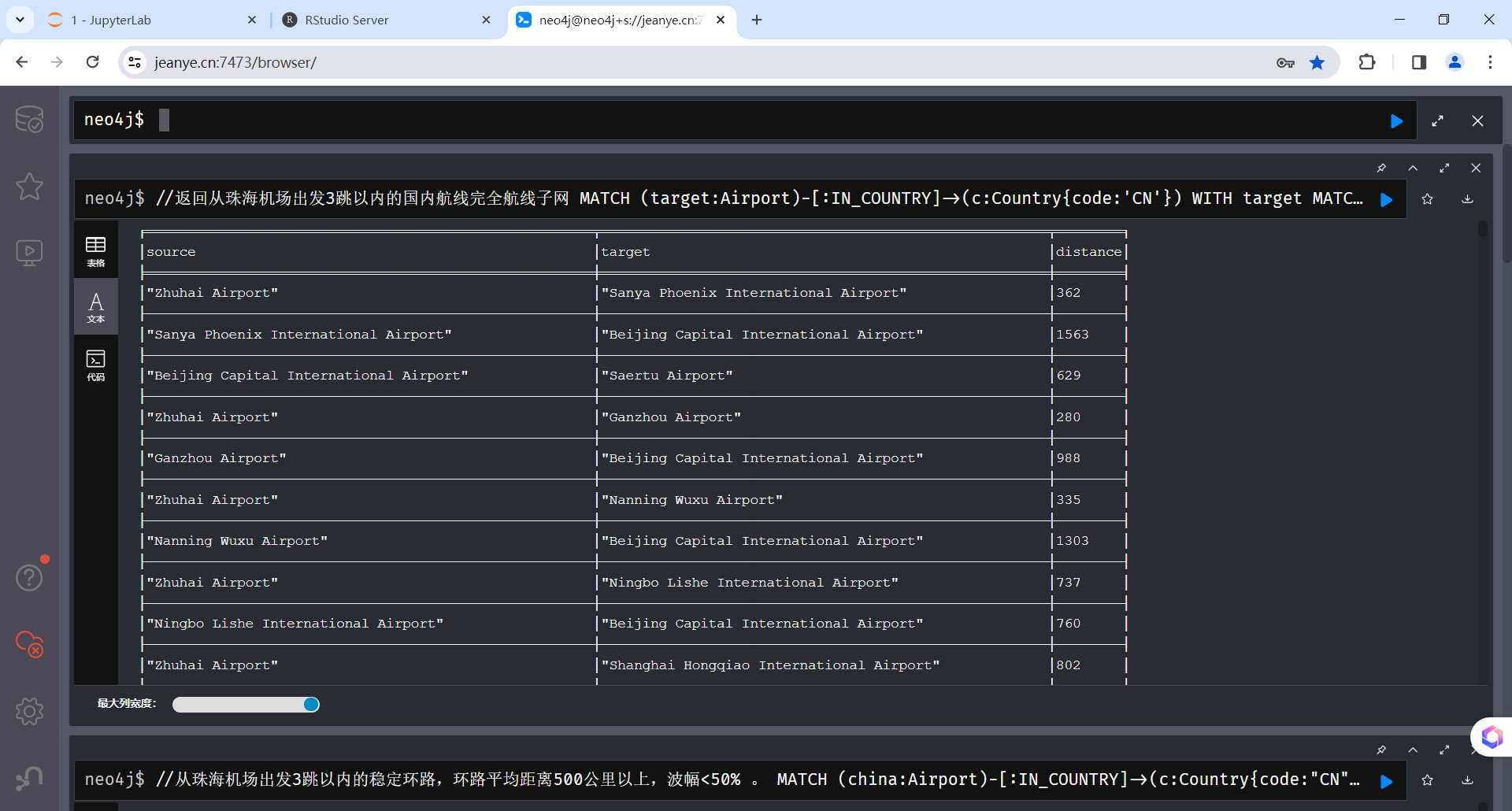
WITH \*, relationships(path) as flights

UNWIND flights AS flight

WITH DISTINCT flight

MATCH (n:Airport)-[flight]->(m:Airport)

RETURN n.descr AS source, m.descr as target, flight.distance AS distance;



二、用Python API连接Neo4j

现在可以编写Python脚本访问Neoe4j，调用上面的Cypher语句，查询从某机场出发指定跳数以内稳定环路等。Neo4j有Java、Python、R、JavaCsript等的[API Driver](https://github.com/neo4j)，其中R语言Dirver是社区贡献的，还不够稳定，这里用[Python Driver](https://github.com/neo4j/neo4j-python-driver)连接，注意Driver的版本要与Neo4j的版本匹配，[参阅资料](https://neo4j.com/developer/kb/neo4j-supported-versions/)。安装请参阅其主页：

pip install neo4j

下面的Python程序neo4jAirports.py会在后面的Shiny APP中调用，有几个地方要说明一下：

1、Neo4j数据库的协议、网址，用户名/口令在这一行指定，连接协议bolt+ssc是自签证书SSL连接专用，不会验证证书的有效性，但会建立加密连接。用户名和密码另加一个模块neo4jkeys.py定义，是为了使用和发布上的方便。

driver = GraphDatabase.driver("bolt+ssc://localhost:7687", auth=basic\_auth(neo4jkeys.user, neo4jkeys.password))

2、函数getAirLines(source, length)调用path\_to\_airports(tx, source,length)，执行了一个40行的Cypher大查询，返回从节点source出发，length跳以内，所有通达节点的最短路径。getRings(source, length, amplitude, threshold)调用airports\_ring(tx, source, length, amplitude, threshold)返回给定机场的稳定强环路，它将用于后面的3D可视化演示。

3、函数getAirports(country)调用countryAirports(tx, country)执行一个Cypher查询，返回指定国家内有出发航线的所有机场。该数据集不完整，有些机场没有出发航线，会引起Shiny APP报错，要排除。

4、Cypher语句中可以通过$varname占位符定义参数，然后在调用时用tx.run(CypherSQL, varname=value)替换成具体的值，比较方便。

import pandas as pd

from neo4j import GraphDatabase, basic\_auth

from neo4j import neo4jkeys

# bolt： 无SSL

# bolt+s :SSL并检查证书链

# bolt+ssc: SSL但不检查证书链，自签证书专用

# 同样适用于neo4j协议, neo4j+s，neo4j+ssc

# https://neo4j.com/docs/python-manual/current/client-applications/

# # GDS升级到2.5.5，算法调用有变化，已更正, Neo4j 升级到5.10.0

# 只返回最短路径

def getAirLines(source, length):

driver = GraphDatabase.driver("bolt+ssc://localhost:7687", auth=basic\_auth(neo4jkeys.user, neo4jkeys.password))

with driver.session(database="neo4j") as session:

graph = session.execute\_read(path\_to\_airports, source,length)

driver.close()

return graph

# GDS升级到2.5.5，算法调用有变化，已更正

def path\_to\_airports(tx, source, length):

# 先建立图的投影

cypher = '''

CALL gds.graph.exists('airportsNetwork\_CN')

YIELD graphName, exists

WHERE NOT exists

CALL gds.graph.project.cypher(

'airportsNetwork\_CN',

'MATCH (n:Airport)-[:IN\_COUNTRY]->(c:Country{code:"CN"})

RETURN id(n) AS id',

'MATCH (startNode:Airport)-[r:HAS\_ROUTE]->(targetNode:Airport),

(startNode:Airport)-[:IN\_COUNTRY]->(c:Country{code:"CN"}),

(targetNode:Airport)-[:IN\_COUNTRY]->(c:Country{code:"CN"}) RETURN id(startNode) AS source, id(targetNode) AS target, r.distance AS distance'

) YIELD

graphName AS graph, nodeQuery, nodeCount AS nodes, relationshipQuery, relationshipCount AS rels

RETURN graphName, nodes, rels;

'''

result = tx.run(cypher).single()

# 再调用图算法

cypher = '''

CALL {

MATCH (start:Airport{descr:$name})

CALL gds.allShortestPaths.delta.stream('airportsNetwork\_CN',

{ sourceNode:start,

relationshipWeightProperty:'distance',

delta: 3.0 })

YIELD targetNode AS nodeId, totalCost AS distance, path

RETURN nodeId, distance, path

ORDER BY distance

//跳过第一条，是自己

SKIP 1

//LIMIT 10;

}

WITH path

WHERE length(path)<=$len

WITH nodes(path) AS pnodes

UNWIND range(0, size(pnodes)-1) AS index

WITH pnodes[index] AS current, pnodes[index+1] AS next

MATCH (current)-[r:HAS\_ROUTE]->(next)

RETURN current.descr AS source, next.descr as target, r.distance AS distance;

'''

start = []; target=[]; distance=[]

for record in tx.run(cypher, name=source, len=length):

start.append(record["source"])

target.append(record["target"])

distance.append(record["distance"])

# 从 list建立 pandas data frame

data = pd.DataFrame({"source":start,"target":target,"distance":distance})

return data

# 返回完全子网

def getAirLines2(source, length):

driver = GraphDatabase.driver("bolt+ssc://localhost:7687", auth=basic\_auth(neo4jkeys.user, neo4jkeys.password))

with driver.session(database="neo4j") as session:

graph = session.execute\_read(airports\_network, source,length)

driver.close()

return graph

# Neo4j 不允许将可变长路径中的最大长度作为参数，这里用变通的办法绕过，把长度参数转换为字符串，先拼接成固定的Cypher语句。

# 另一个参数source则通过占位符在执行时动态替换

# 参阅：

# https://stackoverflow.com/questions/42386273/cypher-query-to-give-path-length-as-a-parameter-for-variable-length-relationship

# https://stackoverflow.com/questions/42668225/neo4j-pass-parameter-to-variable-length-relationship

# https://blog.csdn.net/xckkcxxck/article/details/79691594

def airports\_network(tx, source,length):

# 先建立图的投影

cypher = '''

CALL gds.graph.exists('airportsNetwork\_CN')

YIELD graphName, exists

WHERE NOT exists

CALL gds.graph.project.cypher(

'airportsNetwork\_CN',

'MATCH (n:Airport)-[:IN\_COUNTRY]->(c:Country{code:"CN"})

RETURN id(n) AS id',

'MATCH (startNode:Airport)-[r:HAS\_ROUTE]->(targetNode:Airport),

(startNode:Airport)-[:IN\_COUNTRY]->(c:Country{code:"CN"}),

(targetNode:Airport)-[:IN\_COUNTRY]->(c:Country{code:"CN"}) RETURN id(startNode) AS source, id(targetNode) AS target, r.distance AS distance'

) YIELD

graphName AS graph, nodeQuery, nodeCount AS nodes, relationshipQuery, relationshipCount AS rels

RETURN graphName, nodes, rels;

'''

result = tx.run(cypher).single()

# 再调用图算法

cypher = (\

'''

MATCH (target:Airport)-[:IN\_COUNTRY]->(c:Country{code:'CN'})

WITH target

MATCH path = (start:Airport{descr:$name})-[r:HAS\_ROUTE\*1..'''+str(int(length))+"]->(target)"+\

'''

WITH \*, relationships(path) as flights

UNWIND flights AS flight

WITH DISTINCT flight

MATCH (n:Airport)-[flight]->(m:Airport)

RETURN n.descr AS source, m.descr as target, flight.distance AS distance;

'''

)

# print(cypher)

start = []; target=[]; distance=[]

for record in tx.run(cypher, name=source):

start.append(record["source"])

target.append(record["target"])

distance.append(record["distance"])

# 从 list建立 pandas data frame

data = pd.DataFrame({"source":start,"target":target,"distance":distance})

return data

def getRings(source, length, amplitude, threshold):

driver = GraphDatabase.driver("bolt+ssc://localhost:7687", auth=basic\_auth(neo4jkeys.user, neo4jkeys.password))

with driver.session(database="neo4j") as session:

graph = session.execute\_read(airports\_ring, source, length, amplitude, threshold)

driver.close()

return graph

def airports\_ring(tx, source, length, amplitude, threshold):

# 先建立图的投影

cypher = '''

CALL gds.graph.exists('airportsNetwork\_CN')

YIELD graphName, exists

WHERE NOT exists

CALL gds.graph.project.cypher(

'airportsNetwork\_CN',

'MATCH (n:Airport)-[:IN\_COUNTRY]->(c:Country{code:"CN"})

RETURN id(n) AS id',

'MATCH (startNode:Airport)-[r:HAS\_ROUTE]->(targetNode:Airport),

(startNode:Airport)-[:IN\_COUNTRY]->(c:Country{code:"CN"}),

(targetNode:Airport)-[:IN\_COUNTRY]->(c:Country{code:"CN"}) RETURN id(startNode) AS source, id(targetNode) AS target, r.distance AS distance'

) YIELD

graphName AS graph, nodeQuery, nodeCount AS nodes, relationshipQuery, relationshipCount AS rels

RETURN graphName, nodes, rels;

'''

result = tx.run(cypher).single()

# 再调用图算法

cypher = (\

'''

MATCH (china:Airport)-[:IN\_COUNTRY]->(c:Country{code:"CN"})

WITH COLLECT(china.id) AS targets

MATCH path = (source:Airport{descr:$name})-[r:HAS\_ROUTE\*2..'''+str(int(length))+"]->(target)"+\

'''

WHERE target.descr = source.descr

AND ALL( node IN nodes(path) where node.id IN targets)

WITH path, [r in relationships(path)|r.distance] AS distances, $amplitude AS amplitude, $threshold AS threshold

//WITH path, apoc.coll.avg(distances) AS avgDist, apoc.coll.min(distances) as minDist, apoc.coll.max(distances) as maxDist

//WHERE minDist/avgDist >= (1-amplitude) AND maxDist/avgDist <= (1+amplitude) AND avgDist >= threshold

WITH path, apoc.coll.median(distances) AS medianDist, apoc.coll.min(distances) as minDist, apoc.coll.max(distances) as maxDist

WHERE minDist/medianDist >= (1-amplitude) AND maxDist/medianDist <= (1+amplitude) AND medianDist >= threshold

WITH path, relationships(path) as flights

UNWIND flights AS flight

WITH DISTINCT flight

MATCH (n:Airport)-[flight]->(m:Airport)

RETURN n.descr AS source, m.descr as target, flight.distance AS distance;

'''

)

print(cypher)

start = []; target=[]; distance=[]

for record in tx.run(cypher, name=source, amplitude=amplitude, threshold=threshold ):

start.append(record["source"])

target.append(record["target"])

distance.append(record["distance"])

# 从 list建立 pandas data frame

data = pd.DataFrame({"source":start,"target":target,"distance":distance})

return data

# 返回国内机场列表

def getAirports(country):

driver = GraphDatabase.driver("bolt+ssc://localhost:7687", auth=basic\_auth(neo4jkeys.user, neo4jkeys.password))

with driver.session(database="neo4j") as session:

# graph = session.read\_transaction(countryAirports, country) # Neo4j 5.16.0

graph = session.execute\_read(countryAirports, country)

driver.close()

return graph

def countryAirports(tx, country):

cypher="match (n:Airport)-[:IN\_COUNTRY]->(c:Country{code:$code}), \

(n:Airport)-[:HAS\_ROUTE]->(target) return distinct n.descr as name"

airports = []

for record in tx.run(cypher, code=country):

airports.append(record["name"])

# 从 list建立 pandas data frame

data = pd.DataFrame({"airport":airports})

return data

# source = "Zhuhai Airport"

# graph = getAirLines(source,2)

# graph2 = getAirLines2(source,2)

# graph3 = getRings(source,2,0.2,500)

airports = getAirports("CN")

Rstudio中运行测试一下，运行通过：

> reticulate::source\_python('~/scripts/neo4jAirports.py')

> source = "Zhuhai Airport"

> graph = getAirLines(source,2)

> graph[1:10,]

source target distance

1 Zhuhai Airport Changzhoudao Airport 168

2 Zhuhai Airport Haikou Meilan International Airport 236

3 Zhuhai Airport Shantou Waisha Airport 237

4 Zhuhai Airport Meixian Airport 238

5 Zhuhai Airport Ganzhou Airport 280

6 Zhuhai Airport Bailian Airport 295

7 Zhuhai Airport Guilin Liangjiang International Airport 306

8 Zhuhai Airport Haikou Meilan International Airport 236

9 Haikou Meilan International Airport Zhanjiang Airport 89

10 Zhuhai Airport Nanning Wuxu Airport 335

**第三节 Shiny 3D网络分析APP**

这个Shiny APP实现了在浏览器中3D可视化的分析探索，选择感兴趣的机场航线环路，返回给调用该APP的宿主网页。3D可视化可以旋转与透视，提供了更好的探索效果。另外还可以增加第4维时间维，做成动画，这就是4D了。还可以通过物体的颜色、大小、形状、纹理等增加更多的数据表现维度。

所有运行的代码加起来约800行，虽然只有一个可见的jsp页面，但实现了完整的多层应用体系，麻雀虽小，五脏俱全。一共用了R、Python、Java、Cypher、SQL、javascript、HTML/CSS等7种语言，集成了Shiny、Neo4j、Tomcat 3个平台，使用了Rstudio Server、JupyterLab、Eclipse、Neo4j Browser 4个IDE开发环境来开发，融会贯通，可谓集大成者，螺蛳壳里做道场。这说明每一种语言、工具和平台都有它的长处和用处，最佳实践是从解决问题出发，根据需要加以综合运用，不必拘束于语言与平台之争。[体验网址](https://jeanye.cn:8443/testNeo4j/indexRings.jsp)。

1. Tomcat Web APP

应用层实现服务器端的业务逻辑。因为是演示性质，本示例的应用层只有一个页面indexRings.jsp，并且非常简单，只是抽取了国内的机场列表。可以扩展一下获得传入参数之前在宿主业务系统中的前置业务过程（逻辑），以及获得传出结果之后的后续业务过程，比如规划航线上的航班票务，那就是完整的应用层业务逻辑。注意，这里用到了Neo4j的[Java Driver](https://github.com/neo4j/neo4j-java-driver)。

1、先直接用java连接Neo4j执行一个Cypher查询抽取网络中国内的机场列表，在jsp中生成一个下拉列表，供选择传入Shiny APP作为初始参数。

2、通过HTML5的postMessage()机制与嵌入的Shiny APP交换数据，收发的是字符串，多个参数也可以封装为JSON格式。

3、通过IFRAME嵌入Shiny APP。

涉及到Shiny APP UI及与父窗口通讯的html与javascript会嵌入Shiny APP的ui.R程序中，后面再介绍。

<%@ page language=*"java"* contentType=*"text/html; charset=GBK"*

pageEncoding=*"GBK"*%>

<%@ page import=*"java.util.List,java.util.ArrayList,org.neo4j.driver.\*,static org.neo4j.driver.Values.parameters"*%>

<%

//Driver driver = GraphDatabase.driver("bolt+ssc://106.52.33.185:7687", AuthTokens.basic("neo4j", "Jean@2022"));

Driver driver = GraphDatabase.driver("bolt+ssc://jeanye.cn:7687", AuthTokens.basic("neo4j", "password"));

Session neo4j\_session = driver.session();

Transaction tx = neo4j\_session.beginTransaction();

String cypherQuery = "match (n:Airport)-[:IN\_COUNTRY]->(c:Country{code:$country}), (n:Airport)-[:HAS\_ROUTE]->(target) return distinct n.descr as name; ";

List<Record> result = **new** ArrayList<Record>();

Result res = tx.run(cypherQuery, parameters("country", "CN"));

**while** (res.hasNext()) {

Record row = res.next();

result.add(row);

}

tx.commit();

tx.close();

**for** (Record record : result) {

System.out.println(record.get("name").asString());

}

driver.close();

%>

<!DOCTYPE html>

<html>

<head>

<meta charset=*"GBK"*>

<title>航线交通网络环路分析示例 3D</title>

<script type=*"text/javascript"*>

//接受消息

window.addEventListener('message', receiveMessage);

**function** receiveMessage(event) {

//alert(event.data);

**var** nodesSelected = document.getElementById('nodesSelected');

**var** message = event.data.trim();

nodesSelected.value = message;

**if** (message.length >30){

nodesSelected.size = message.length;

}**else**{

nodesSelected.size = 30;

}

}

//发送消息

**function** setAirport(){

**try**{

**var** airport = document.getElementById("airport");

//alert(airport.value);

// document.getElementById("shinyappframe") 失败。

**var** win = document.getElementsByTagName('iframe')[0].contentWindow;

// postMessage()，窗口间通信，不受浏览器同源规则限制，可用于任何内嵌的iframe。

// 第二个参数是接受窗口的源，协议+域名+端口，可用\*表示发给所有窗口。

win.postMessage(airport.value,'\*');

//alert("Sent!");

} **catch** (error) {

alert(error);

}

}

</script>

</head>

<body>

<table>

<tr><h1>国内机场航线环路分析示例 <a href=*"index.jsp"*>2D效果示例</a> <a href=*"index3D.jsp"*>3D效果示例</a></h1></tr>

<tr>

Shiny APP通过IFRAME嵌入网页的方案非常灵活，可以嵌入任何现存的传统WEB界面应用系统，从而扩充系统的功能，集成深度数据分析与展示的能力，盘活存量的软硬件与数据资产。<br>

但Shiny APP与嵌入的网页窗口之间要能够通过javascript在程序的层面互相操作（流程与数据），实现真正的集成连通，嵌入才有意义。<br>

浏览器有同源规则的安全限制，不同源的网页之间不能互相操作。Shiny APP与宿主网页一般是不同源的，即协议+域名+端口三项一般不完全一致，这里通过HTML5引入的跨文档通信 API（Cross-document messaging）来解决，这个API为window对象新增了一个window.postMessage方法，允许跨窗口通信，不论这两个窗口是否同源。<br/>

传入的参数可以用于为Shiny APP设定初始的运行状态，传出的参数可以向调用者报告APP运行的状态或结果，从而实现与嵌入环境的集成。<br/>

本示例先按《传入》按钮传入数据给嵌入的Shiny APP，记录调用的父窗口，然后才可以按《传出》按钮传出数据给父窗口。<br/>

通过Shiny.setInputValue(id, data)通知Shiny服务器端input变量的更新事件,Shiny的reactive与observe机制会被触发更新所有相关的变量与状态，但浏览器端的可视化控件，比如本例的selectInput等，是JS 的C3控件，不会自动更新，要由服务器端通知浏览器去更新，通过在observeEvent()中调用updateSelectInput(session, id, value = data)来完成。父窗口的页面源码包含了发消息给IFRAME中Shiny APP的javascript，可以在浏览器中查看页面的源码。<br>

本例中要打开“记录点击的结点”开关选择结点，然后点击《传出》按钮传给父窗口。因为可视化分析网络要缩放、拖动，这时点击结点就不是选中。<br/>

本例会自动高亮显示选中结点之间的边，并且会有动画效果，会有光子从各边的源运动到目标，光子的数量与速度显示整个子网承载对象的流量情况。单击一条边也会有动画效果。这是3D Force Directed Grap库与ThreeJS库的方便功能。动画增加了第4维时间维，还可以通过设定物体的颜色、大小、形状、纹理等来增加更多的视觉维度。<br>

如果有多重边，就要通过.linkCurvature(curv)设置曲率，比如.linkCurvature(0.25)，否则会合并显示为一条直线。设置曲率后缩放时要重新计算一次，计算量大（每条边100个点）有点卡，不过算好后旋转移动的速度就不受影响。<br>

查询环路的速度受网络稠密程度影响，航线网络比较稠密，本例中查询4跳以内的环路，10秒以内，5跳的话大约3分钟，其它网络没有这么稠密跳数可以多一点。<br>

<h3>国内机场列表</h3>

</tr>

<tr>

<td>

<form action=*"#"*>

<label for=*"airport"*>机场</label>

<select name=*"airport"* id=*"airport"*>

<% **for** (Record record : result) { %>

<option value=*"*<%=record.get("name").asString()%>*"*><%=record.get("name").asString()%></option>

<% }%>

</select>

<input type=*"button"* value=*"传入参数"* onclick="setAirport()" />

传出选中的结点：<input id=*"nodesSelected"* type=*"text"* value=*""* size=*30*></input>

</form>

</td>

</tr>

</table>

<iframe id=*"shinyappframe"* src=*"https://jeanye.cn/shiny/users/jean/airportsRings/"* style="border: *1px solid #AAA*; width: *100%*; height: *800px*"></iframe>

</body>

</html>

二、Shiny APP

由Neo4j图数据库通过Cypher查询语言返回图数据，为了方便R语言处理，这里返回边的[source,target, distance]列表组成的dataframe，igraph根据dataframe在内存中重建图的实例，再转换为3D Force-Directed Graph的json格式图数据，另一个查询返回国内机场列表，都是文本列表数据。

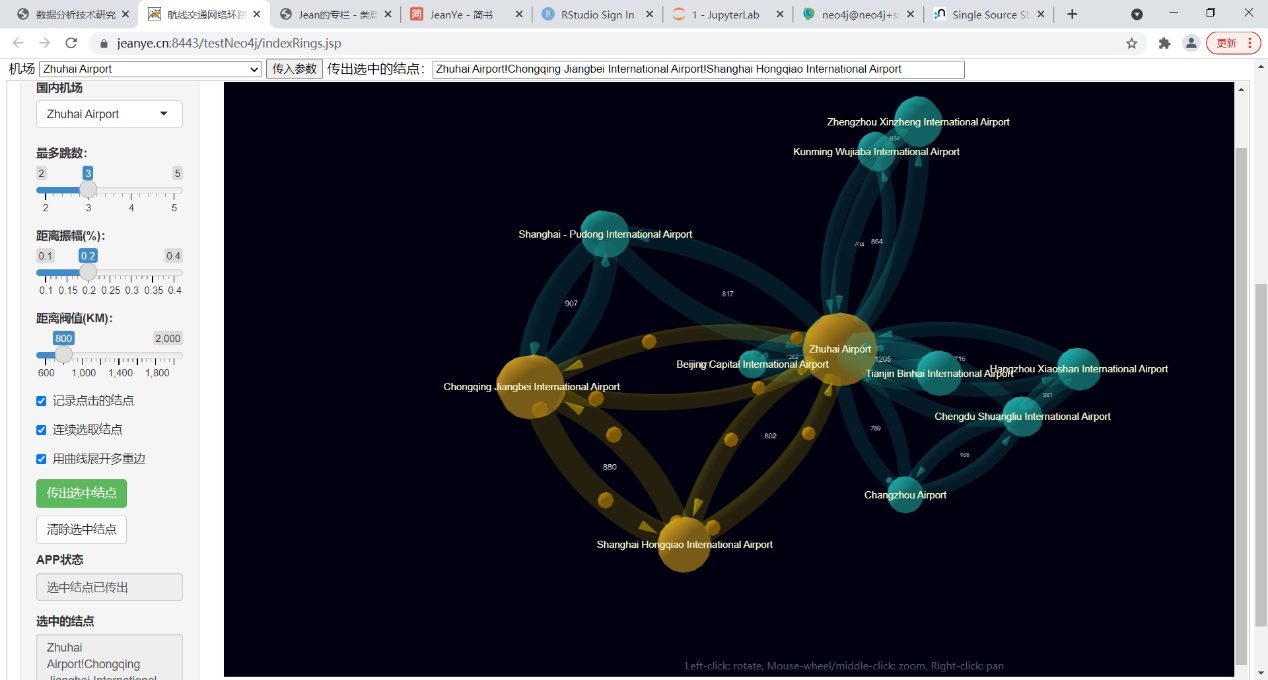
浏览器端使用比较流行的[ThreeJS](https://threejs.org/) 开源javascript 3D渲染（render）引擎，以及在它上面开发的有向图3D可视化开源项目 [3D Force-Directed Graph](https://github.com/vasturiano/3d-force-graph)，它的开发者是葡萄牙人[Vasco Asturiano](https://github.com/vasturiano)，Neo4j也在使用，更多的浏览器端开源可视化项目可以参考[这篇文章](https://awesomeopensource.com/projects/graph-visualization)。

[3D Force-Directed Graph](https://github.com/vasturiano/3d-force-graph)是纯粹的浏览器端JS库，没有直接支持Shiny Reactive编程的render()函数和配对的output()函数，通过Shiny的消息通信机制，由服务器在reactive input的响应函数中，直接发送数据消息回浏览器，然后由浏览器中注册的消息处理函数载入数据，用[3D Force-Directed Graph](https://github.com/vasturiano/3d-force-graph)的API生成浏览器端图的实例，来解决问题。Shiny的javascript通信机制可以参考[这篇文章](https://shiny.posit.co/r/articles/build/communicating-with-js/)。具体的API可以看[3D Force-Directed Graph的Github主页](https://github.com/vasturiano/3d-force-graph)。

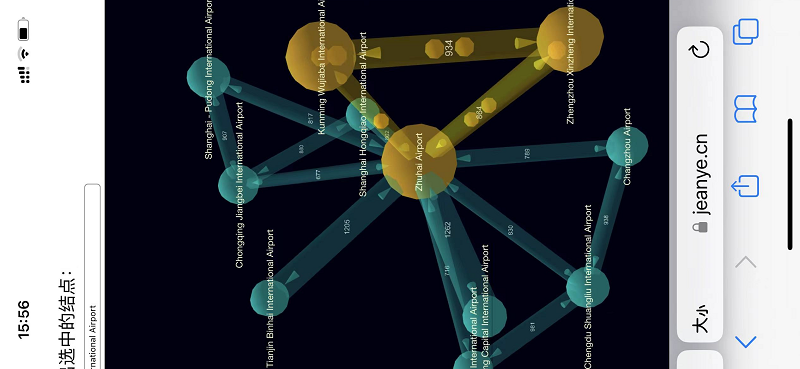
下面具体看看，先看看运行的效果，手机上运行也比较流畅，效果非常好。

1、运行效果

1）PC上运行效果



2）手机上运行效果



2、global.R

装入reticulate、igraph等引用的包，通过reticulate连接Python，然后装入定义了业务功能函数的Python脚本，最后调用Python函数getAirports()连接Neo4j得到国内的机场列表。

加载jsonlite库，以便将Cypher查询的结果（图）以json格式的文本返回给浏览器，由javascript处理。

library(reticulate)

library(sqldf)

library(igraph)

library(jsonlite)

# 确定连接Neo4j的Python脚本的目录位置

print(getwd())

location<- substr(getwd(),1,4)

# 调用Pyhton脚本连接 Neo4j, Neo4j的官方Python Driver比开源R Driver好用。

if (location != '/srv'){ # Rstudio 开发环境

part<- '~'

}else{ # 服务器部署环境

part<- '..'

}

path<- paste(part,"/scripts/","neo4jAirports.py",sep="")

print(path)

# 装入脚本

source\_python(path)

# 调用Python函数载入国内有出发航线数据的机场列表

airports<- getAirports("CN")

3、ui.R

用Shiny的各种UI组件定义了Shiny APP的UI，就是上面运行效果见到的样子，它们实际上是HTML Widget，绑定到浏览器上运行的各种javascript库，由R的htmlwidgets包支持。具体可参阅 [Shiny的API文档](https://shiny.posit.co/r/reference/shiny/)。

1）嵌入javascript处理与父窗口通过postMessage()机制交换数据。收到父窗口传入的机场名字参数时，调用Shiny的Shiny.setInputValue()函数通知服务器端输入变量的更新，再由reactive机制更新浏览器端的UI。

2）UI本身。Shiny APP的所有输入变量都在input列表下，所有输出变量都在output列表下，与server.R交叉引用这些输入输出变量，通过reactive()和observe()、observeEvent()机制监测输入的变化，然后由各种render()函数动态生成UI中的可视化输出（DOM模型下的javascript与HTML/CSS），再由ui.R中的对应的output()函数（起占位符的作用）生成浏览器端的可视化UI代码， sever.R中的render()函数与ui.R中的output()函数一一对应。注意最后面又嵌入了一段javascript，禁止用户自己修改输入变量“选中的节点”node\_selected，它在记录点击的节点时自动更新。

浏览器端用3D Force-Directed Graph 的API操作各种可视化效果，具体API请看项目的主页，上面还有各种例子学习，可以直接看到它的实现代码和效果。

程序已经注释的很清楚了，可以直接看程序。要用Shiny.addCustomMessageHandler('updateGraphData', function(data) {}注册一个javascript函数处理服务器发来的json格式图数据，Shiny.addCustomMessageHandler('setSourceNode', function(data) {}对源结点着色并设置相机的初始视野，以便开始可视化探索，以及Shiny.addCustomMessageHandler('clearSelectedNodes', function(data) {}注册一个函数处理清除选中结点的指令，消息类型是自定义的。其它都是javascript在浏览器DOM模型下的常规编程。与前面一样，javascript都是嵌入到ui.R的R语言程序中，不过请记住ui.R运行在服务器端而嵌入的javascript随着由ui.R动态生成的UI HTML发往浏览器端，在浏览器上运行。因为3D Force-Directed Graph操作图可视化是纯粹的javascript客户端，所以嵌入的javascript比较多，成为了ui.R的主体。

这里要讲讲装入本例3D Force-Directed Graph及其依赖的ThreeJS库的方式，直接用Shiny的tags$script(src = "3d-force-graph.js")从www子目录装入会报错：“ Cannot use import statement outside a module”，用includeScript(path = "./www/3d-force-graph.js")从本地的www子目录装入是可以的，不过会和ui.R生成的html页面，包括里面嵌入的javascript合并在一起，非常庞大，用Chrome的devtools调试javascript脚本时就比较不方便了，3d-force-graph.js、three.js、CSS2DRenderer.js、three-spritetext.js这4个JS文件加起来有67000多行源码。所以本例中还是用URI的方式从网上装入：tags$script(src="//unpkg.com/3d-force-graph")。如果Shiny Server部署在内网，就只能从本地www子目录include了。

另外，服务器查询图要执行比较复杂的路径算法，需要点时间，响应函数更新浏览器端应用状态的消息会被阻塞至所有操作完成，所以在浏览器端监测Shiny的shiny:inputchanged事件，先在浏览器端更新APP的状态，以免用户不知道APP是否在正确的运行，这通过注册一个shiny:inputchanged事件处理函数来完成，Shiny 在浏览器端会触发的事件列表，可以参考[这篇文章](https://shiny.posit.co/r/articles/build/js-events/)：

$(document).on('shiny:inputchanged', function(event) {…} 。

为了在浏览器上用javascript对图进行操作，为3D Force-Directed Graph的javascript图数据结构补充了邻接表与链接表的结构，在高亮显示选中结点的子网时就用它们来查找要高亮显示的边，也可以用它们来执行javascript端的其它图算法，比如遍历算法等。

参考3D Force-Directed Graph主页上的例子，对结点与边的着色不是通过设置它们的颜色属性进行，而是通过设置selectedNodes与highlightLinks这两个集合，然后由nodeColor()与linkColor()重新渲染，在渲染中根据是否在集合中分别赋予不同的颜色。3D效果是渲染图，每次改变都要重新渲染。也可以在函数中指定颜色属性的名称来指定结点与边的颜色，比如nodeColor(‘color’)与linkColor(‘color’)，结点与边都可在服务器端由igraph预先设置好颜色属性’color’的值，值改变后用这两个函数重新渲染即可，具体可参阅[这个链接](https://github.com/vasturiano/3d-force-graph/issues/520)及[项目主页的文档](https://github.com/vasturiano/3d-force-graph)。

边上质点流动的动画效果由linkDirectionalParticles()与emitParticle（）直接实现，运动质点的数量、速度、大小、颜色等都可以根据图中边的属性来确定，这样它就可以反映图上承载对象在时间维度上的流动状况，可视化探索就扩展为4D。就一般需求而言，结点的大小、形状、颜色与标签，边的宽度、方向、颜色与标签解决了，可视化就比较满意了，4D动画是锦上添花。

展开/折叠多重边通过调用mygraph.linkCurvature(0.25).linkVisibility(false).linkVisibility(true)完成，设置曲率为0是折叠，不为0是展开，通过设置边的可见性强制重新渲染视图。

# 浏览器端UI函数，画机场航线网络图。

fluidPage(

# Javascript 处理父窗口传入的参数等。

tags$head(

tags$script(HTML("

// 记录父窗口，初始为空。

parent = null;

// 处理接收到的消息。

window.addEventListener('message', function(e) {

//alert(e.data);

try{

//记录父窗口以备回发信息

parent = e.source;

//向服务器发送input变量更新消息

//Shiny.setInputValue('airport', e.data, {priority:'event'});

var appstatus = document.getElementById('appstatus');

// 更新APP状态，因为耗时较长

appstatus.value = '正在查询网络数据...';

Shiny.setInputValue('airport', e.data);

} catch (error){

alert(error);

}

},false);

// 点击时传出参数

$(document).on('click', '.btn-success', function (evt) {

evt.preventDefault();

var selected = document.getElementById('nodes\_selected');

if (selected ==null){

alert('No node is selected!');

}else{

if (parent == null){

alert(selected.value);

} else {

try{

parent.postMessage(selected.value, '\*');

} catch(error){

alert(error);

}

}

}

});

// 触发了shiny:inputchanged事件，从浏览器端更新状态提示，因为服务器端要执行完了才返回。

// 在触发服务器reactive()与observe之前先更新Browser端状态，因为耗时较长。

// 清除选择结点更新状态在Shiny.addCustomMessageHandler()执行完了再设置，不在这里更新。

$(document).on('shiny:inputchanged', function(event) {

var appstatus = document.getElementById('appstatus');

if (event.name === 'airport') {

appstatus.value = '正在查询网络数据...';

}

if (event.name === 'hops') {

appstatus.value = '正在查询网络数据...';

}

if (event.name === 'amplitude') {

appstatus.value = '正在查询网络数据...';

}

if (event.name === 'threshold') {

appstatus.value = '正在查询网络数据...';

}

if (event.name === 'sendout') {

appstatus.value = '选中结点已传出';

}

if (event.name === 'ifMultiEdge') {

// 不向服务器端传送信息，边曲线是纯粹的可视化效果

event.preventDefault();

try{

var ifMultiEdge = document.getElementById('ifMultiEdge');

// .linkVisibility(false).linkVisibility(true) 确保刷新视图

if (ifMultiEdge.checked){

mygraph.linkCurvature(0.25)

.linkVisibility(false)

.linkVisibility(true);

appstatus.value = '已展开重叠边的直线';

}else{

mygraph.linkCurvature(0)

.linkVisibility(false)

.linkVisibility(true);

appstatus.value = '已叠加多重边为直线';

}

}catch(error){

alert(error);

}

}

});

"))

),

sidebarLayout(

# Sidebar with a selectioninput and a numericInput

sidebarPanel(

# Application title

tags$h3("机场航线环路"),

selectInput(

'airport',

'国内机场',

airports,

selected = 'Zhuhai Airport'

),

sliderInput("hops",

"最多跳数：",

min = 2, max = 5, value = 3, step = 1),

sliderInput("amplitude",

"距离振幅(%)：",

min = 0.1, max = 0.4, value = 0.2, step = 0.05),

sliderInput("threshold",

"距离阀值(KM)：",

min = 600, max = 2000, value = 800, step = 100),

checkboxInput("ifRecord","记录点击的结点", value = FALSE),

# 手机没有键盘，这个开关是为了方便手机使用

checkboxInput("ifMultiSelect","连续选取结点", value = TRUE),

checkboxInput("ifMultiEdge","用曲线展开多重边", value = FALSE),

actionButton("sendout", "传出选中结点", class = "btn-success"),

tags$h6(" "),

actionButton("clearNodes", "清除选中结点", class = "btn-cleear"),

tags$h6(" "),

textInput("appstatus","APP状态", value ="正在查询网络数据..."),

textAreaInput("nodes\_selected","选中的结点", rows =6, resize="vertical", value =""),

# 插入javascript，禁止自己修改 nodes\_selected textAreaInput, appstatus textInput

tags$script(HTML("

var nodes\_selected = document.getElementById('nodes\_selected');

nodes\_selected.disabled = true;

var appstatus = document.getElementById('appstatus');

appstatus.disabled = true;

")),

# 1/6

width = 2

),

# Show the network

mainPanel(

# 画航线网络图

# 定义图显示的区域, 在该区域中加载图

tags$div(id="3d-graph"),

# Style sheet of node label

tags$head(

tags$style("

.node-label {

font-size: 12px;

padding: 1px 4px;

border-radius: 4px;

color: LightGoldenRodYellow;

background-color: none; // rgba(0,0,0,0.5)

user-select: none;

}

")),

# 定义后面消息处理函数要使用的全局变量

HTML("

<script>

//定义图的全局变量

//保存对Graph对象的全局引用，取消结点选择时要处理图的结点与边

let mygraph = null;

//选中的结点集

let selectedNodes = new Set();

//选中的结点集之间的边

let highlightLinks = new Set();

//选中结点与未选中结点的颜色

// 参见下面链接中的颜色表。

// https://www.w3schools.com/tags/ref\_colornames.asp

// 最好写成#十六进制，有些按名称引用的可能不认识。边的颜色会被稀释淡化，所以不少颜色之间可能看不出区别。

let colorNodeSelected = 'GoldenRod';

let colorNode = 'LightSeaGreen';

// 选中边与未选中边的颜色

let colorEdgeSelected = 'gold';

let colorEdge = '#20B2AA'; // #20B2AA, LightSeaGreen， rgb(32, 178, 170)。

// 更新加载状态的引用

let APPStatus = document.getElementById('appstatus');

// 初始化为一个空图

let gData = {

nodes: [],

links: []

};

</script>

"),

# Scripts for Vasturiano 3D Force-Directed Graph

# load 3d-force-graph.js, three.js

# tags$script(src = "3d-force-graph.js"),

# Will cause an error: Cannot use import statement outside a module

# It's O.K. this way, but need to connect to unpkg.com

tags$script(src="//unpkg.com/three"),

tags$script(src="//unpkg.com/3d-force-graph"),

# 定义后面消息处理函数updateGraphData()要导入的javascript库，ECMAScript modules

HTML('

<script type="importmap">{ "imports": {

"three": "//unpkg.com/three/build/three.module.js",

"three/addons/": "//unpkg.com/three/examples/jsm/",

"three-spritetext": "//unpkg.com/three-spritetext/dist/three-spritetext.mjs"

}}</script>

'),

# 注册一个javascript消息处理函数，接收服务器发来的json图数据，更新图

# 用HTML(), 否则javascript lambda表达式的=>箭头会被替换成&转义符，导致javascript执行出错。

# 消息类型是updateGraphData，格式是json。

HTML("

<script type='module'>

import { CSS2DRenderer, CSS2DObject } from '//unpkg.com/three/examples/jsm/renderers/CSS2DRenderer.js';

import SpriteText from 'three-spritetext';

//接收服务器传来的json图数据，转换为图对象

Shiny.addCustomMessageHandler('updateGraphData', function(data) {

//装入图数据json对象

try {

gData = JSON.parse(data);

} catch(error){

alert(error);

return;

}

// 为Vasturiano 3D Force-Directed Graph的javascript表示补充邻接表结构，以便javascript中执行图算法。

// 初始化邻接表，为结点增加neighbors列表与links列表

gData.nodes.forEach(node => {

node.neighbors=[];

node.links=[];

});

// 邻接是双向的，这里初始化为无向图邻接，也可以按需要初始化为有向图邻接。

gData.links.forEach(link => {

// 按属性值查找出源与目标结点

var sourceNode = gData.nodes.find(obj => {return obj.id === link.source});

var targetNode = gData.nodes.find(obj => {return obj.id === link.target});

sourceNode.neighbors.push(link.target);

targetNode.neighbors.push(link.source);

sourceNode.links.push(link);

targetNode.links.push(link);

});

APPStatus.value = '正在加载网络数据...';

//加载图，可视化显示

const Graph = ForceGraph3D({

// 用于为结点加上标签

// extraRenderers: [new THREE.CSS2DRenderer()]

extraRenderers: [new CSS2DRenderer()]

})

// 获得显示的区域，上面tags$div(id='3d-graph')定义的对象

(document.getElementById('3d-graph'))

// 加载数据

.graphData(gData )

// 加载测试数据

//.jsonUrl('https://raw.githubusercontent.com/vasturiano/3d-force-graph/master/example/datasets/miserables.json')

// 定义结点标签，鼠标移动到上面时会显示

.nodeLabel('id')

// 设定结点的大小

.nodeVal(node => node.size)

// 按组显示颜色

//.nodeAutoColorBy('group')

// 选中的结点以不同的颜色标记

.nodeColor(node => selectedNodes.has(node) ? colorNodeSelected : colorNode)

// 也可以通过指定结点的颜色属性名称来指定结点的颜色，这样可以先在igraph中处理

//.nodeColor('color')

// 为结点加上标签，使用结点扩展

.nodeThreeObject(node => {

const nodeEl = document.createElement('div');

nodeEl.textContent = node.id;

//nodeEl.style.color = node.color;

nodeEl.className = 'node-label';

//return new THREE.CSS2DObject(nodeEl);

return new CSS2DObject(nodeEl);

})

.nodeThreeObjectExtend(true)

// 为边加上标签，使用边扩展

.linkThreeObjectExtend(true)

.linkThreeObject(link => {

// extend link with text sprite

const sprite = new SpriteText(`${link.distance}`);

sprite.color = 'lightgrey';

sprite.textHeight = 1.5;

return sprite;

})

.linkPositionUpdate((sprite, { start, end }) => {

const middlePos = Object.assign(...['x', 'y', 'z'].map(c => ({

[c]: start[c] + (end[c] - start[c]) / 2 // calc middle point

})));

// Position sprite

Object.assign(sprite.position, middlePos);

})

// 为边加上箭头

.linkDirectionalArrowLength(3.5)

.linkDirectionalArrowRelPos(1)

// 设定边的宽度

.linkWidth(link => link.width)

// 设置边的曲率，不设置就是直线，设置曲率对于多重边的效果比较好一点，不设置的话多重边会合并成双向直线。

//.linkCurvature(0.25)

// 处理结点的点击事件，选中的结点以不同的颜色标记，并高亮显示选中结点之间的边。

.onNodeClick((node, event) => {

// 查看'记录点击的结点'开关及'连续选取结点开关'

var ifRecord = document.getElementById('ifRecord');

var ifMultiSelect = document.getElementById('ifMultiSelect');

//alert(ifRecord.checked);

if(ifRecord.checked){

// multi-selection

if (event.ctrlKey || event.shiftKey || event.altKey || ifMultiSelect.checked) {

//selectedNodes.has(node) ? selectedNodes.delete(node) : selectedNodes.add(node);

if (selectedNodes.has(node)){

// 先删除结点再处理边

selectedNodes.delete(node);

if(node.links.length>0){

node.links.forEach(link => {

if (selectedNodes.has(link.source) || selectedNodes.has(link.target)){

//alert('delete highlight edge!');

highlightLinks.delete(link);

}

});

}

}

else{

// 在添加结点前先处理边

if(node.links.length>0){

node.links.forEach(link => {

if (selectedNodes.has(link.source) || selectedNodes.has(link.target)){

//alert('add highlight edge!');

highlightLinks.add(link);

}

});

}

// 添加结点

selectedNodes.add(node);

}

} else { // single-selection

const untoggle = selectedNodes.has(node) && selectedNodes.size === 1;

// 清空选中的结点集及高亮显示的边集

highlightLinks.clear();

selectedNodes.clear();

// 如果原来没有结点就添加该结点

if ( !untoggle){

selectedNodes.add(node);

}

}

// update color of selected nodes and Edges

Graph.nodeColor(node => selectedNodes.has(node) ? colorNodeSelected : colorNode)

.linkColor(link => highlightLinks.has(link)? colorEdgeSelected : colorEdge);

// 通知服务器选中的结点有变化

var nodeList = '';

selectedNodes.forEach(node => {

if (nodeList.length>0)

nodeList = nodeList.concat('!',node.label);

else

nodeList = node.label;

});

Shiny.setInputValue('nodes\_selected', nodeList);

}

})

// 设置边的颜色，开始时没有高亮的边。

.linkColor((link) => {return colorEdge;})

// 也可以通过指定边的颜色属性名称来指定颜色，这样可以先在igraph中处理

//.linkColor('color')

// 边动画演示，加亮显示的边有1个流动的点，从源到目标结点，其它边没有

.linkDirectionalParticles(link => highlightLinks.has(link) ? 2 : 0)

// 通过速度来表示流量，各条边的流量不同。

.linkDirectionalParticleSpeed(link => link.paths \* 0.01)

// 边动画演示中流动小点的尺寸

.linkDirectionalParticleWidth(3)

// 边动画演示中流动小点的颜色，不定义就是与边的颜色一样

.linkDirectionalParticleColor(() => 'orange')

// 点击边时显示动画，从源到目标结点。

.onLinkClick((link, event) =>{Graph.emitParticle(link)});

;

// Spread nodes a little wider

Graph.d3Force('charge').strength(-120);

// 把Graph对象传出事件处理函数之外，以便全局引用，比如清除选择的结点与边

mygraph = Graph;

// 更新加载状态

APPStatus.value = '网络数据已加载';

});

//alert('Registered!');

</script>

"),

tags$script(HTML("

//接收服务器传来清除选中结点的指令。

Shiny.addCustomMessageHandler('clearSelectedNodes', function(data) {

try{

// 清空 selectedNodes, highlightLinks，并设置选中源结点

selectedNodes.clear();

highlightLinks.clear();

var sourceNode = gData.nodes.find(obj => {return obj.id === data});

selectedNodes.add(sourceNode);

// 应用前面保存的全局图对象，重新设置选中与没有选中结点的颜色

mygraph.nodeColor(node => selectedNodes.has(node) ? colorNodeSelected : colorNode)

.linkColor((link) => {return colorEdge;});

// 更新结点选择状态

APPStatus.value = '已清除选择的结点';

}catch(error){

alert(error);

}

});

//alert('Registered!');

")),

tags$script(HTML("

//接收服务器传来设置源结点初始视野的指令。

Shiny.addCustomMessageHandler('setSourceNode', function(data) {

try{

selectedNodes.clear();

//设置起点机场为默认选中

var sourceNode = gData.nodes.find(obj => {return obj.id === data});

selectedNodes.add(sourceNode);

// 应用前面保存的全局图对象，重新设置选中与没有选中结点的颜色

mygraph.nodeColor(node => selectedNodes.has(node) ? colorNodeSelected : colorNode);

// 测试通过属性指定颜色

//mygraph.nodeColor('color')

var hops = document.getElementById('hops');

if (hops.value < '5'){

// 如果是5跳，就不设置变焦

// 设置初始焦点

// 参阅： https://github.com/vasturiano/3d-force-graph/issues/523

mygraph.onEngineTick(() => {

//focusNode();

// 设置焦距

var distance = 320;

var distRatio = 1 + distance/Math.hypot(sourceNode.x, sourceNode.y, sourceNode.z);

mygraph.cameraPosition(

// new position

{ x: sourceNode.x \* distRatio, y: sourceNode.y \* distRatio, z: sourceNode.z \* distRatio },

sourceNode, // lookAt ({ x, y, z })

3000 // ms transition duration

);

// From now on, we don't want to invoke this function anymore.

mygraph.onEngineTick(() => {});

});

}

// 等Force Engin执行完成后，图稳定下来，自动调整焦距到看到全图。

// https://github.com/vasturiano/3d-force-graph/issues/340

mygraph.onEngineStop(() => {

mygraph.zoomToFit(0,1,node =>node.id===sourceNode.id);

// From now on, we don't want to invoke this function anymore.

mygraph.onEngineStop(() => {});

});

// 更新结点选择状态

APPStatus.value = '已设自动调整焦距';

}catch(error){

alert(error);

}

});

//alert('Registered!');

")),

# 5/6

width = 10

)

)

)

4、server.R

server.R中通过observ()函数监测airport、hops、amplitude、threshold四个输入变量的变化，调用Python函数查询给定机场指定跳数内的稳定强环路，这里是Shiny的Reactive机制，然后在该函数的最后，把查询返回的图数据，转换为 3D Force-Directed Graph API要求的json数据格式，调用session$sendCustomMessage("updateGraphData", jsonGraph)发送给浏览器，这里updateGraphData是自定义的消息类型，jsonGraph就是json格式的图数据，调用session$sendCustomMessage("setSourceNode", airport())通知浏览器设置相机的初始视野。后面清除选中结点也是通过这个机制，发送一个消息给浏览器执行：session$sendCustomMessage("clearSelectedNodes", airport()) 。

# 服务端函数，根据浏览器选定的机场选出其给定步数内航线可达的机场。

function(input, output, session) {

# 定义reactive变量，监视输入的变化

# 选定的机场

airport<- reactive({

input$airport

})

# 指定的跳数

hops<- reactive({

as.integer(input$hops)

})

# 指定的振幅内

amplitude<- reactive({

as.numeric(input$amplitude)

})

# 指定的阀值内

threshold<- reactive({

as.integer(input$threshold)

})

# 选中的结点

nodes\_selected<-reactive({

input$nodes\_selected

})

# 监视输入变量airport与hops的变化事件。

observe({

# 这一句执行了,证明触发了

print(paste("Observe triggered: ",paste(airport(),hops())))

try({

t1<-proc.time()

# 调Python函数得到指定机场指定跳数以内的航线环路图

graph<- getRings(airport(),hops(), amplitude(), threshold())

t2<-proc.time()

timeUsed <- t2-t1

# 合并多重边，存在多条路径都经过同一条边的情况，统计多重边的数量作为合并后边的权重。

graph2<-sqldf("select source,target,distance, count(\*) as paths

from graph group by source,target,distance

order by source,target")

# 建立 igraph图

g <- graph.data.frame(graph2, directed = TRUE)

# 节点标签

V(g)$label<-V(g)$name

# 节点的航班数，最短路径子网统计出度和入度，因为末端的结点只有入度。这是为了演示设置结点的大小。

# 对于完全子网，航线比较多，只统计入度

V(g)$size <- degree(g, mode="out")

# 最短路径子网边的宽度

E(g)$width<-E(g)$distance/200

# 边标签

E(g)$label<-as.character(E(g)$distance)

# 测试通过颜色属性设置结点的颜色，这样可以通过igraph先设定结点与边的颜色，再传给3D Forced Directed Graph

# 这4句只是测试用途，可以不用管

V(g)$color<-"#00BFFF"

V(g)$color[1]<-"red"

E(g)$color<-"green"

E(g)$color[1]<-"red"

# 转换为 3D Force-Directed Graph 要求的json网络格式。

# 结点列表

verts<- as\_data\_frame(g, what="vertices")

# 必须有id列

names(verts)<-c("id", "label", "size","color")

# 边列表

edges<- as\_data\_frame(g, what="edges")

names(edges)<-c("source", "target","distance","paths","width","label","color")

jsonVerts<- toJSON(verts)

jsonEdges<- toJSON(edges)

# 合并结点与边的json数据集

jsonGraph<- paste('{

"nodes": ',jsonVerts,',

"links": ',jsonEdges,'}')

# 发送给浏览器，更新网络图

session$sendCustomMessage("updateGraphData", jsonGraph)

# 更新浏览器端selectInput的状态

# 注意这里用updateTextInput更新文本到浏览器端，如果用updateSelectInput()会报错，可能要求的是下标。

updateTextInput(session, "airport", value = airport())

# 设置初始视野，通知浏览器端聚焦源结点并着色，以便分析。

session$sendCustomMessage("setSourceNode", airport())

# 这个过程需要点时间，告知客户端更新APP状态。

updateTextInput(session, "appstatus", value = paste("网络数据已加载: ",round(timeUsed[3],1),"秒",sep=""))

})

})

# 浏览器端结点的javascript中.onNodeClick((node, event)事件处理，

# 发送消息函数Shiny.setInputValue('nodes\_selected', nodeList)会触发这个事件处理函数。

observeEvent(nodes\_selected(),{

# 这一句执行了,证明触发了

print(paste("Nodes selected event triggered: ",nodes\_selected()))

# 在session中记录选中的结点，记录APP的状态

session$userData$nodes\_selected<- nodes\_selected()

# 更新浏览器端显示的输入变量nodes\_selected

updateTextAreaInput(session, "nodes\_selected", value = nodes\_selected())

})

# 清除选中节点，浏览器端点击'清除选中结点'按钮会触发这个事件处理函数。

observeEvent(input$clearNodes,{

print("Clear nodes selected event triggered.")

# 更新session中记录的用户数据

session$userData$nodes\_selected <- ""

# 更新浏览器端显示的输入变量nodes\_selected

updateTextAreaInput(session, "nodes\_selected", value = "")

# 发送一个清理指令给浏览器端注册的javascript函数，重置图中选中结点的颜色并更新APP的状态。

session$sendCustomMessage("clearSelectedNodes", airport())

})

}

**第四节 开发Neo4j自定义函数**

在各行各业实际落地应用的网络分析中，作为载荷的目标子网是千变万化的，从图数据库中选择目标子网就要使用一些过滤结点或边或路径的方法或算法，Neo4j中有很多这样的机制，不过不一定能满足各种应用场景的需求，所以它提供了用户 [自定义函数与过程](https://neo4j.com/docs/java-reference/current/extending-neo4j/customized-code/)的机制。比如在机场航线稳定强环路查找的例子中，对于查询经过源机场的环路，用了 [APOC插件](https://neo4j.com/labs/apoc/5/overview/)基于集合的三个函数apoc.coll.avg()、apoc.coll.min()、apoc.coll.max()来过滤出稳定强环路。假如要用中位数而不是平均值（更符合实际），就没有现成的函数可用，需要自己写了。

1、 [Neo4j的用户自定义函数与过程](https://neo4j.com/docs/java-reference/current/extending-neo4j/customized-code/)

用户自定义过程与函数的区别是，自定义过程返回一个Stream流序列，在Cypher中引用时要用YIELD逐组提取。比如 [Neo4j自定义过程样板项目](https://github.com/neo4j-examples/neo4j-procedure-template)中提取结点连接类型的用户自定义过程 [GetRelationshipTypes.java](https://github.com/neo4j-examples/neo4j-procedure-template/blob/5.x/src/main/java/example/GetRelationshipTypes.java)， [它的Junit测试](https://github.com/neo4j-examples/neo4j-procedure-template/blob/5.x/src/test/java/example/GetRelationshipTypesTests.java)中是这样引用的：

/\*\*

\* We should be getting the correct values when there is only one type in each direction

\*/

@Test

public void shouldReturnTheTypesWhenThereIsOneEachWay() {

final String expectedIncoming = "INCOMING";

final String expectedOutgoing = "OUTGOING";

// In a try-block, to make sure we close the session after the test

try(Session session = driver.session()) {

//Create our data in the database.

session.run(String.format("CREATE (:Person)-[:%s]->(:Movie {id:1})-[:%s]->(:Person)", expectedIncoming, expectedOutgoing));

//Execute our procedure against it.

Record record = session.run(**"MATCH (u:Movie {id:1}) CALL example.getRelationshipTypes(u) YIELD outgoing, incoming RETURN outgoing, incoming**").single();

//Get the incoming / outgoing relationships from the result

assertThat(record.get("incoming").asList(Value::asString)).containsOnly(expectedIncoming);

assertThat(record.get("outgoing").asList(Value::asString)).containsOnly(expectedOutgoing);

}

}

看看 [它的源码](https://github.com/neo4j-examples/neo4j-procedure-template/blob/5.x/src/main/java/example/GetRelationshipTypes.java)，通过Java Annotation @Procedure定义为用户自定义过程及规定它的调用名字example.getRelationshipTypes，然后返回Stream类型。

package example;

import java.util.ArrayList;

import java.util.List;

import java.util.Map;

import java.util.stream.Stream;

import org.neo4j.graphdb.Direction;

import org.neo4j.graphdb.Node;

import org.neo4j.graphdb.Relationship;

import org.neo4j.logging.Log;

import org.neo4j.procedure.Context;

import org.neo4j.procedure.Description;

import org.neo4j.procedure.Name;

import org.neo4j.procedure.Procedure;

/\*\*

\* This is an example showing how you could expose Neo4j's full text indexes as

\* two procedures - one for updating indexes, and one for querying by label and

\* the lucene query language.

\*/

public class GetRelationshipTypes {

// This gives us a log instance that outputs messages to the

// standard log, normally found under `data/log/console.log`

@Context

public Log log;

/\*\*

\* This procedure takes a Node and gets the relationships going in and out of it

\*

\* @param node The node to get the relationships for

\* @return A RelationshipTypes instance with the relations (incoming and outgoing) for a given node.

\*/

**@Procedure(value = "example.getRelationshipTypes")**

**@Description("Get the different relationships going in and out of a node.")**

public **Stream<RelationshipTypes>** getRelationshipTypes(@Name("node") Node node) {

List<String> outgoing = new ArrayList<>();

node.getRelationships(Direction.OUTGOING).iterator()

.forEachRemaining(rel -> AddDistinct(outgoing, rel));

List<String> incoming = new ArrayList<>();

node.getRelationships(Direction.INCOMING).iterator()

.forEachRemaining(rel -> AddDistinct(incoming, rel));

**return Stream.of(new RelationshipTypes(incoming, outgoing));**

}

/\*\*

\* Adds the distinct type of a relationship to the given List<String>

\*

\* @param list the list to add the distinct relationship type to

\* @param relationship the relationship to get the name() from

\*/

private void AddDistinct(List<String> list, Relationship relationship){

AddDistinct(list, relationship.getType().name());

}

/\*\*

\* Adds an item to a List only if the item is not already in the List

\*

\* @param list the list to add the distinct item to

\* @param item the item to add to the list

\*/

private <T> void AddDistinct(List<T> list, T item){

if(!list.contains(item))

list.add(item);

}

/\*\*

\* This is the output record for our search procedure. All procedures

\* that return results return them as a Stream of Records, where the

\* records are defined like this one - customized to fit what the procedure

\* is returning.

\* <p>

\* These classes can only have public non-final fields, and the fields must

\* be one of the following types:

\*

\* <ul>

\* <li>{@link String}</li>

\* <li>{@link Long} or {@code long}</li>

\* <li>{@link Double} or {@code double}</li>

\* <li>{@link Number}</li>

\* <li>{@link Boolean} or {@code boolean}</li>

\* <li>{@link Node}</li>

\* <li>{@link org.neo4j.graphdb.Relationship}</li>

\* <li>{@link org.neo4j.graphdb.Path}</li>

\* <li>{@link Map} with key {@link String} and value {@link Object}</li>

\* <li>{@link List} of elements of any valid field type, including {@link List}</li>

\* <li>{@link Object}, meaning any of the valid field types</li>

\* </ul>

\*/

public static class RelationshipTypes {

// These records contain two lists of distinct relationship types going in and out of a Node.

public List<String> outgoing;

public List<String> incoming;

public RelationshipTypes(List<String> incoming, List<String> outgoing) {

this.outgoing = outgoing;

this.incoming = incoming;

}

}

}

用户自定义函数则返回一个单一的值，又分为标量函数与聚合函数两种，它们的Java Annotation与API都不同。

这个是APOC中 [apoc.agg.median()函数的源码](https://github.com/neo4j/apoc/blob/dev/core/src/main/java/apoc/agg/Median.java)，它是一个聚合函数，通过@UserAggregationFunction定义，通过@Description定义函数调用的方式，要实现由Java Annotation指定的aggregate()与result()接口：

package apoc.agg;

import org.neo4j.procedure.\*;

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.List;

/\*\*

\* @author mh

\* @since 18.12.17

\*/

public class Median {

**@UserAggregationFunction("apoc.agg.median")**

**@Description("apoc.agg.median(number) - returns median for non-null numeric values")**

public MedianFunction median() {

return new MedianFunction();

}

public static class MedianFunction {

private List<Double> values = new ArrayList<>();

**@UserAggregationUpdate**

**public void aggregate(@Name("value") Object value) {**

if (value instanceof Number) {

values.add(((Number)value).doubleValue());

}

}

**@UserAggregationResult**

**public Object result() {**

if (values.isEmpty()) return null;

Collections.sort(values);

int size = values.size();

if (size % 2 == 1) {

**return values.get(size /2);**

} else {

**return (values.get(size /2-1) + values.get(size /2)) / 2D;**

}

}

}

}

这个函数为什么在稳定强环路查询中不能使用呢？因为它是对查询中的所有结点或边求中位数，而上面的查询需要对特定路径中边的权重（集合）求中位数，即按path分组，所以要自己写一个。

这个是APOC中 [apoc.coll.avg()函数的源码](https://github.com/neo4j/apoc/blob/dev/core/src/main/java/apoc/coll/Coll.java)，通过@UserFunction定义，也通过@Description定义函数调用的方式，是一个标量函数，它不需要实现特定的接口。

**@UserFunction**

**@Description("apoc.coll.avg([0.5,1,2.3])")**

public Double avg(@Name("numbers") List<Number> list) {

if (list == null || list.isEmpty()) return null;

double avg = 0;

for (Number number : list) {

avg += number.doubleValue();

}

**return (avg/(double)list.size());**

}

2、建立用户自定义函数Maven项目

可以参考 [Neo4j的文档来自行建立Maven项目](https://neo4j.com/docs/java-reference/current/extending-neo4j/project-setup/)的pom.xml项目描述文件，我直接拷贝 [Neo4j的样板项目](https://github.com/neo4j-examples/neo4j-procedure-template)稍作修改，因为它已经配好了编译与Junit测试的依赖，修改依赖的Neo4j服务器版本号等即可，这里改为5.10.0，因为目前中文社区版的版本是5.10.0。另外还要给个合适的groupId与artifactId及版本号，生成的jar包以artifactId+version命名。

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>jean.neo4j</groupId>

<artifactId>my-neo4j-functions5x</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>My Neo4j Functions</name>

<description>A function for median value</description>

<properties>

<java.version>17</java.version>

<maven.compiler.release>${java.version}</maven.compiler.release>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<neo4j.version>5.10.0</neo4j.version>

<neo4j-java-driver.version>5.10.0</neo4j-java-driver.version>

<junit-jupiter.version>5.10.0</junit-jupiter.version>

<maven-shade-plugin.version>3.5.1</maven-shade-plugin.version>

<maven-compiler-plugin.version>3.11.0</maven-compiler-plugin.version>

<assertj.version>3.24.2</assertj.version>

<maven-surefire-plugin.version>3.1.2</maven-surefire-plugin.version>

<maven.version>3.9.4</maven.version>

<maven-enforcer-plugin.version>3.4.1</maven-enforcer-plugin.version>

</properties>

<dependencies>

<dependency>

<!-- This gives us the Procedure API our runtime code uses.

We have a `provided` scope on it, because when this is

deployed in a Neo4j Instance, the API will be provided

by Neo4j. If you add non-Neo4j dependencies to this

project, their scope should normally be `compile` -->

<groupId>org.neo4j</groupId>

<artifactId>neo4j</artifactId>

<version>${neo4j.version}</version>

<scope>provided</scope>

</dependency>

<!-- Test Dependencies -->

<dependency>

<!-- This is used for a utility that lets us start Neo4j with

a specific Procedure, which is nice for writing tests. -->

<groupId>org.neo4j.test</groupId>

<artifactId>neo4j-harness</artifactId>

<version>${neo4j.version}</version>

<scope>test</scope>

<exclusions>

<exclusion>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-nop</artifactId>

</exclusion>

</exclusions>

</dependency>

<dependency>

<!-- Used to send cypher statements to our procedure. -->

<groupId>org.neo4j.driver</groupId>

<artifactId>neo4j-java-driver</artifactId>

<version>${neo4j-java-driver.version}</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-engine</artifactId>

<version>${junit-jupiter.version}</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.assertj</groupId>

<artifactId>assertj-core</artifactId>

<version>${assertj.version}</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<artifactId>maven-enforcer-plugin</artifactId>

<version>${maven-enforcer-plugin.version}</version>

<executions>

<execution>

<id>enforce</id>

<goals>

<goal>enforce</goal>

</goals>

<phase>validate</phase>

<configuration>

<rules>

<requireJavaVersion>

<version>${java.version}</version>

</requireJavaVersion>

<requireMavenVersion>

<version>${maven.version}</version>

</requireMavenVersion>

</rules>

</configuration>

</execution>

</executions>

</plugin>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>${maven-compiler-plugin.version}</version>

</plugin>

<plugin>

<artifactId>maven-surefire-plugin</artifactId>

<version>${maven-surefire-plugin.version}</version>

</plugin>

<plugin>

<!-- This generates a jar-file with our procedure code,

plus any dependencies marked as `compile` scope.

This should then be deployed in the `plugins` directory

of each Neo4j instance in your deployment.

After a restart, the procedure is available for calling. -->

<artifactId>maven-shade-plugin</artifactId>

<version>${maven-shade-plugin.version}</version>

<configuration>

<createDependencyReducedPom>false</createDependencyReducedPom>

</configuration>

<executions>

<execution>

<phase>package</phase>

<goals>

<goal>shade</goal>

</goals>

</execution>

</executions>

</plugin>

</plugins>

</build>

</project>

3、开发apoc.coll.median()函数

参考上面apoc.agg.median()的源码，实现很简单。

package apoc.coll;

import org.neo4j.procedure.Description;

import org.neo4j.procedure.Name;

import org.neo4j.procedure.UserFunction;

import java.util.Collections;

import java.util.Comparator;

import java.util.List;

public class MyFunctions {

@UserFunction

@Description("apoc.coll.median([0.5,1,2.3])")

public Double median(@Name("numbers") List<Number> values) {

if (values == null || values.isEmpty()) return null;

Collections.sort(values,new Comparator<Number>() {

@Override

public int compare(Number o1, Number o2) {

Double d1 = (o1 == null) ? Double.POSITIVE\_INFINITY : o1.doubleValue();

Double d2 = (o2 == null) ? Double.POSITIVE\_INFINITY : o2.doubleValue();

return d1.compareTo(d2);

}

});

int size = values.size();

if (size % 2 == 1) {

return values.get(size /2).doubleValue();

} else {

double first = values.get(size /2-1).doubleValue();

double second = values.get(size /2).doubleValue();

return (first + second)/2;

}

}

}

4、Junit单元测试

需要编写一个单元测试程序，测试时需要加载一个嵌入式的Neo4j Server并连接到它执行Cypher语句，Neo 4j样板项目中的 [JoinTest.java](https://github.com/neo4j-examples/neo4j-procedure-template/blob/5.x/src/test/java/example/JoinTest.java)稍为修改一下即可，然后在Project Explorer项目树上的程序名上右键->Run As->Junit Test即可。

package apoc.coll;

import org.junit.jupiter.api.AfterAll;

import org.junit.jupiter.api.BeforeAll;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.TestInstance;

import org.neo4j.driver.Driver;

import org.neo4j.driver.GraphDatabase;

import org.neo4j.driver.Session;

import org.neo4j.harness.Neo4j;

import org.neo4j.harness.Neo4jBuilders;

import apoc.coll.MyFunctions;

import static org.assertj.core.api.Assertions.assertThat;

@TestInstance(TestInstance.Lifecycle.PER\_CLASS)

public class TestMedian {

private Neo4j embeddedDatabaseServer;

@BeforeAll

void initializeNeo4j() {

this.embeddedDatabaseServer = Neo4jBuilders.newInProcessBuilder()

.withDisabledServer()

.withFunction(MyFunctions.class)

.build();

}

@AfterAll

void closeNeo4j() {

this.embeddedDatabaseServer.close();

}

@Test

void medianNumbers() {

// This is in a try-block, to make sure we close the driver after the test

try(Driver driver = GraphDatabase.driver(embeddedDatabaseServer.boltURI());

Session session = driver.session()) {

// When

//double result = session.run( "RETURN apoc.coll.median([0.5,1,2.3,2,3]) AS result").single().get("result").asDouble();

double result = session.run( "RETURN apoc.coll.median([4,2,3,1]) AS result").single().get("result").asDouble();

// Then

//assertThat( result).isEqualTo(2);

assertThat( result).isEqualTo(2.5);

}

}

}

5、打包发布

在项目根目录上右键->Run As->Maven Install，会在项目的target目录下生成打包好的jar文件。拷贝到服务器的plugins目录下，重启服务器加载，然后在Neo4j Browser中访问测试，执行上面的测试Cypher语句。

RETURN apoc.coll.median([0.5,1,2.3,2,3]) AS result;

RETURN apoc.coll.median([4,2,3,1]) AS result;

6、在Python函数中引用

替代neo4jAirports.py中原来引用的apoc.coll.avg()函数即可。

def airports\_ring(tx, source, length, amplitude, threshold):

cypher = (\

'''

MATCH (china:Airport)-[:IN\_COUNTRY]->(c:Country{code:"CN"})

WITH COLLECT(china.id) AS targets

MATCH path = (source:Airport{descr:$name})-[r:HAS\_ROUTE\*2..'''+str(int(length))+"]->(target)"+\

'''

WHERE target.descr = source.descr

AND ALL( node IN nodes(path) where node.id IN targets)

WITH path, [r in relationships(path)|r.distance] AS distances, $amplitude AS amplitude, $threshold AS threshold

//WITH path, apoc.coll.avg(distances) AS avgDist, apoc.coll.min(distances) as minDist, apoc.coll.max(distances) as maxDist

//WHERE minDist/avgDist >= (1-amplitude) AND maxDist/avgDist <= (1+amplitude) AND avgDist >= threshold

WITH path, apoc.coll.median(distances) AS medianDist, apoc.coll.min(distances) as minDist, apoc.coll.max(distances) as maxDist

WHERE minDist/medianDist >= (1-amplitude) AND maxDist/medianDist <= (1+amplitude) AND medianDist >= threshold

WITH path, relationships(path) as flights

UNWIND flights AS flight

WITH DISTINCT flight

MATCH (n:Airport)-[flight]->(m:Airport)

RETURN n.descr AS source, m.descr as target, flight.distance AS distance;

'''

)

print(cypher)

start = []; target=[]; distance=[]

for record in tx.run(cypher, name=source, amplitude=amplitude, threshold=threshold ):

start.append(record["source"])

target.append(record["target"])

distance.append(record["distance"])

# 从 list建立 pandas data frame

data = pd.DataFrame({"source":start,"target":target,"distance":distance})

return data

通过这个小例子可以看到，开发测试Neo4j用户自定义函数是比较简单方便的，这个机制可以解决很多应用场景中过滤结点、边和路径的特殊需要，有效的选出目标子网进一步分析。用户自定义过程的开发可以参考 [Neo4j自定义过程样板项目](https://github.com/neo4j-examples/neo4j-procedure-template)，不过如果要开发嵌入GDS库中配合其它GDS库算法使用的自定义过程，比如有向图最小生成树算法等，就比较复杂，这里就不展开介绍了。