# 2021 大数据管理课程指导书

## 实验的环境准备

### 软件的安装

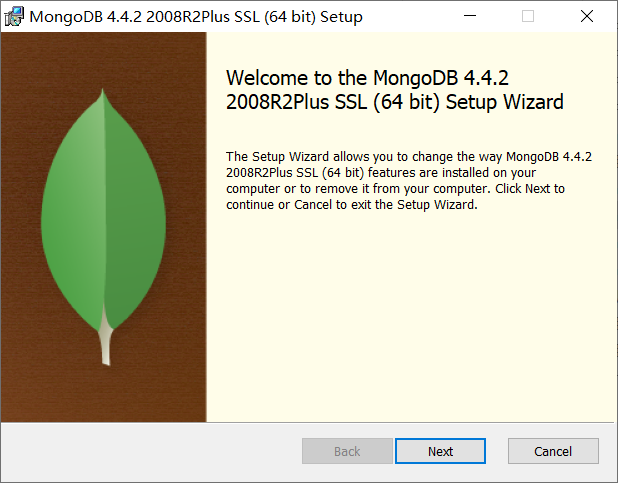
主要是介绍在windows上的安装，其它平台可以参考对应软件官网安装指南

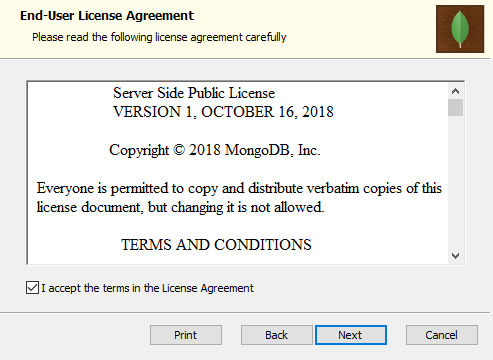
最好使用相同版本号的软件，避免因为版本而产生问题

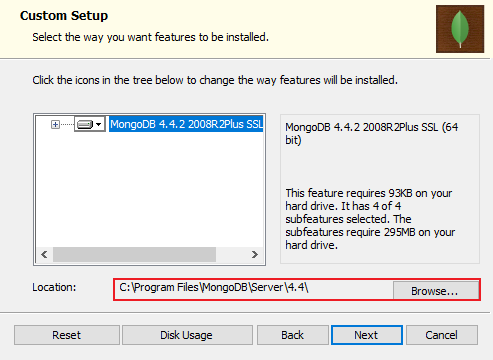
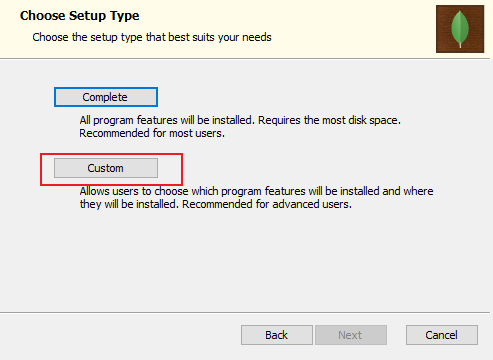
MongoDB 基本安装

**mongodb-windows-x86\_64-4.4.2-signed.msi**

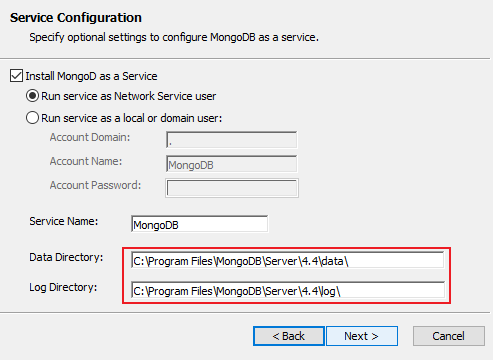
双击安装：



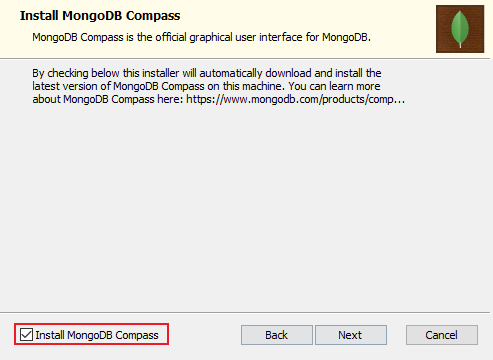




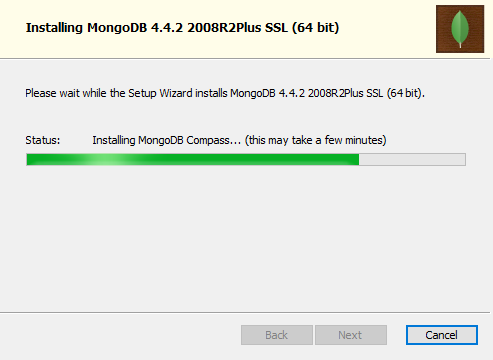
选择一个合适的位置安装，



这个地方，通常会随着上面的选择而自动的改变，可根据自己的情况改变



这个地方安装一个GUI工具，方便使用，也可以不安装，去掉勾即可



等待安装的完成。

安装完成之后，默认的情况下是无账号密码即可进入使用，在安装路径的bin文件夹下

输入./mongo 即可进入交互终端

### 数据的导入

为了方便的导入数据，需要安装一个工具mongodb-database-tools-windows-x86\_64-100.3.1.msi。安装过程比较简单，这里不再演示，注意要记得安装的路径，安装完成后，到安装的路径下应该可以看到如下的文件

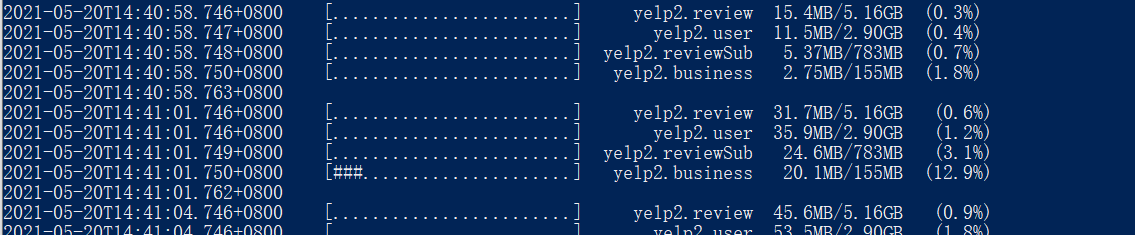


使用命令导入数据，首先解压给出data\_for\_mongo压缩包，记下解压的路径，然后

./mongorestore -h localhost -d yelp --dir=填写上面解压的路径（例如G:\mongo\_data\yelp

）

运行的类似下图（下图中使用的数据库名字是yelp2，请忽略）



### Neo4J的安装

软件需要java环境，需要首先安装好java的环境

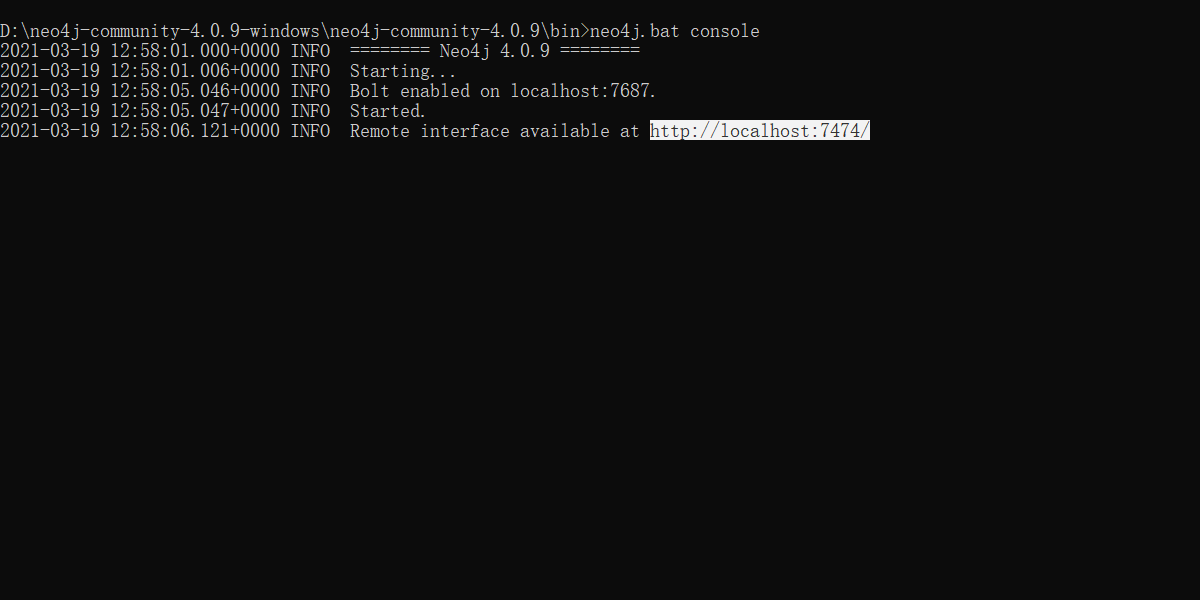
Java安装请查相关资料（需要jdk11）

可使用提供的 jdk-11.0.9\_windows-x64\_bin.exe

解压neo4j-community-4.0.9-windows.zip到自己喜欢的目录

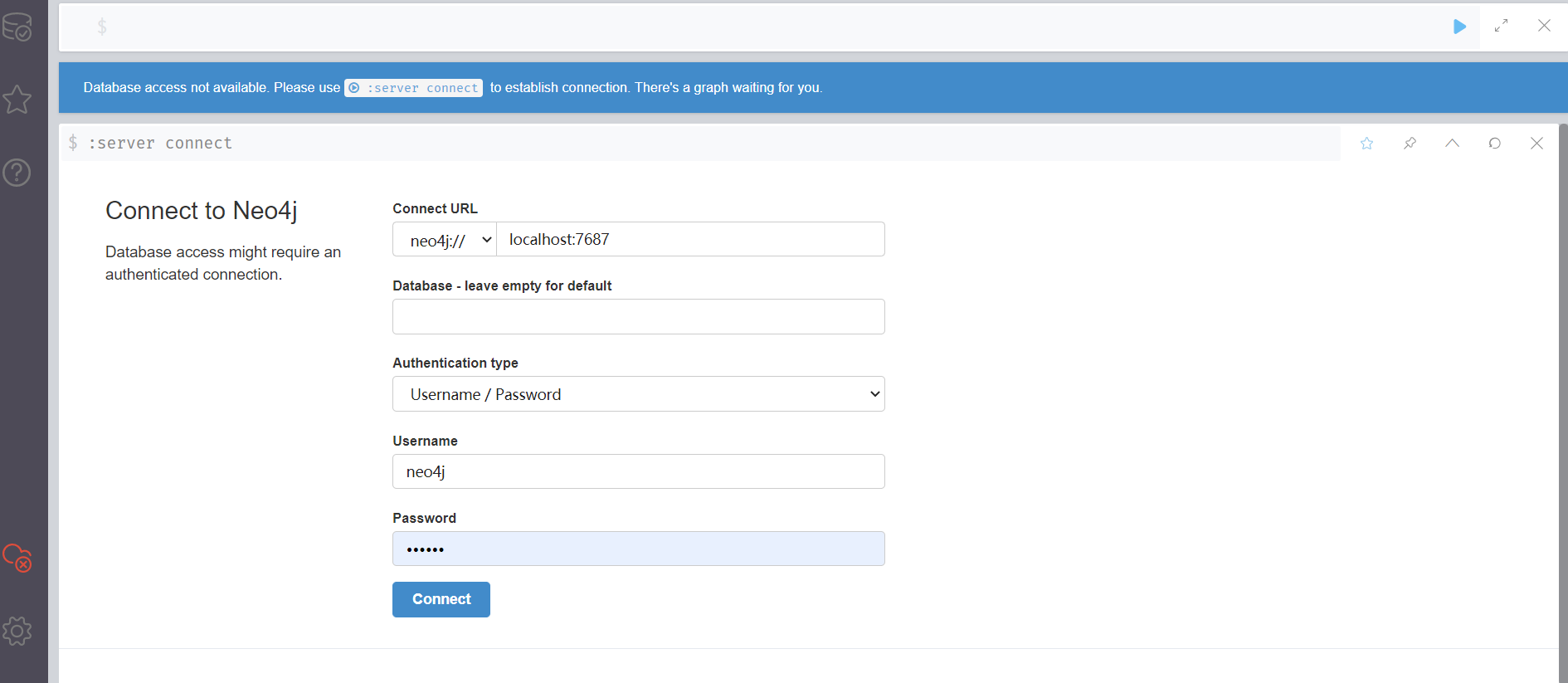
把my\_start.bat文件拷贝到安装路径下的bin目录中（例如：D:\neo4j-community-4.0.9-windows\neo4j-community-4.0.9\**bin**）然后双击这个my\_start.bat即可启动服务，

启动界面如下



在浏览器打开<http://localhost:7474/>

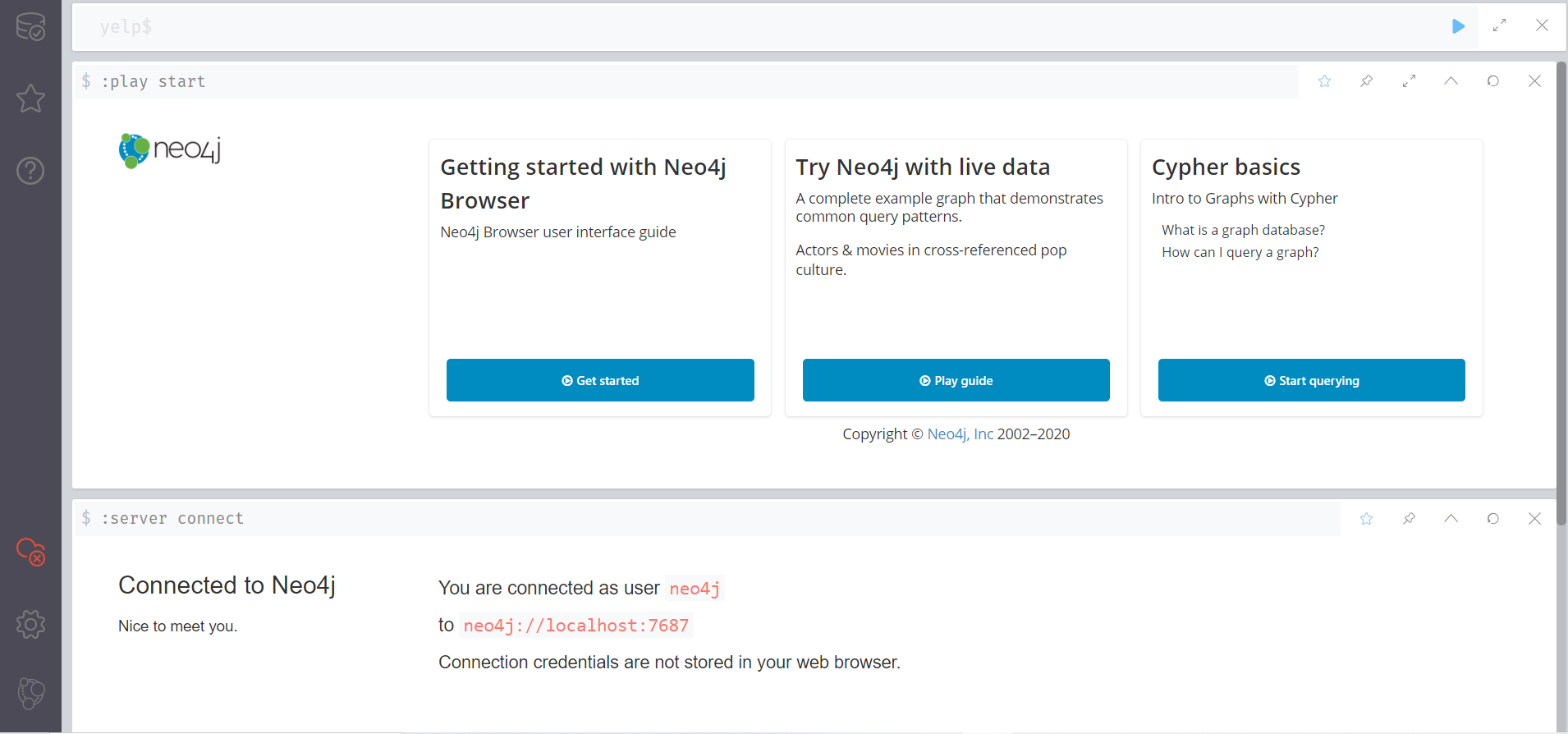
打开的界面如下



第一次登陆默认的账号密码都是neo4j

登陆之后会重定向到修改密码页面，然后修改密码，以后登陆就需要使用修改的密码

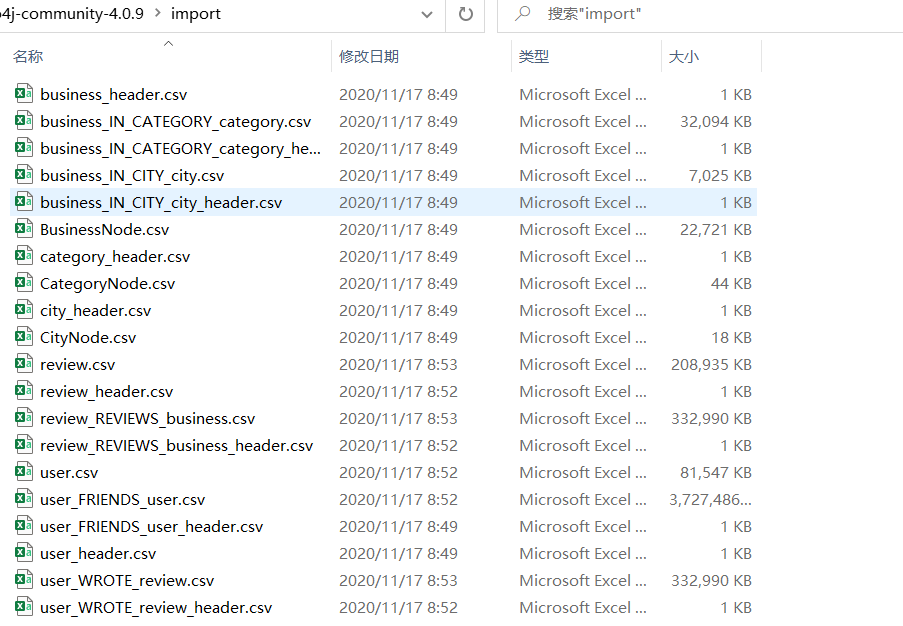
登陆成功后



接下来就能进行使用了

### 数据导入

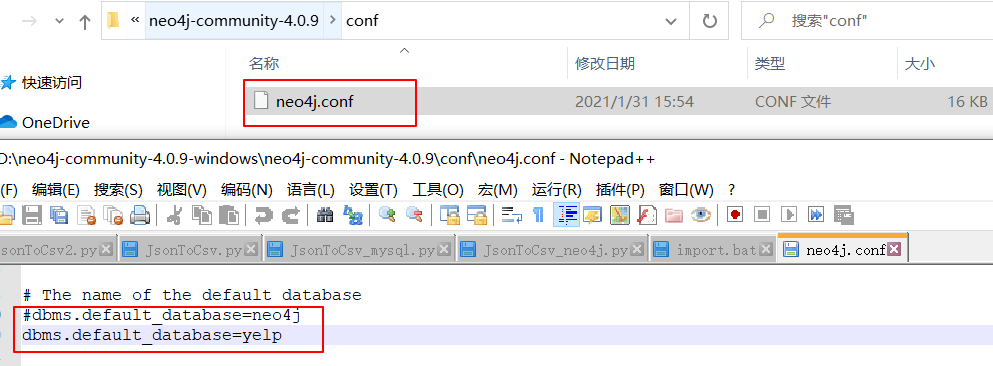
在使用之前，我们需要导入实验需要用到的数据，将我们提供的import.bat拷贝到neo4j安装的bin目录下，把data\_for\_neo4j文件夹下的数据全部拷贝到neo4j安装目录下的import文件夹下，如图



到bin目录下双击运行import.bat导入数据到系统。

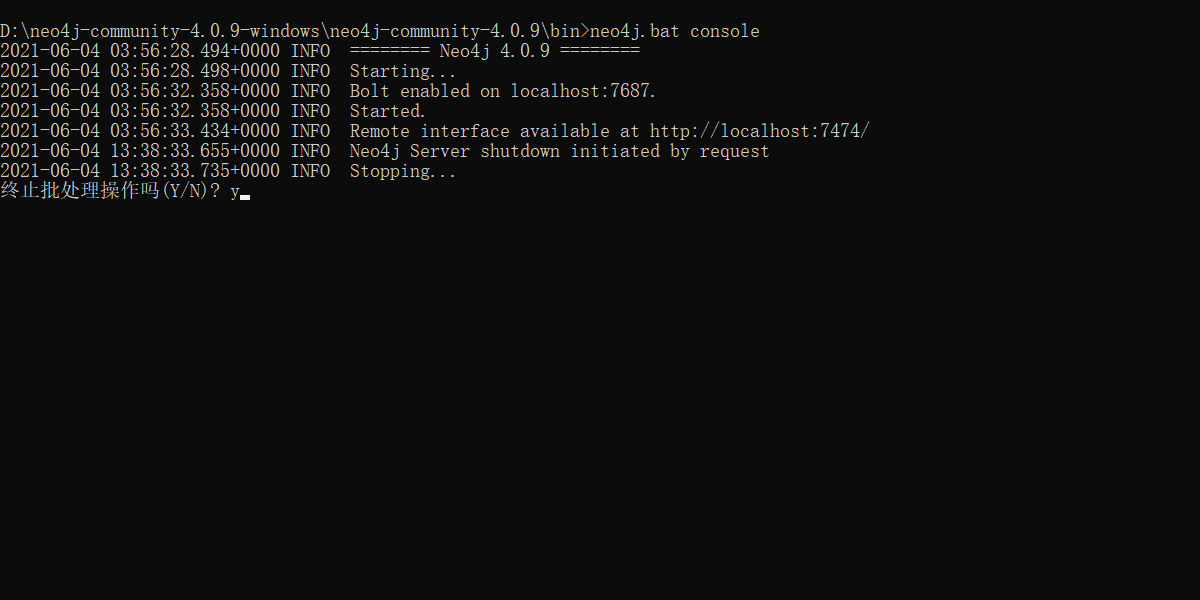
导入成功之后，需要修改 neo4j安装目下的conf/neo4j.conf文件

去掉dbms.default\_database=yelp（注意去掉前面的#）



如上图所示，修改之后重启neo4j服务

在之前启动的界面 输入Ctrl+C

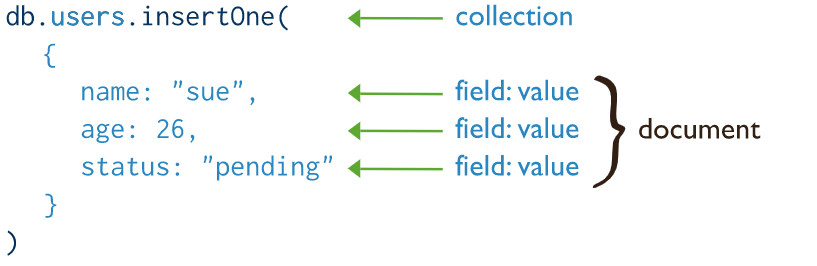


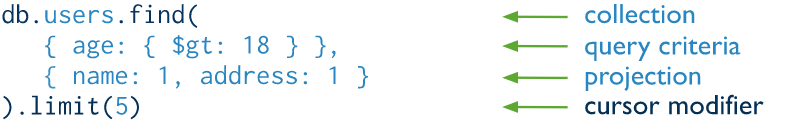
输入Y之后，再双击my\_start.bat 重启，然后F5在浏览器界面刷新即可

## 相关数据库的基本语法

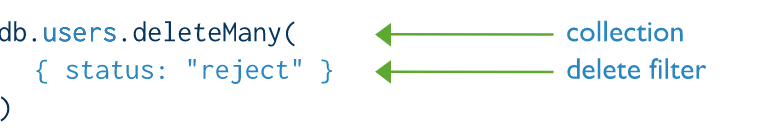
### MongoDB的基本操作

#### 基本结构









以上便是CRUD的基本结构

#### 插入：

·1. 向inventory集合中插入一个文档

db.inventory.insert\_one(

{"item": "canvas",

"qty": 100,

"tags": ["cotton"],

"size": {"h": 28, "w": 35.5, "uom": "cm"}})

1. 插入多条

db.inventory.insert\_many([

{"item": "journal",

"qty": 25,

"tags": ["blank", "red"],

"size": {"h": 14, "w": 21, "uom": "cm"}},

{"item": "mat",

"qty": 85,

"tags": ["gray"],

"size": {"h": 27.9, "w": 35.5, "uom": "cm"}},

{"item": "mousepad",

"qty": 25,

"tags": ["gel", "blue"],

"size": {"h": 19, "w": 22.85, "uom": "cm"}}])

查询：

可以先创建如下集合，进行验证

db.inventory.insert\_many([

{"item": "journal",

"qty": 25,

"size": {"h": 14, "w": 21, "uom": "cm"},

"status": "A"},

{"item": "notebook",

"qty": 50,

"size": {"h": 8.5, "w": 11, "uom": "in"},

"status": "A"},

{"item": "paper",

"qty": 100,

"size": {"h": 8.5, "w": 11, "uom": "in"},

"status": "D"},

{"item": "planner",

"qty": 75, "size": {"h": 22.85, "w": 30, "uom": "cm"},

"status": "D"},

{"item": "postcard",

"qty": 45,

"size": {"h": 10, "w": 15.25, "uom": "cm"},

"status": "A"}])

1. 选择inventory集合中所有的文档

cursor = db.inventory.find({})

1. 查询集合inventory 中‘status’**等于**“D”的文档

cursor = db.inventory.find({"status": "D"})

1. 查询inventory集合中status等于A**或者**D

cursor = db.inventory.find({"status": {"$in": ["A", "D"]}})

1. AND条件，查询集合inventory中status等于A且qty小于30

cursor = db.inventory.find({"status": "A", "qty": {"$lt": 30}})

1. OR条件，查询集合中，status等于A或者是qty小于30的

cursor = db.inventory.find(

{"$or": [{"status": "A"}, {"qty": {"$lt": 30}}]})

条件操作符：

(>) 大于 $gt

(<) 小于 $lt

(>=) 大于等于 $gte

(<= ) 小于等于 $lte

1. 返回指定的属性（投影），在find函数第二个参数指定格式{<field>:1}，以下查询返回item和status属性，不返回\_id，（如果不显示指定\_id:0, 默认会返回\_id）

cursor = db.inventory.find(

{"status": "A"}, {"item": 1, "status": 1, "\_id": 0})

1. Limit和skip使用与SQL类似，一下查询返回第二条文档

db. inventory.find({},{"status":1,\_id:0}).limit(1).skip(1)

1. 排序sort() 方法可以通过参数指定排序的字段，并使用 1 和 -1 来指定排序的方式，其中 1 为升序排列，而 -1 是用于降序排列

db. inventory.find().sort({‘qty’: 1})

1. 聚合(aggregate)主要用于处理数据(诸如统计平均值，求和等)，并返回计算后的数据结果

db.orders.aggregate([

{ $match: { status: "A" } },

{ $group: { \_id: "$cust\_id", total: { $sum: "$amount" } } }

])

这个例子，分为2个阶段执行，第一个阶段主要是使用第一条规则

Match 查询status等于A的文档

第二个阶段，按照cust\_id进行分组，计算每一个组的amount的合

以上类似管道（Pipeline）的概念，一个任务分为多个步骤来完成，前一个任务的结果作为下一个任务的输入。聚合中常用的命令有：

$project：修改输入文档的结构。可以用来重命名、增加或删除域，也可以用于创建计算结果以及嵌套文档。

$match：用于过滤数据，只输出符合条件的文档。$match使用MongoDB的标准查询操作。

$limit：用来限制MongoDB聚合管道返回的文档数。

$skip：在聚合管道中跳过指定数量的文档，并返回余下的文档。

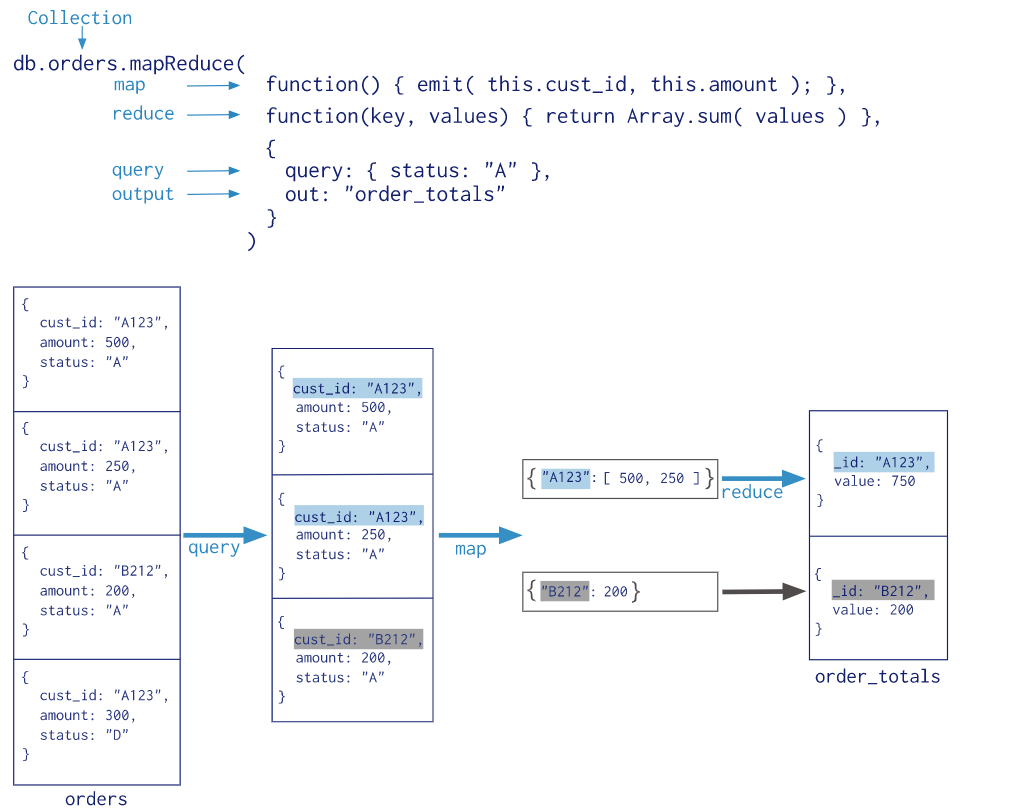
$unwind：将文档中的某一个数组类型字段拆分成多条，每条包含数组中的一个值。

$group：将集合中的文档分组，可用于统计结果。

$sort：将输入文档排序后输出。

$geoNear：输出接近某一地理位置的有序文档。

比起聚合更加强大的是MongoDB中提供的MapReduce功能



使用，需要编写 map函数，reduce函数，以及指定输出的集合

假设有如下的数据

db.orders.insertMany([

{ \_id: 1, cust\_id: "Ant O. Knee", ord\_date: new Date("2020-03-01"), price: 25, items: [ { sku: "oranges", qty: 5, price: 2.5 }, { sku: "apples", qty: 5, price: 2.5 } ], status: "A" },

{ \_id: 2, cust\_id: "Ant O. Knee", ord\_date: new Date("2020-03-08"), price: 70, items: [ { sku: "oranges", qty: 8, price: 2.5 }, { sku: "chocolates", qty: 5, price: 10 } ], status: "A" },

{ \_id: 3, cust\_id: "Busby Bee", ord\_date: new Date("2020-03-08"), price: 50, items: [ { sku: "oranges", qty: 10, price: 2.5 }, { sku: "pears", qty: 10, price: 2.5 } ], status: "A" },

{ \_id: 4, cust\_id: "Busby Bee", ord\_date: new Date("2020-03-18"), price: 25, items: [ { sku: "oranges", qty: 10, price: 2.5 } ], status: "A" },

{ \_id: 5, cust\_id: "Busby Bee", ord\_date: new Date("2020-03-19"), price: 50, items: [ { sku: "chocolates", qty: 5, price: 10 } ], status: "A"},

{ \_id: 6, cust\_id: "Cam Elot", ord\_date: new Date("2020-03-19"), price: 35, items: [ { sku: "carrots", qty: 10, price: 1.0 }, { sku: "apples", qty: 10, price: 2.5 } ], status: "A" },

{ \_id: 7, cust\_id: "Cam Elot", ord\_date: new Date("2020-03-20"), price: 25, items: [ { sku: "oranges", qty: 10, price: 2.5 } ], status: "A" },

{ \_id: 8, cust\_id: "Don Quis", ord\_date: new Date("2020-03-20"), price: 75, items: [ { sku: "chocolates", qty: 5, price: 10 }, { sku: "apples", qty: 10, price: 2.5 } ], status: "A" },

{ \_id: 9, cust\_id: "Don Quis", ord\_date: new Date("2020-03-20"), price: 55, items: [ { sku: "carrots", qty: 5, price: 1.0 }, { sku: "apples", qty: 10, price: 2.5 }, { sku: "oranges", qty: 10, price: 2.5 } ], status: "A" },

{ \_id: 10, cust\_id: "Don Quis", ord\_date: new Date("2020-03-23"), price: 25, items: [ { sku: "oranges", qty: 10, price: 2.5 } ], status: "A" }

])

统计每个顾客的总价格

1. 编写map函数，映射cust\_id， price

var mapFunction1 = function() {

emit(this.cust\_id, this.price);

};

1. 编写reduce函数，累加price

var reduceFunction1 = function(keyCustId, valuesPrices) {

return Array.sum(valuesPrices);

};

1. 在集合上使用mapReduce，指定结果到map\_reduce\_example集合中

db.orders.mapReduce(

mapFunction1,

reduceFunction1,

{ out: "map\_reduce\_example" }

)

1. 全文索引的创建，使用createIndex函数，值需要指定为text，在reviews上的comments上建立全文索引

db.reviews.createIndex( { comments: "text" } )

1. 地理索引，进行地理信息查询

首先创建索引

db.collection.createIndex( { <location field> : "2dsphere" } )

注意：用于索引的域需要是GeoJson格式或者是经纬度坐标值即<field>: [ <x>, <y> ]格式

查询格式：

db.<collection>.find( { <location field> :

{ **$near :**

{ $geometry :

{ type : "Point" ,

coordinates : [ <longitude> , <latitude> ] } ,

$maxDistance : <distance in meters>

} } } )

下面的查询返回的文档是距离指定的GeoJSON点至少1000米，最多5000米，按最近到最远排序

db.places.find(

{

location:

{ $near :

{

$geometry: { type: "Point", coordinates: [ -73.9667, 40.78 ] },

$minDistance: 1000,

$maxDistance: 5000

}

}

}

)

### MySQL的基本操作

略

### Neo4j基本操作

基本的概念：

节点：

以下都是节点

() *//匿名的节点，无法引用数据库中的节点*

(p:Person) *//变量p表示是person类型的节点*

(:Technology) *//没有变量，只知道是一个Technology类型的节点*

(work:Company) *//变量work表示是company类型的节点*

关系：

有向边：--> 或者 <--

无向边 --

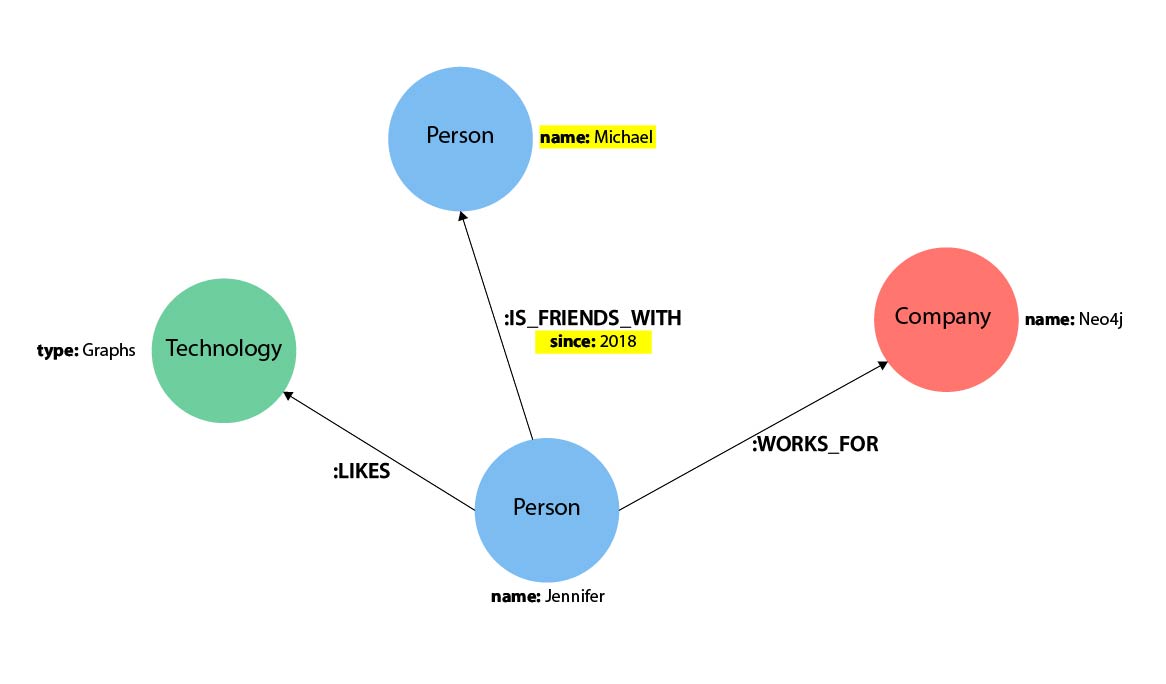
例如：(p:Person)-[:LIKES]->(t:Technology)表示 从Person类型的节点p连接到Technology类型的节点t的有向边（箭头的方向），边的类型是LIKES。

(p:Person)-[:LIKES]-(t:Technology) 同理，不过这次是无向边。

属性：节点，关系都有属性

节点属性：(p:Person {name: 'Jennifer'})

关系属性：-[rel:IS\_FRIENDS\_WITH {since: 2018}]->



#### Neo4J Cypher简单的查询

1. 查询标签是Person的节点

MATCH (p:Person)

RETURN p

LIMIT 1

1. 查询标签是Perosn，同时name是Tom Hanks的节点

MATCH (tom:Person {name: 'Tom Hanks'})

RETURN tom

1. 找出哪一步电影是Tom Hanks指导过的

MATCH (:Person {name: 'Tom Hanks'})-[:DIRECTED]->(movie:Movie)

RETURN movie

1. 同上一个查询，但是只需要返回电影的title

MATCH (:Person {name: 'Tom Hanks'})-[:DIRECTED]->(movie:Movie)

RETURN movie.title

1. 查询同上，但是为了结果的可读性，可以使用AS给属性起一个别名

MATCH (tom:Person {name:'Tom Hanks'})-[rel:DIRECTED]-(movie:Movie)

RETURN tom.name AS name, tom.born AS `Year Born`, movie.title AS title, movie.released AS `Year Released`

#### Neo4J过滤查询

1. 查找名字是Jennifer的Person标签类型节点

MATCH (j:Person {name: 'Jennifer'})

RETURN j

1. 同上查询使用where

MATCH (j:Person)

WHERE j.name = 'Jennifer'

RETURN j

1. Not使用，查询名字不是Jennifer的Person节点

MATCH (j:Person)

WHERE NOT j.name = 'Jennifer'

RETURN j

1. 范围查询

MATCH (p:Person)

WHERE 3 <= p.yearsExp <= 7

RETURN p

1. 找出Jennifer的没有在公司工作的朋友

MATCH (p:Person)-[r:IS\_FRIENDS\_WITH]->(friend:Person)

WHERE p.name = 'Jennifer'

AND NOT exists((friend)-[:WORKS\_FOR]->(:Company))

RETURN friend.name

#### Neo4J聚集/统计查询

1. 统计Person类型节点的数量

MATCH (p:Person)

RETURN count(\*)

1. 使用collect把属性转变为列表，下述查询使用name分组，显示每个人的朋友列表

MATCH (p:Person)-[:IS\_FRIENDS\_WITH]->(friend:Person)

RETURN p.name, collect(friend.name) AS friend

1. Size函数返回集合大小

MATCH (p:Person)-[:IS\_FRIENDS\_WITH]->(friend:Person)

RETURN p.name, size(collect(friend.name)) AS numberOfFriends

1. 使用with进行链式查询。查询用户的twitter非空或者type中包含‘Graphs’或者‘Query Languages’

MATCH (user:Person)

WHERE user.twitter IS NOT null

WITH user

MATCH (user)-[:LIKES]-(t:Technology)

WHERE t.type IN ['Graphs','Query Languages']

RETURN DISTINCT user.name

1. 查询有最多朋友的TOP3用户

MATCH (p:Person)-[r:IS\_FRIENDS\_WITH]-(other:Person)

RETURN p.name, count(other.name) AS numberOfFriends

ORDER BY numberOfFriends DESC

LIMIT 3

1. 匹配从n到m，任意关系、任意深度的节点

MATCH p=(n)-[\*]->(m) RETURN p

1. 匹配分页返回

MATCH (n) WHERE n.productName='apple' RETURN n SKIP 10 LIMIT 10

1. 匹配去重返回

MATCH (n) WHERE n.productName=’apple' RETURN DISTINCT n

1. 匹配分组返回值

MATCH (n) WHERE n.productName='apple' RETURN n ORDER BY n.price DESC