hw8: MongoDB & Neo4J

Task
Part 1 mongodb
    什么样的数据适合放到mongodb中
    实现效果
Part 2 Neo4j
    install neo4j on mac
        spring创建关系
    实现结果
    附录
reference
    neo4j语法

Task

请你在大二开发的E-Book系统的基础上，完成下列任务：

1. 将你认为合适的内容改造为在MongoDB中存储，例如书的产品评价或书评。你可以参照课程样例将数据分别存储在MySQL和MongoDB中，也可以将所有数据都存储在MongoDB中，如果采用后者，需要确保系统功能都能正常实现，包括书籍浏览、查询、下订单和管理库存等。
2. 为你的每一本图书都添加一些标签，在Neo4J中将这些标签构建成一张图。在系统中增加一项搜索功能，如果用户按照标签搜索，你可以将Neo4J中存储的与用户选中的标签以及通过2重关系可以关联到的所有标签都选出，作为搜索的依据，在MySQL中搜索所有带有这些标签中任意一个或多个的图书，作为图书搜索结果呈现给用户。

–请将你的工程所有源代码和资源文件压缩后上传，请勿压缩编译后生成文件和依赖的第三方包

评分标准：

1. 能够正确实现上述数据存储方案。(3分)
2. 能够正确实现图书标签图在Neo4J中的构建与存储，并实现针对标签图的模糊搜索功能(2分)

安装mongodb

参考：https://www.runoob.com/mongodb/mongodb-osx-install.html

接下来我们使用以下命令在后台启动 mongodb：

```
mongod --dbpath /usr/local/var/mongodb --logpath /usr/local/var/log/mongodb/mongo.log --fork
```

- **-dbpath** 设置数据存放目录
- **-logpath** 设置日志存放目录
- **-fork** 在后台运行

如果不想在后端运行，而是在控制台上查看运行过程可以直接设置配置文件启动：

```
mongod --config /usr/local/etc/mongod.conf
```

查看 mongod 服务是否启动：

```
ps aux | grep -v grep | grep mongod
```

使用以上命令如果看到有 mongod 的记录表示运行成功。

启动后我们可以使用 **mongo** 命令打开一个终端：

```
$ cd /usr/local/mongodb/bin
$ ./mongo
MongoDB shell version v4.0.9
connecting to: mongodb://127.0.0.1:27017/?gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("3c12bf4f-695c-48b2-b160-8420110ccdcf") }
MongoDB server version: 4.0.9
……
> 1 + 1
2
>
```

Part 1 mongodb

什么样的数据适合放到mongodb中

mongodb里的数据是从MySQL中清洗出来存到mongodb中的，mongodb只做单点的业务需求，综合的数据还是在MySQL中。

mongodb的优势就是文档存储:

1. 业务经常变动，需要不时的添加字段，那么mongodb比较适合，关系型数据库添加字段的复杂度也还好

2. 嵌套文档，业务数据比较复杂，适合嵌套文档式存储，那么mongodb非常合适，这个关系型数据库比较难搞

在我的实现中，选择将书评（description）放到mongodb里面？

pom.xml

```
// pom.xml
// 配置dependency,注意不带版本号自动匹配不然会出错
...
<dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-mongodb</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-rest</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.data</groupId>
            <artifactId>spring-data-mongodb</artifactId>
 </dependency>
```

```
// application.properties
...
// root是用户名, password是密码 27017是端口 myBookStoreDataBase是数据库
spring.data.mongodb.uri=mongodb://root:password@localhost:27017/myBookStoreDataBase
```

遇到问题：Command failed with error 18 (AuthenticationFailed)

这个是因为在mongoDB还没有验证，于是

```
>use admin

switched to db admin

>db.auth("root","password")

1

>show dbs

admin                0.000GB
config               0.000GB
local                0.000GB
myBookStoreDataBase  0.000GB

>use myBookStoreDataBase

switched to db myBookStoreDataBase

>db.createUser({user:"root",pwd:"password",roles:[{role:"dbOwner",db:"myBookStoreDataBase"}]})

Successfully added user: {
  "user" : "root",
  "roles" : [
    {
      "role" : "dbOwner",
      "db" : "myBookStoreDataBase"
    }
  ]
}
```

BookDescriptionRepository.java

```
package com.reins.bookstore.repository;
import com.reins.bookstore.entity.BookDescription;
import org.springframework.data.mongodb.repository.MongoRepository;

public interface BookDescriptionRepository  extends MongoRepository<BookDescription, Integer> {

}
```

BookDescription.java

```
package com.reins.bookstore.entity;

import javax.persistence.Id;

public class BookDescription {
    @Id
    private int id;
    private String description;

    public BookDescription(int id, String description) {
        this.id = id;
        this.description = description;
    }
```

```
    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }
}
```

BookDaoImpl

autowired 加入BookDescriptionRepository

get

每次从BookRepository 拿到一本书返回之前先从BookDescriptionRepository用id拿到简介进行拼装。

```java
@Override
public Book findOne(Integer id){
    Book book=null;
    System.out.println("Searching Book: " + id + " in Redis");
    Object b = redisUtil.get("book" + id);
    if(b==null){
        System.out.println("Book: "+id+" is not in Redis");
        System.out.println("Searching Book: "+id+" in DB");
        book = bookRepository.getById(id);
        book.setDescription(bookDescriptionRepository.findById(id).get().getDescription());
        //debug
        System.out.println("findOne: description:");
        System.out.println(book.getDescription());

        redisUtil.set("book"+id,JSONArray.toJSON(book));
    }else{
        book = JSONArray.parseObject(b.toString(),Book.class);
        book.setDescription(bookDescriptionRepository.findById(id).get().getDescription());
        //debug
        System.out.println("findOne: description:");
        System.out.println(book.getDescription());
        System.out.println("Book: "+id+" is in Redis");
    }
    return book;
}
```

```java
@Override
public List<Book> getBooks(Integer page) {
    List<Book>bookList = new ArrayList<Book>();
    List<Integer> bookIdList =new ArrayList<Integer>();
    List<Book>targetList =  new ArrayList<Book>();
    Object bIdList=redisUtil.get("bookIdList");

    if(bIdList==null){
        // refresh the cache
        bookList=bookRepository.findAll();
        for (int i = 0; i < bookList.size(); i++) {
            Book b=bookList.get(i);
            b.setDescription(bookDescriptionRepository.findById(b.getBookId()).get().getDescription());
            //debug
            System.out.println("getBooks: description:");
            System.out.println(b.getDescription());

            redisUtil.set("book"+b.getBookId(),JSONArray.toJSON(b));
            bookIdList.add(b.getBookId());
            System.out.println("set Book in redis: "+bookList.get(i).getBookId());
        }
        redisUtil.set("bookIdList",JSONArray.toJSON(bookIdList));
        // prepare the result
        for(int i=4*page-3 ;i<bookList.size() && i<=4*page;i++){
            targetList.add(bookList.get(i));
        }
    }
    else{
        bookIdList=JSONArray.parseArray(bIdList.toString(),Integer.class);
        for(int i=4*page-3 ;i<bookIdList.size() && i<=4*page;i++){
            targetList.add(findOne(bookIdList.get(i)));
        }
```

insert: 插入数据的时候在mongodb中也插入

```java
@Override
public Book addBook( String isbn, String name, String type, String author, Integer price, String description, Integer inventory, String image) {
    Book book= new Book(isbn,name, type, author, price, description, inventory, image);
    Book b=bookRepository.save(book);
    b.setDescription(description);
    BookDescription bookDescription=new BookDescription(b.getBookId(),description);
    bookDescriptionRepository.save(bookDescription);
    //debug
    System.out.println("add book: description:");
    System.out.println(bookDescriptionRepository.findById(b.getBookId()));


    redisUtil.set("book"+b.getBookId(),JSONArray.toJSON(b));
    updateIdList( add: true,b.getBookId());
    return book;
}
```

delete: 删除的时候在mongodb也删除

```java
@Override
public Book deleteBook(Integer bookId) {
    Optional<Book> b=bookRepository.findById(bookId);
    bookRepository.deleteById(bookId);
    bookDescriptionRepository.deleteById(bookId);
        //isPresent方法用来检查Optional实例中是否包含值
        if (b.isPresent()) {
            //在Optional实例内调用get()返回已存在的值
            // update cache
            redisUtil.del( ...key: "book"+b.get().getBookId());
            updateIdList( add: false,b.get().getBookId());
            return b.get();
        }else return null;
}
```

update

```java
@Override
public Book changeBook(Integer id,String isbn, String name, String type, String author, Integer price, String description, Integer inventory, Stri
    Book b=this.findOne(id);
    BookDescription bd = bookDescriptionRepository.findById(id).get();
    b.setBookId(id);
    b.setIsbn(isbn);
    b.setName(name);
    b.setType(type);
    b.setAuthor(author);
    b.setPrice(price);
    b.setDescription(description);
    b.setInventory(inventory);
    bookRepository.save(b);
    bd.setDescription(description);
    bookDescriptionRepository.save(bd);
    // update cache
    redisUtil.set("book"+b.getBookId(),JSONArray.toJSON(b));
    return b;
}
```

实现效果

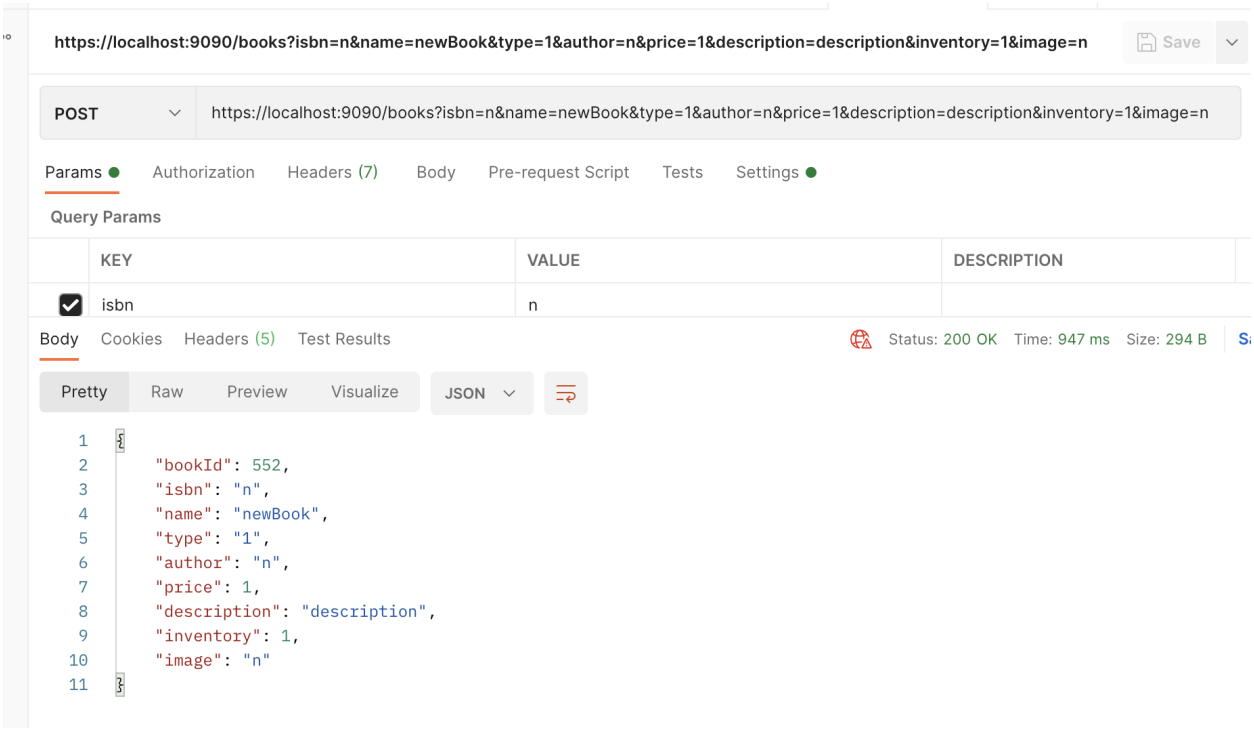
get

可以看到拿到了完整是书的数据

```
findOne: description:
Steven Paul Jobs (/dʒɒbz/; February 24, 1955 – October 5, 2011) was an American business magnate, industrial designer,
```

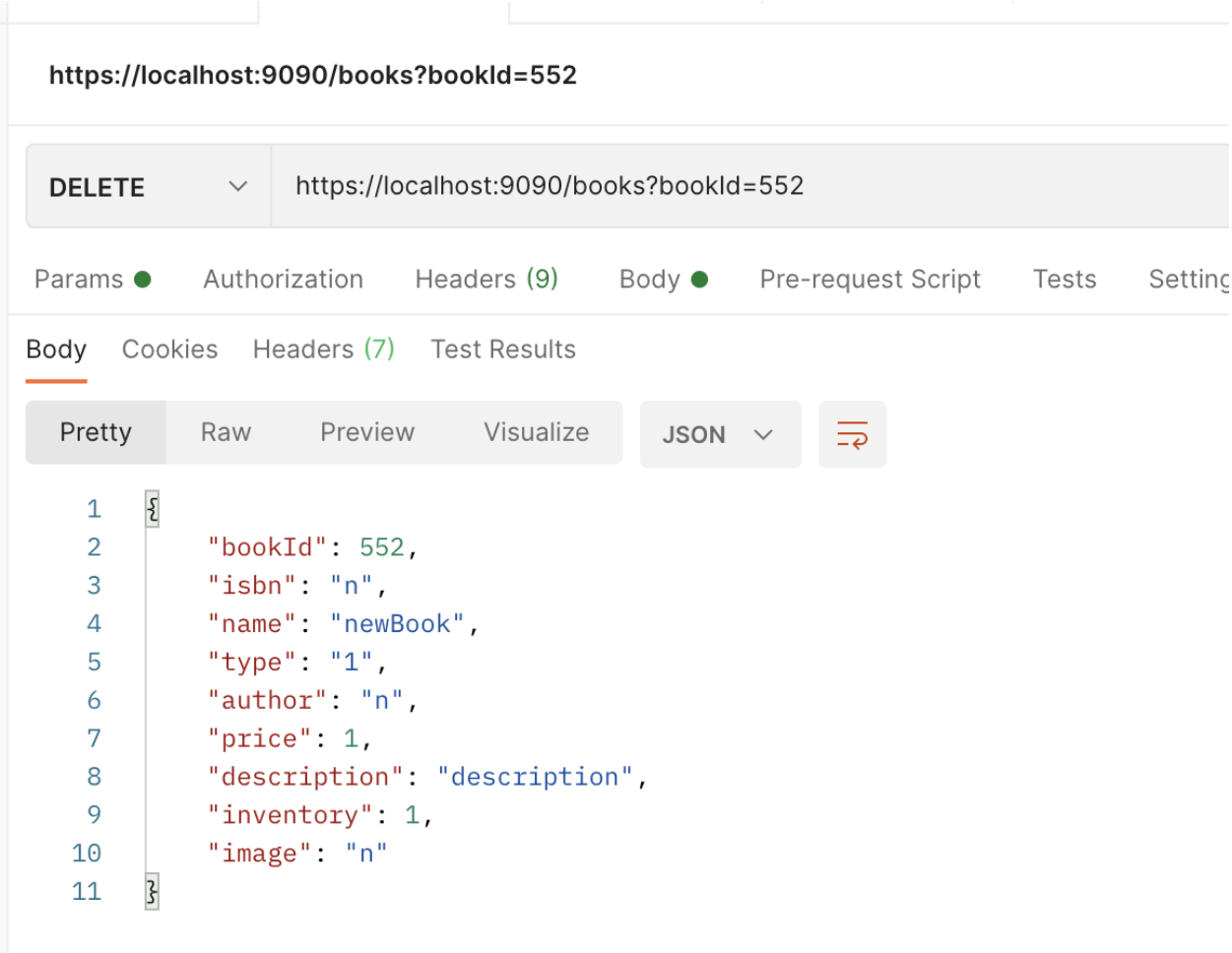
insert

可以看见在mongodb的数据库中成功插入数据

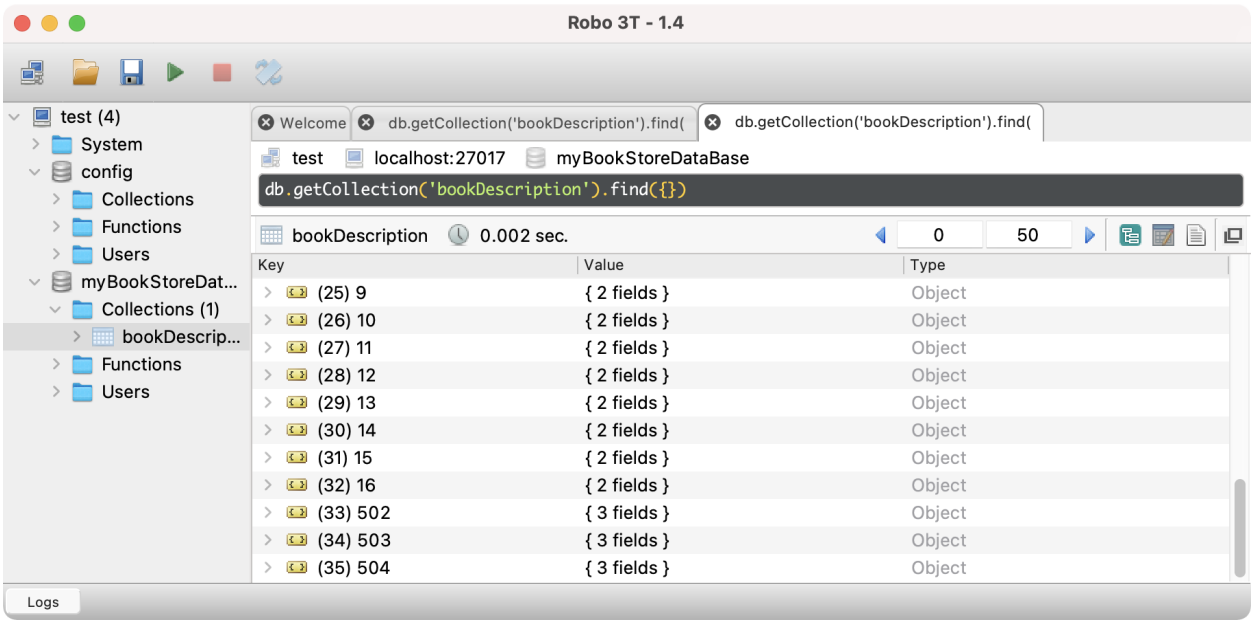


(36) 552	{ 3 fields }	Object
_id	552	Int32
description	description	String
_class	com.reins.bookstore.entity.BookDescript...	String

delete

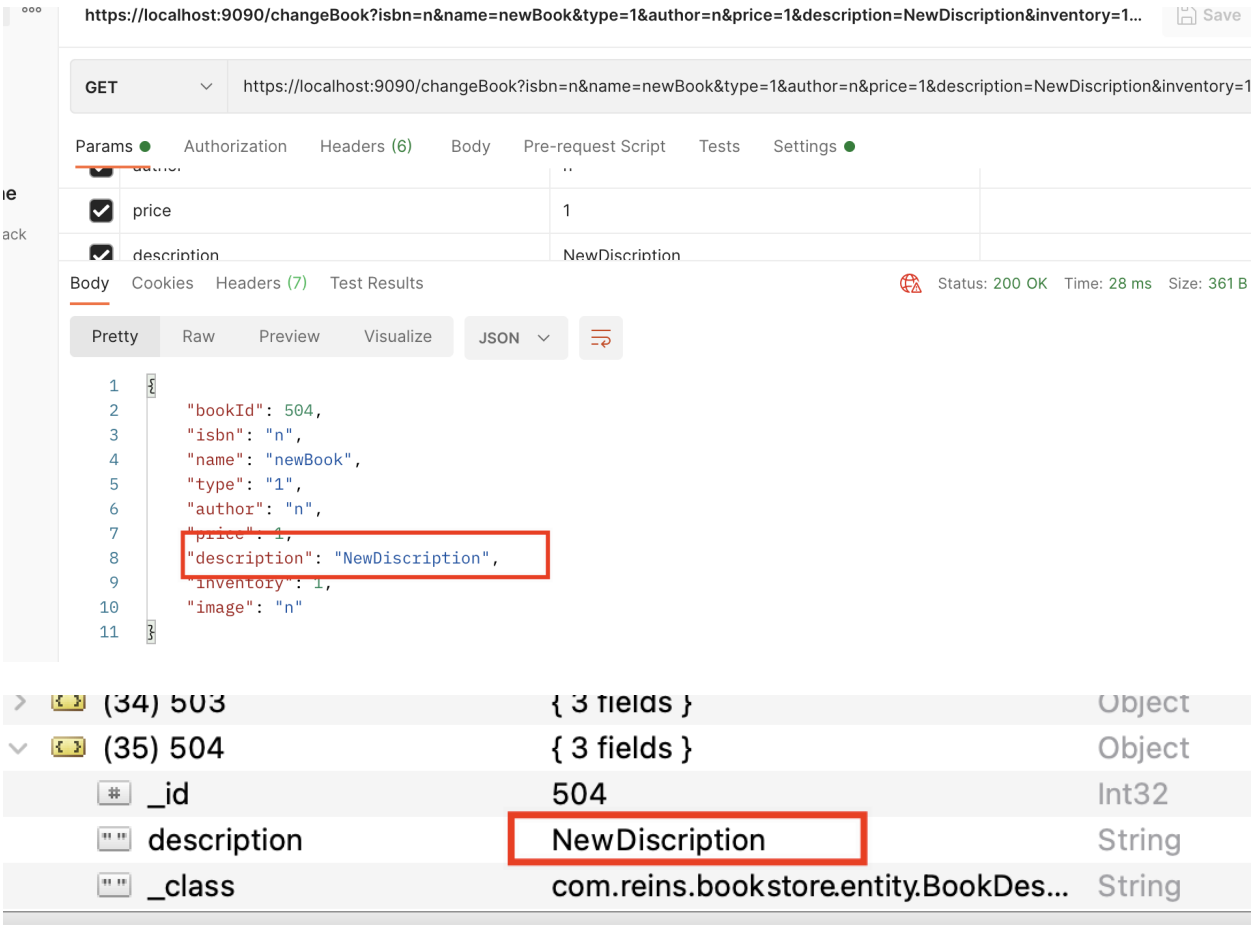


可以看见这条记录也从mongodb数据库中删除了



update

可以看到mongodb中的数据刷新了



Part 2 Neo4j

install neo4j on mac

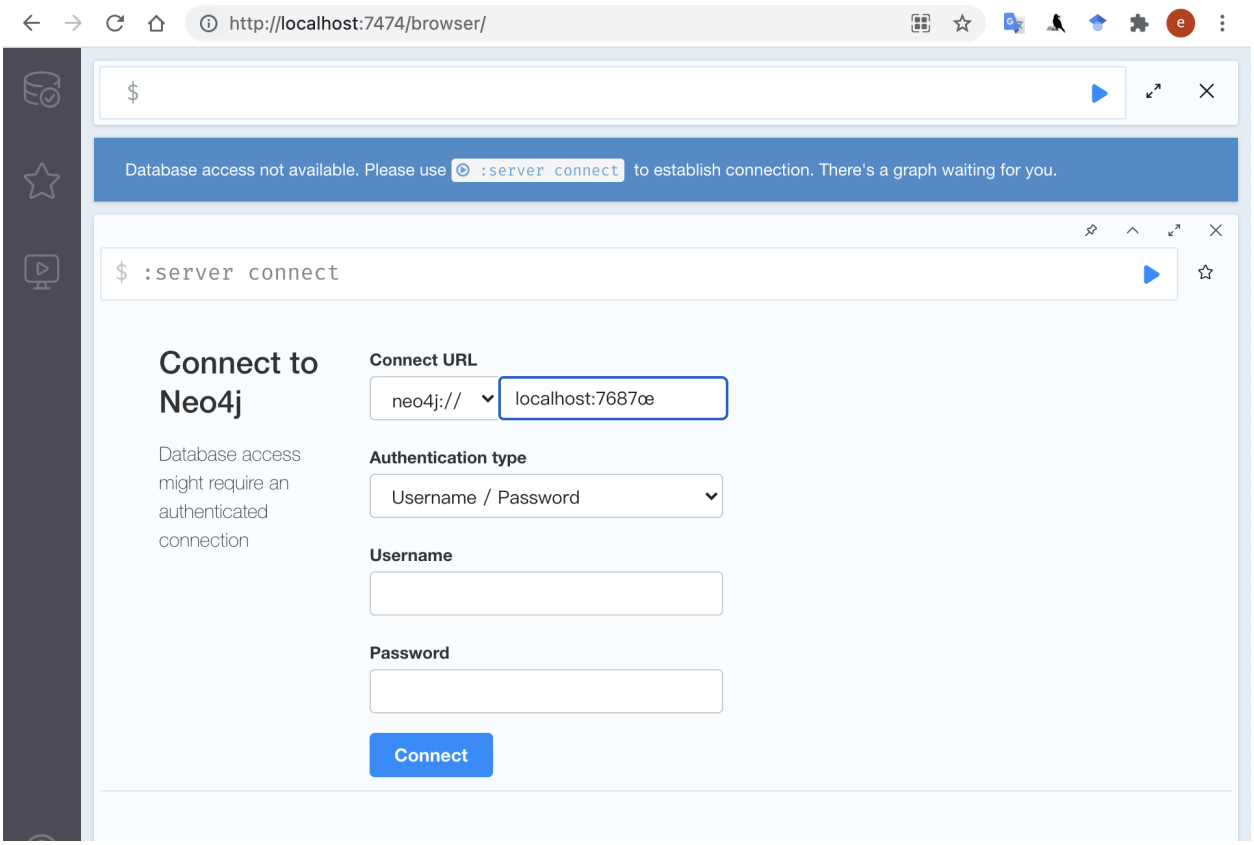
brew install neo4j

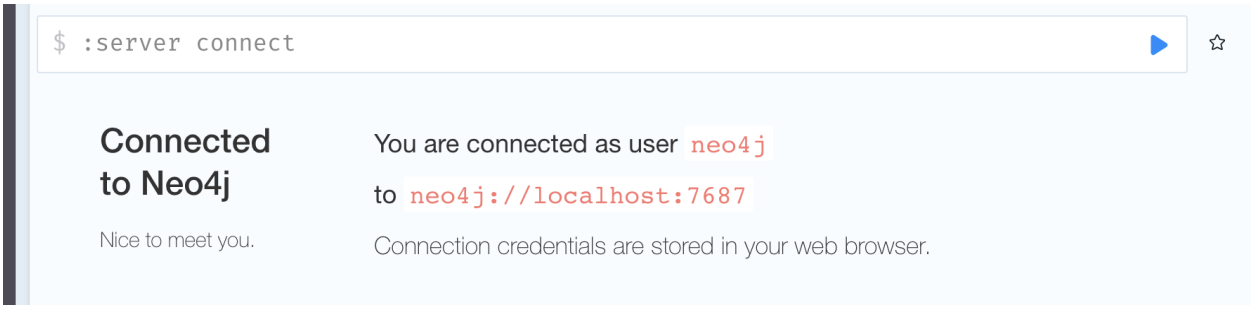
启动 neo4j start

关闭 neo4j stop

在浏览器中打开http://localhost:7474/

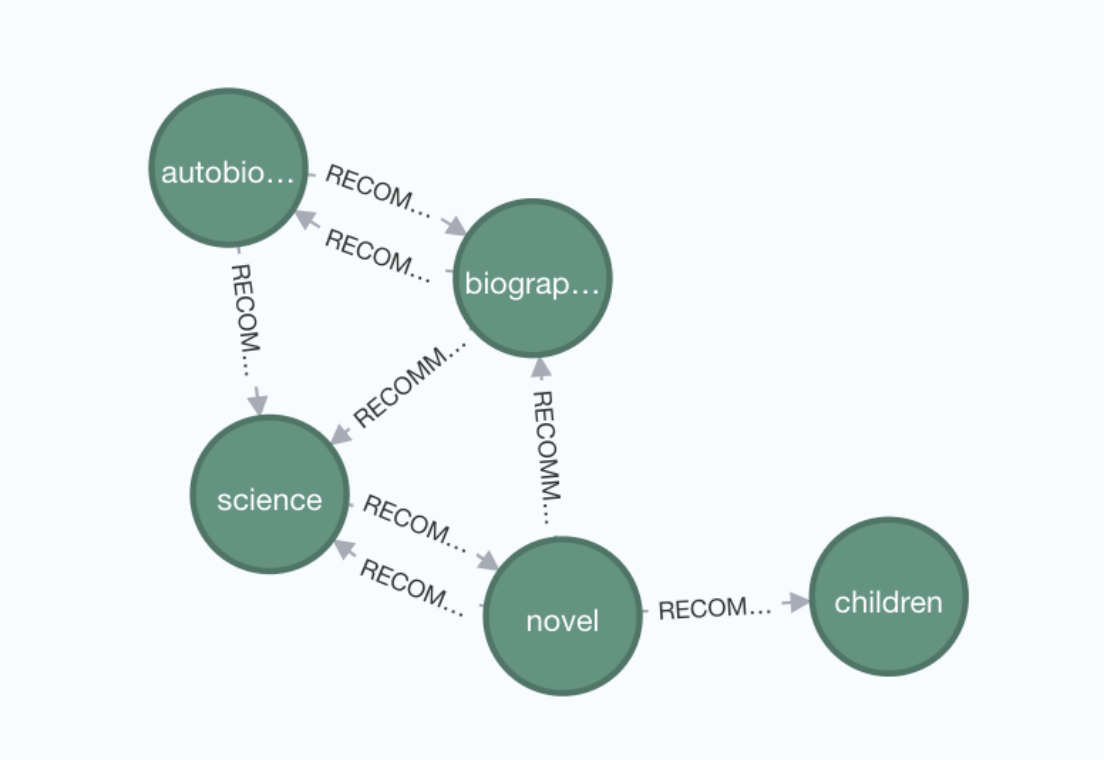
默认的用户名和密码都是neo4j，点击Connect之后输入新密码password





本数据库的实现是：不同书籍属于的type有相互推荐关系，当搜索某一类type(eg.novel），会搜索出所有novel类的书籍以及通过recommend关系的一重和二重推荐关系。这些会放到前端分别展示。

neo4j中的推荐关系：有5种书：novel, children,science, biographies, autobiographies



spring创建关系

BookType.js

```
package com.reins.bookstore.entity;

import java.util.Collections;
import java.util.HashSet;
import java.util.Optional;
import java.util.Set;
import java.util.stream.Collectors;

import org.springframework.data.neo4j.core.schema.Id;
import org.springframework.data.neo4j.core.schema.Node;
import org.springframework.data.neo4j.core.schema.Property;
import org.springframework.data.neo4j.core.schema.Relationship;
import org.springframework.data.neo4j.core.schema.GeneratedValue;

@Node
public class BookType {

    @Id @GeneratedValue private Long id;

    private String name;

    private BookType() {
        // Empty constructor required as of Neo4j API 2.0.5
    };

    public BookType(String name) {
        this.name = name;
    }
```

```
    /**
     * Neo4j doesn't REALLY have bi-directional relationships. It just means when querying
     * to ignore the direction of the relationship.
     * https://dzone.com/articles/modelling-data-neo4j
     */
    @Relationship(type = "RECOMMEND")
    public Set<BookType> recommends;

    public void recommend(BookType person) {
        if (recommends == null) {
            recommends = new HashSet<>();
        }
        recommends.add(person);
    }

    public String toString() {

        return this.name + "'s recommends => "
                + Optional.ofNullable(this.recommends).orElse(
                Collections.emptySet()).stream()
                .map(BookType::getName)
                .collect(Collectors.toList());
    }


    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

BookstoreApplication.java

```
    @Bean
CommandLineRunnerdemo(BookTypeRepositorybookTypeRepository) {
returnargs -> {

            bookTypeRepository.deleteAll();

            BookType novel =newBookType("novel");
            BookType children =newBookType("children");
            BookType biographies =newBookType("biographies");
            BookType autobiographies =newBookType("autobiographies");
            BookType science =newBookType("science");
List<BookType> team = Arrays.asList(novel, children, biographies, autobiographies,science);

bookTypeRepository.save(novel);
            bookTypeRepository.save(children);
            bookTypeRepository.save(biographies);
            bookTypeRepository.save(autobiographies);
            bookTypeRepository.save(science);

            novel = bookTypeRepository.findByName(novel.getName());
            novel.recommend(children);
            novel.recommend(biographies);
            novel.recommend(science);
            bookTypeRepository.save(novel);

            // test no recommend
            children = bookTypeRepository.findByName(children.getName());
            bookTypeRepository.save(children);

            science = bookTypeRepository.findByName(science.getName());
            science.recommend(novel);
            bookTypeRepository.save(science);

            autobiographies = bookTypeRepository.findByName(autobiographies.getName());
```

```
            autobiographies.recommend(science);
            autobiographies.recommend(biographies);
            bookTypeRepository.save(autobiographies);

            biographies = bookTypeRepository.findByName(biographies.getName());
            biographies.recommend(autobiographies);
            biographies.recommend(science);
            bookTypeRepository.save(biographies);

};
```

BookDaoImpl.js

这里的两个函数分别返回一二重推荐结果和同type的查询结果

```
@Override
    public List<Book> recommendBooks(String recommendType) {
        BookType bookType=bookTypeRepository.findByName(recommendType);
//        log.info(recommendType+" recommends :");

        Set<BookType> recommendList=bookType.recommends;
        log.info( recommendList.toString());
        System.out.println(recommendList.size());
        List<Book>bookList =new ArrayList<>();
        List<String> rec1Type=new ArrayList<>();
        for (BookType it :recommendList ) {
            String type=it.getName();
            rec1Type.add(type);
            bookList.addAll(bookRepository.findByType(type));
        }

        List<BookType> recmList2 =bookTypeRepository.findByRecommendsName2(recommendType);
//        log.info(recommendType+" recommends 2 depth :"+recmList2.toString());
        for (BookType it :recmList2 ) {
            String type=it.getName();
            if(!rec1Type.contains(type) && !type.equals(recommendType))
            bookList.addAll(bookRepository.findByType(type));
        }
        return bookList;
    }

    @Override
    public List<Book> booksOfType(String type) {
        BookType bookType=bookTypeRepository.findByName(type);
        return bookRepository.findByType(bookType.getName());
    }
```

BookDao.java

```
@GetMapping(path = "/books", params = "recommendType")
    public List<Book> recommendBooks(@RequestParam("recommendType") String recommendType){
        return bookService.recommendBooks(recommendType);
    }

    @GetMapping(path = "/books", params = "type")
    public List<Book> booksOfType(@RequestParam("type") String type){
        return bookService.booksOfType(type);
    }
```

BookTypeRepository.java

这里用query实现了查找二重推荐，一重推荐因为之前在保存的时候已经存在了BookType的recommends数组里面，在findByName之后可以直接调用这个数组，两者拼合就得到了所有的推荐图书。

```
package com.reins.bookstore.repository;
```

```
import java.util.List;

import com.reins.bookstore.entity.BookType;
import org.springframework.data.neo4j.repository.Neo4jRepository;
import org.springframework.data.neo4j.repository.query.Query;

public interface BookTypeRepository extends Neo4jRepository<BookType, Long> {

    BookType findByName(String name);
    List<BookType> findByRecommendsName(String name);

    //Find double recommendations
    @Query("Match (n:BookType {name:$name})-[:RECOMMEND]->()-[:RECOMMEND]->(m:BookType) RETURN m")
    List<BookType> findByRecommendsName2(String name);
}
```

BookServiceImpl.java

```
@Override
    public List<Book> recommendBooks(String recommendType) {
        return bookDao.recommendBooks(recommendType);
    }
```

BookService.java

```
List<Book> recommendBooks(String recommendType);
```

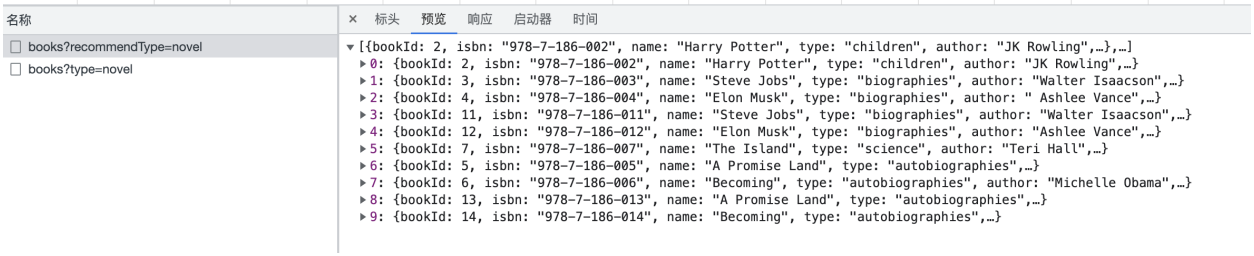
BookController.js

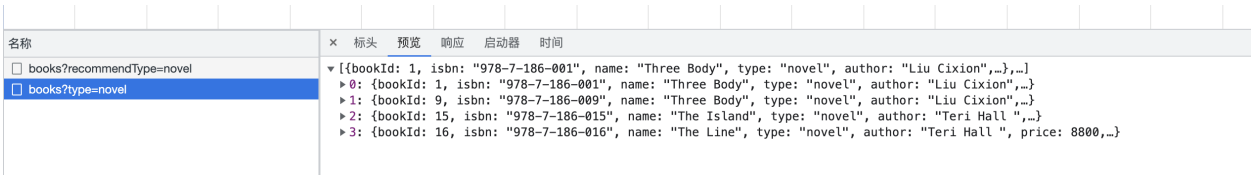
```
...
@GetMapping(path = "/books", params = "recommendType")
    public List<Book> recommendBooks(@RequestParam("recommendType") String recommendType){
        return bookService.recommendBooks(recommendType);
    }

    @GetMapping(path = "/books", params = "type")
    public List<Book> booksOfType(@RequestParam("type") String type){
        return bookService.booksOfType(type);
    }
```

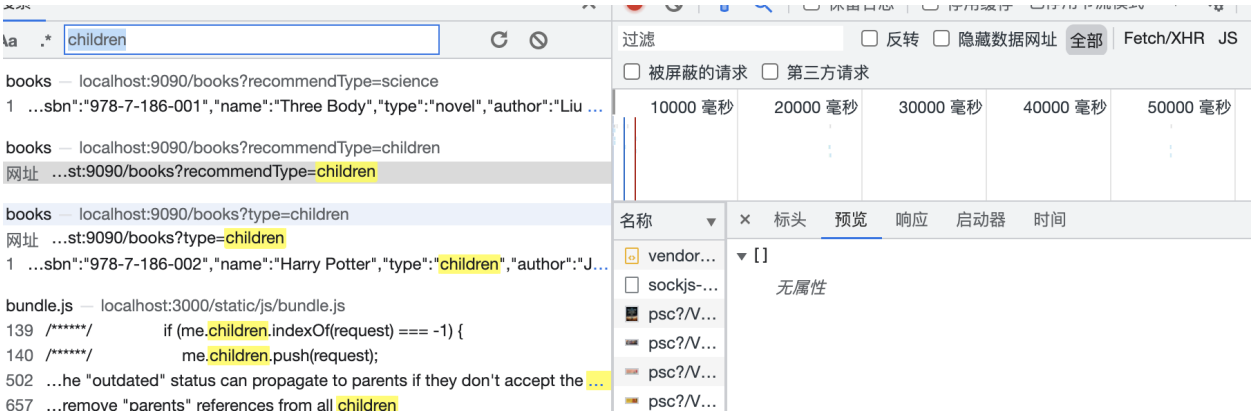
后端数据：

1. 选择type:novel

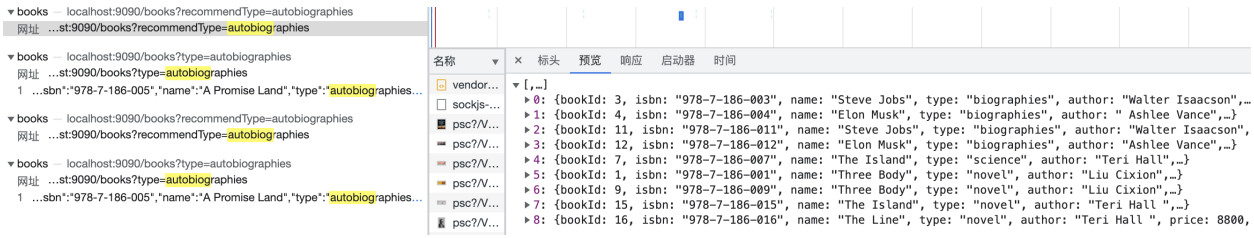




可以看见recommendType=novel的时候推荐的书籍的type包括：children,biographies, science, autobiographies, 这个和neo4j的图关系相符合



测试没有推荐其他类的children类，确实没有获得推荐内容



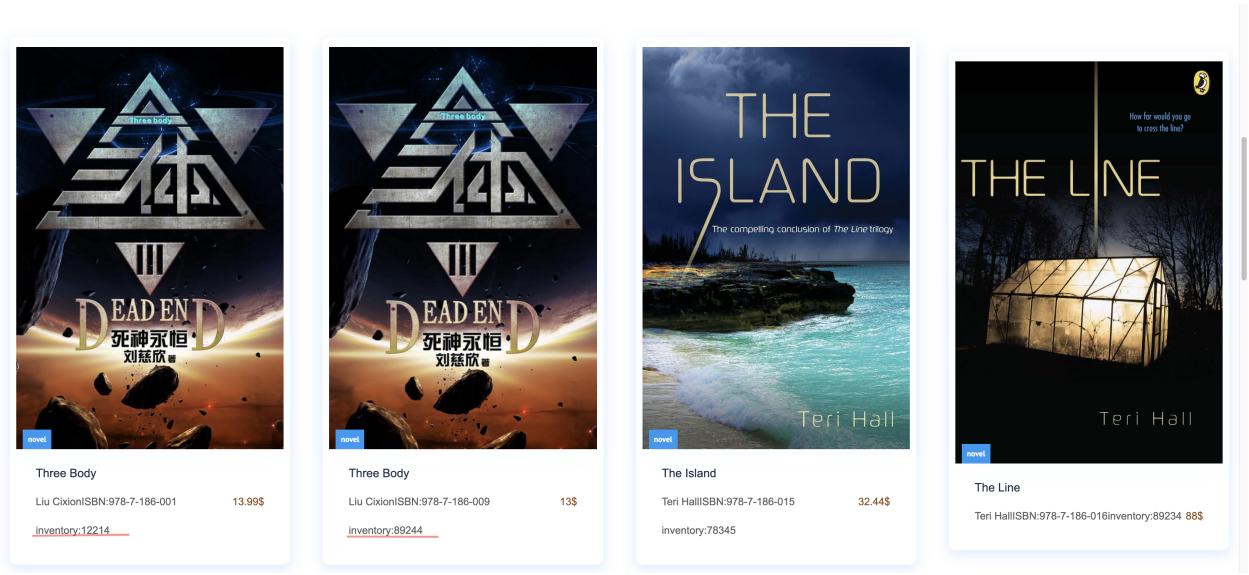
测试autobiographies 类，推荐了novel, biographies, science 和neo4j的图匹配。

实现结果

输入type:novel

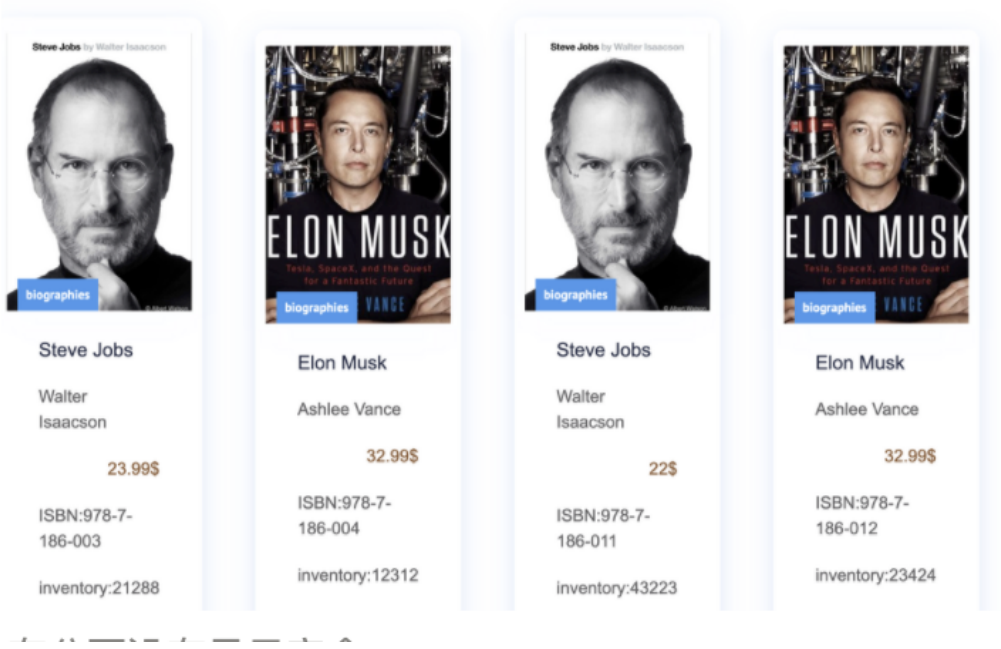
前端展示：

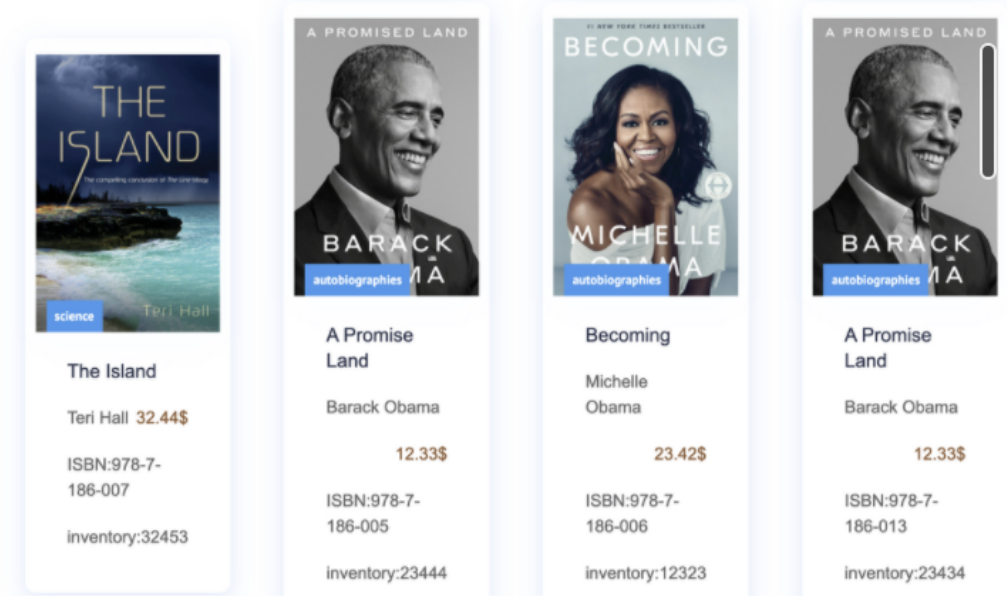
同类型搜索：注意第一二本书不一样（见isbn和inventory只是封面设置一样了）



一重和二重推荐

（有分页，未显示完全，全部数据可见上方的后端API返回数据）





附录

neo4j建立图的另一种方法尝试，直接把书籍的信息而不是type放到数据库里面，每一种书籍就是一个type, 然后不同type相会推荐。我觉得这样的好处是，不用拿到type之后再在mysql中找，性能提升，但是坏处是mysql每次插入删除一本书，neo4j数据库也要跟着改变，所以最后采用了上面这样的只存了type的做法。

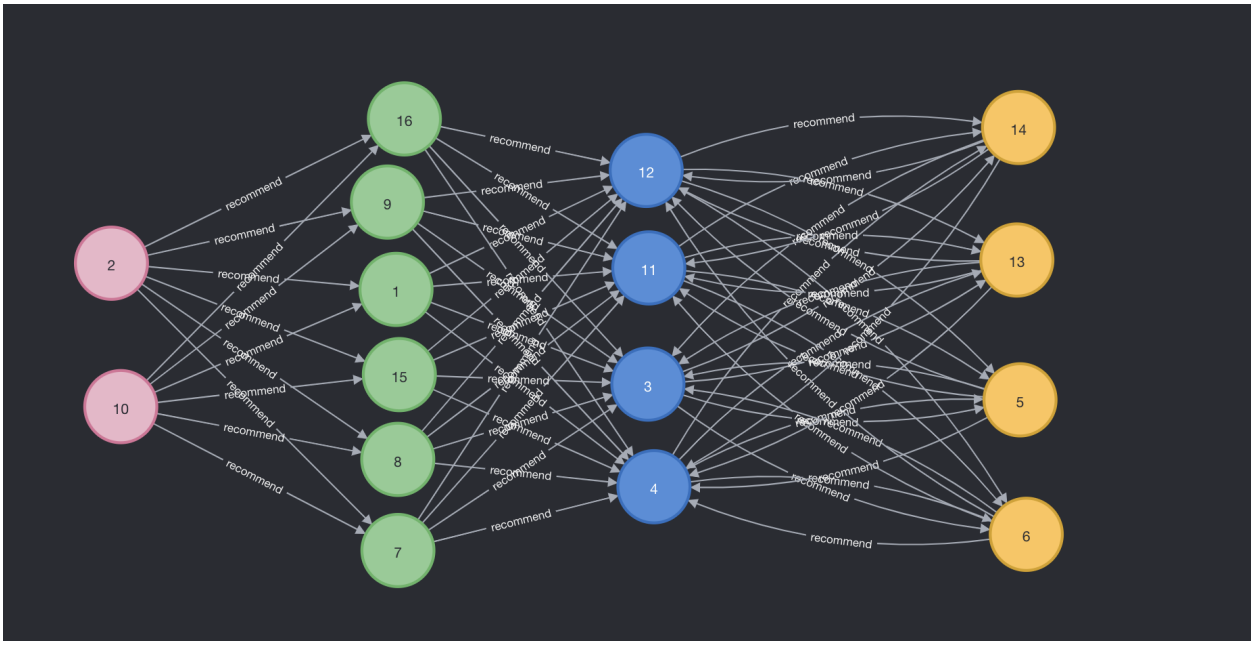
```
create (n:novel {id:1 }) return n;
create (n:children { id:2 }) return n;
create (n:biographies {id:3 }) return n;
create (n:biographies {id:4 }) return n;
create (n:autobiographies{id:5 }) return n;
create (n:autobiographies{id:6 }) return n;
create (n:novel {id:7 }) return n;
create (n:novel {id:8 }) return n;
create (n:novel {id:9 }) return n;
create (n:children { id:10 }) return n;
create (n:biographies { id:11 }) return n;
create (n:biographies { id:12 }) return n;
create (n:autobiographies{id:13 }) return n;
create (n:autobiographies{id:14 }) return n;
create (n:novel {id:15 }) return n;
create (n:novel {id:16 }) return n;
```

```
MATCH (a:children),(b:novel)
CREATE (a)-[r:recommend]->(b)
RETURN r;

MATCH (a:novel),(b:biographies)
CREATE (a)-[r:recommend]->(b)
RETURN r;

MATCH (a:autobiographies),(b:biographies)
CREATE (a)-[r:recommend]->(b)
RETURN r;

MATCH (a:biographies),(b:autobiographies)
CREATE (a)-[r:recommend]->(b)
RETURN r;
```



不同颜色的点代表不同类别的书。他们之间有相互推荐的关系。粉色：children 绿色：novel 蓝色：biographies 黄色：autobiographies

reference

How to Uninstall & Remove Homebrew Packages

How to Uninstall Packages with Homebrew

If you have installed Homebrew on a Mac to use as a package manager for various unix and command line utilities, you've probably also installed a handful of packages deemed useful to you. But what if you no longer need one, and you want to remove a particular Homebrew package?

https://osxdaily.com/2018/07/29/uninstall-packages-homebrew-mac/

mongodb主要用来干嘛，什么时候用，存什么样的数据？

一年多一直在使用mongodb，基本上从对mongodb不懂的小白，到现在操作mongodb无任何压力，最近的一个项目在做mongodb副本集的管理，三组机器做一个副本集，大概要管几十台上百台副本集，主要用来自动化发布，构建副本集（我们的mongo副本集部署在docker中），所以在写各种mongo shell的代码实

简 https://www.jianshu.com/p/50af2537f190

使用java连接Mongodb时报错：Command failed with error 18 (AuthenticationFailed): 'Authentication failed.'

报错信息： Caused by: com.mongodb.MongoCommandException: Command failed with error 18: 'Authentication failed.' on server XXXXXXX:27017.

http://www.voycn.com/article/shiyongjavalianjiemongodbshibaocuocommand-failed-error-18-authenticationfailed

Mac安装neo4j

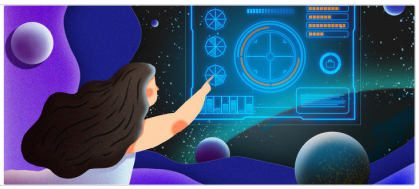
brew install neo4j 2.启动 neo4j start 关闭 neo4j stop neo4j start Active database: graph.db Directories in use: home: /usr/local/Cellar/neo4j/3.3.4/libexec config: /usr/local/Cellar/neo4j/3.3.4/libexec/conf logs: /usr/local/Cellar/neo4j/3.3.4/libexec/logs plugins: /usr/local/Cellar/neo4j/3.3.4/libexec/plugins import:

简 https://www.jianshu.com/p/b8d283ad9d93

最全 Neo4j 可视化图形数据库的工具！

在我们深入研究工具之前，了解现有工具的类别很重要。所有可视化工具包都是根据特定目的构建的，因此您必须确保工具的目的符合您的需要。 我将所有图形可视化工具分为四大类： 该 Neo4j的浏览器 可能与Neo4j的工作时，你会碰到的第一件事情。作为数据库开发人员运行 Cypher 查询的工具，浏览器允许您

知 https://zhuanlan.zhihu.com/p/381044281



neo4j语法

删除所有节点

```
MATCH (n)
OPTIONAL MATCH (n)-[r]-()
DELETE n,r
```

neo4j语法-create

创建节点创建简单节点create (n)创建多个节点create (n),(m)创建一个带标签的节点create (n:Person)创建一个带多个标签的节点create (n:Person:Swedish)创建一个带标签和属性的节点CREATE (n:Person { name: '...

知 https://zhuanlan.zhihu.com/p/354649299