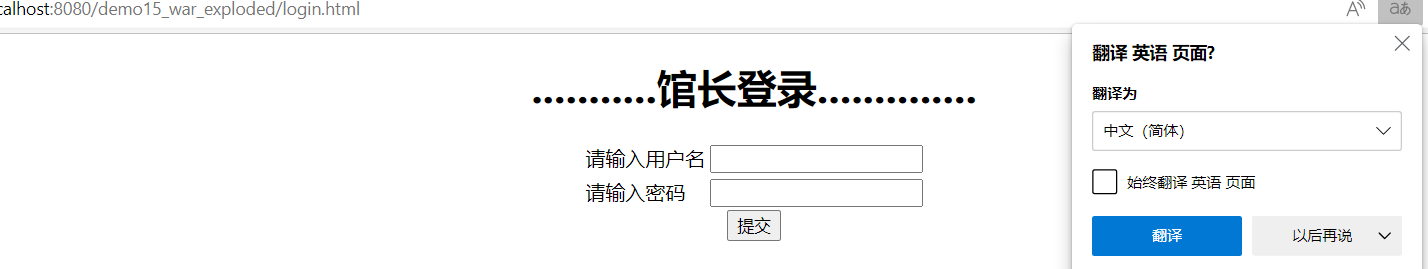
电协网络组考核文档

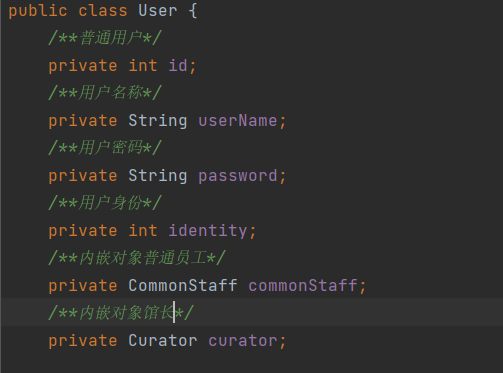
第一部分：项目总览

项目介绍：一个图书管理系统，拥有馆长，普通员工两个角色。馆长和普通员工都拥有的权限是完成新增图书，借出图书，借入图书等等的需求，馆长拥有管理普通员工的权限（拉黑和解封两样需求）其中，这期间也使用了Filter技术完成了对于Servlet层各个接口的拦截等等。

首先演示一下馆长登录部分的接口：



登录的思路：当馆长输入用户名和密码之后，首先会从User表当中查询是否存在此用户，如果存在，那么将关联对应的人员信息：Curator，这是User实体类的设计：

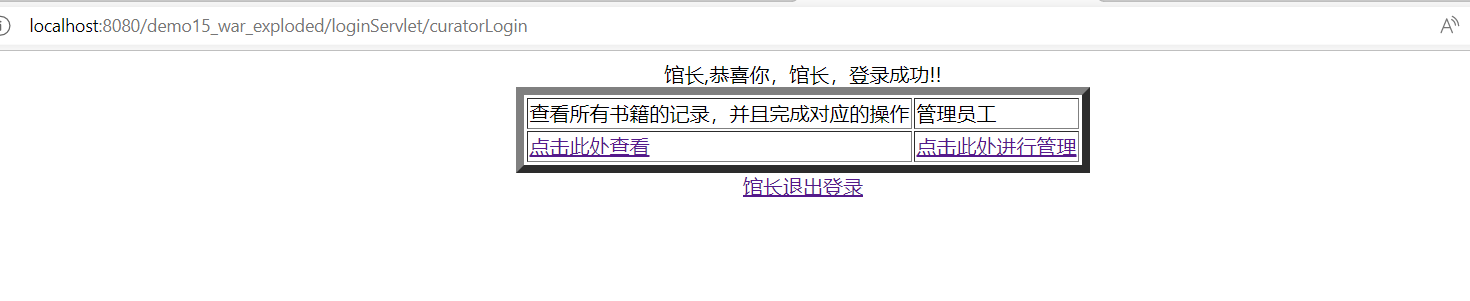


其中，用户的身份(identity)将与对应的实体类信息相匹配，登录成功之后，需要把用户的信息放入Session域当中



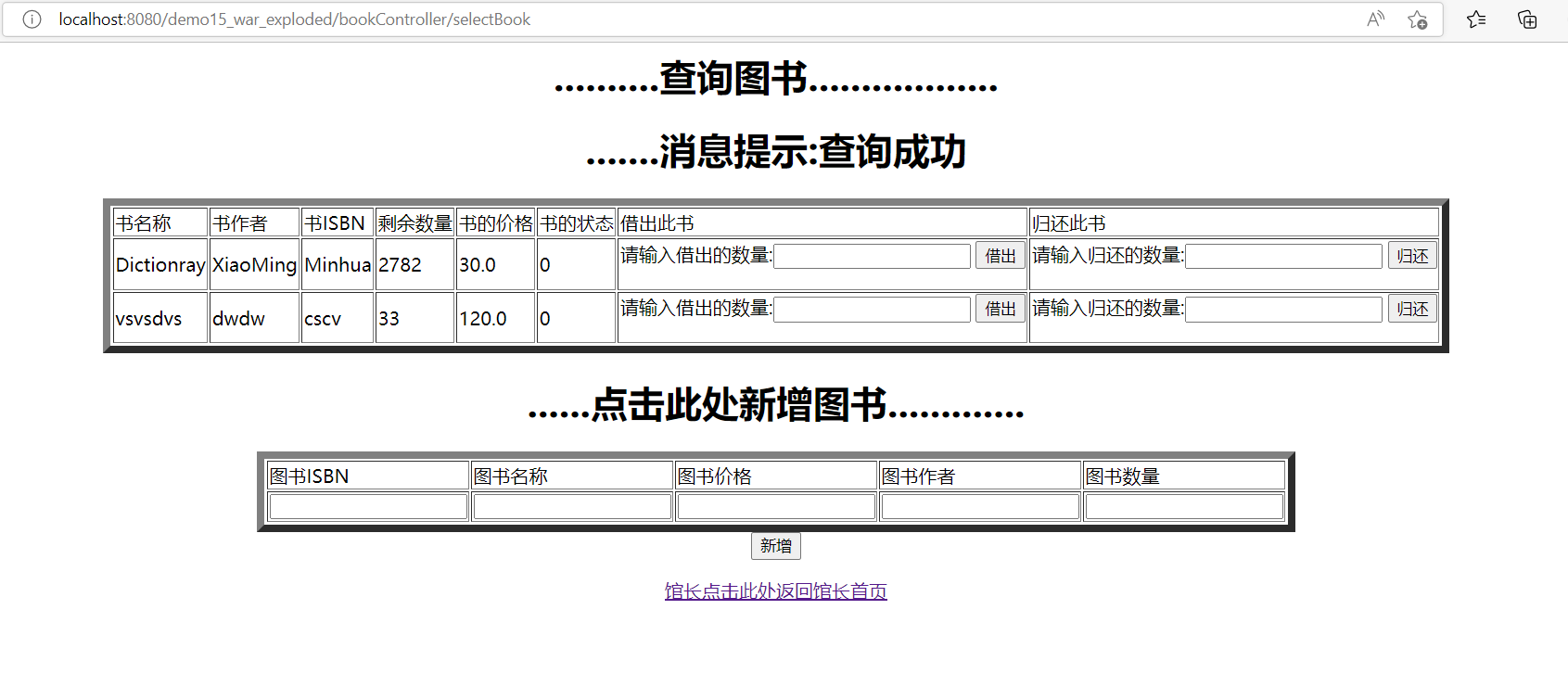
这里，返回前端的结果集当中包含了馆长的用户名等等的信息，如果登录失败，那么将会跳转到登录的页面，无法访问对应的权限。普通员工的进行方式也和馆长一样，没有太大的区别，只是实体类当中嵌套的对象是ConnmoStaff。

下面演示馆长登录之后的操作，登录之后可以进行查阅书籍，管理员工两样权限，还可以选择退出登录。



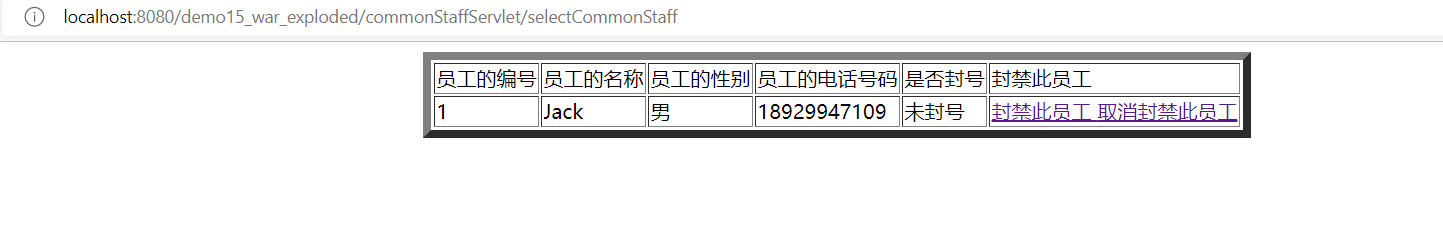
这里我想介绍一下退出登录的操作：由于登录成功之后，把user对象放入了session域当中，那么后面如果登录成功后，想退出登录，就需要把对应的user信息给移除，这样的话，当前用户就没有办法访问响应的接口了（对应的接口已经使用filter保护起来了）

进入管理书籍的页面：



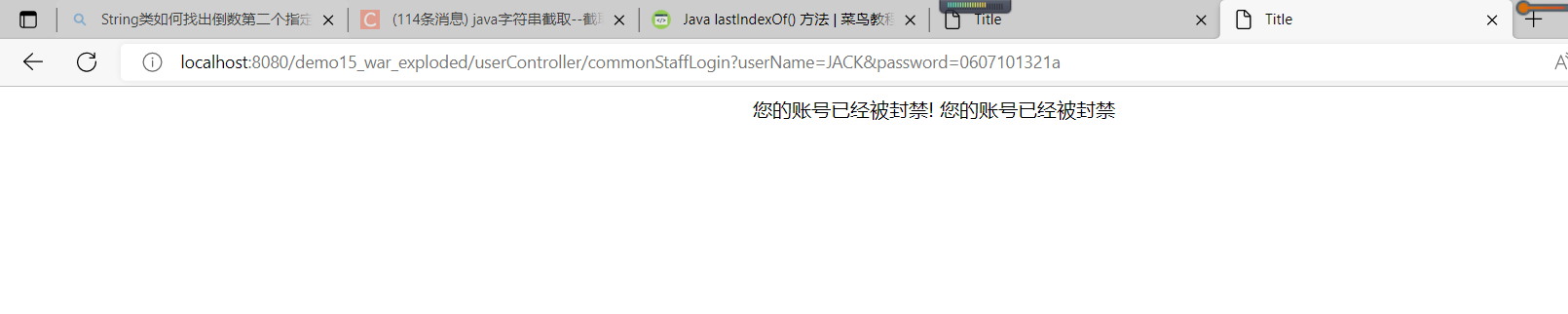
在此处，馆长可以选择借入借出书籍，还可以新增书籍。其中，对一些参数做了一点点简单的校验，比如输入英文等数据，会提示不要输入非法数据

馆长管理员工界面：



如果馆长封禁了对应的员工，那么员工将无法访问接口，并且无法进行各样操作，例如管理书籍等等。

下面的情景，就是员工被封禁之后的情景



项目靓点介绍：

第一点：BaseServlet：需求分析，之前的操作，我们是定义一个操作对应一个Servlet的，并且每一个Servlet都是直接继承于HttpServlet，每个Servlet只能有一个doGet或者doPost方法，那么这样下来，将会需要定义大量的Servlet，为每一个操作需要单独定义一个Servlet，造成代码冗余的同时，也会增加了耦合度（可能一个接口需要处理多个不同的请求等等）那么这样，就引入了BaseServlet，请求转发器，用于统一接受前端的请求，并且完成请求转发。其他的一切子类都将继承于当前的BaseServlet。

同时，还有一个默认的规定，那就是把针对同一个对象的增删改查的方法集中放到一个.java文件当中，增加了代码的可维护性等等。

实现思路：需要一个BaseServlet继承于HttpServlet，并且重写其中的service方法，原因是此方法可以实现doGet和doPost同时接受请求。接着，需要通过req.getRequestURI方法，获取到请求的路径，此时通过字符串截取的方法，截取到最后一个”/”之后的字符串，那个是希望执行的方法的名称

//1，获取请求路径，整个的请求路径

String uri = req.getRequestURI();

//2,获取最后一段路径，截取方法名称

int index = uri.lastIndexOf('/');

//从斜杠往后的都是方法名称

String methodName = uri.substring(index + 1);

这个时候，如果想要执行对应的方法，那么就需要通过反射，来执行。

反射的步骤，通过类对象.getClass()方法获取到对应类的二进制字节码对象。

Class<? extends BaseServlet> cls = this.getClass();

引入问题：为什么此this为被继承类的对象？

This的含义：代表当前对象的引用，即：哪个个对象调用了当前的service方法，那个对象就是this。由于提交的目标类为子类，并且子类继承于BaseServlet，也就相当于子类调用了此service方法。之后，就需要通过反射执行方法：将引入以java.reflect包下面的Method方法。需要通过反射获取方法

Method method = cls.getMethod(methodName, HttpServletRequest.class, HttpServletResponse.class);

这里对于方法进行了一个统一：对应方法仅仅有HttpServletRequest,HttpServlet Response两个接口提供。

方法执行：method.invoke

Invoke：含义执行对应的方法，谁来执行？this来执行，参数有哪些？request和response

method.invoke(this, req, resp);

项目靓点二：

BaseDao和结果集映射：

思路，所有的Dao实现类都要继承于BaseDao，来完成对应的操作；

在Dao层当中，编写完成sql语句之后,通过this.BaseDao当中的方法来调用对应的方法。

例子：

@Override

public void borrowOut(String bookId, String count) {

String sql = "update book set count=count-? where bookId=?";

this.help(sql, count, bookId);

}

结果集映射的思路：需要传入对应的实体类的二进制字节码对象，即.class对象，接着，需要参数列表来拼接生成对应的sql语句，并且产生一个结果集ResultSet对象，并且通过反射来完成对应的操作，比如把结果集查询出来的参数和实体类一一对应赋值，最后返回对象或者返回对象的集合。

代码实现分析：

第一步，创建泛型方法<T>

public <T> ArrayList<T> queryList(String sql, Class<T> clazz, Object... args)

第二步：创建ArrayList<T>集合，并且获取连接，并且遍历args数组

//创建泛型集合

ArrayList<T> list = new ArrayList<>();

ResultSet resultSet = getResultSet(sql, args);

//遍历数组，并且产生对应的结果集

private ResultSet getResultSet(String sql, Object... args) {

Connection connection;

PreparedStatement preparedStatement;

try {

int i = 1;

connection = JavaDataBaseConnectUtil.getConnection();

preparedStatement = connection.prepareStatement(sql);

//遍历数组

if (args.length > 0) {

for (Object object : args) {

preparedStatement.setObject(i, object);

i++;

}

}

return preparedStatement.executeQuery();

} catch (SQLException e) {

logger.warning("executeUpdate方法预编译sql语句异常：" + e);

throw new DaoException("Dao层发生了sql语句异常", e);

}

}

第三步，通过生成的结果集，调用resultSet.getMetaData();方法，来获取到一个名称叫做ResultSetMetaData的接口对象，通过此对象，来获取产生结果集对象的列数，列名称等等的信息

ResultSetMetaData resultSetMetaData = resultSet.getMetaData();

//获取列数

int colum = resultSetMetaData.getColumnCount();

//遍历结果集

//获取列数

int colum = resultSetMetaData.getColumnCount();

//遍历结果集，在执行第一次的遍历之前，指针没有指到数据部分，当执行一次.next之后，就会指到第一行当中。

while (resultSet.next()) {

//通过字节码.getDeclaredConstructor()来获取构造器，并且通过newInstance()方法获取到一个对象

T t = clazz.getDeclaredConstructor().newInstance();

//为当前的对象属性赋值

workForField(colum, t, resultSetMetaData, clazz, resultSet);

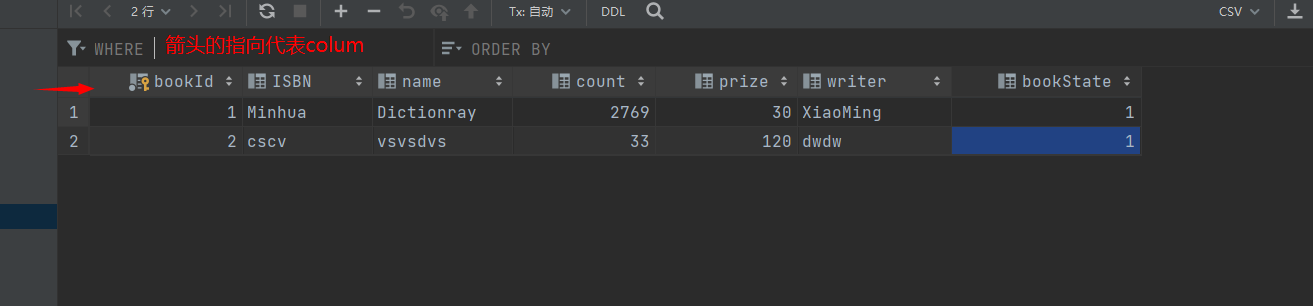
//赋值完成之后添加到对应的集合当中

list.add(t);

}

当执行到next的时候，箭头往下移动了一位，指向第一列，每一列代表每一个对应的对象，下面，就是为属性赋值的方法详解(workToField)方法

整体思路：根据传递过来的参数colum来确定一共有多少列，



//第一步：获取列名称

String columName = resultSetMetaData.getColumnLabel(i + 1);

获取到的值为对应竖着的红色箭头的指向的名称

//获取列的属性columnValue值为列名，数据库的列名称，columnValue为数据库表中的数据

Object columnValue = resultSet.getObject(columName);

由于此时指针指向了colum的第一列，那么通过此列的名称，来获取到对应的值

// 第三步，需要通过反射，查询到对应的类属性

//如果没有此属性，会报异常：noSuchFieldException

Field field = clazz.getDeclaredField(columName);

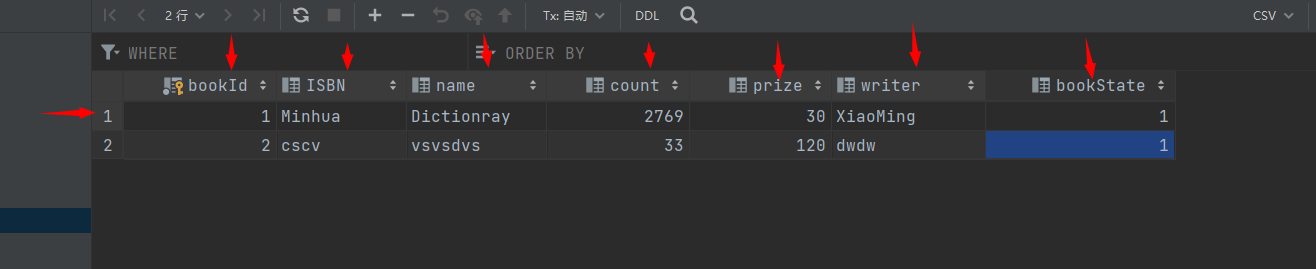
为当前属性赋值:

//无视属性修饰符

field.setAccessible(true);

//设置属性值,t代表对应的对象,columValue代表对应的值

field.set(t, columnValue);



完成一次赋值之后，把对应的对象添加到集合当中，继续执行resultSet.next方法，横着的箭头指针继续下移。

项目规范方面：

第一个：把所有的返回前端的结果集进行了统一的封装

/\*\*此数据为返回前端的数据data，可能是一个实体类，也有可能是一个list,map\*/

private Object data;

/\*\*状态码\*/

private Integer code;

/\*\*简要的消息提示\*/

private String message;

这么做的意义：在项目的实际开发当中，前后端的互交一般都是后端把需要的数据和一些状态码等等的信息，以及简要的消息提示返回给前端，如果按照传统的做法，那么仅仅返回一个list或者map到前端，那就要前端的人员进行数据的解析之后，才给用户提示，这样的操作比较冗余，因而在项目的开发当中，为了减少前端人员的工作量，那么就需要把此三种信息封装成一个结果集。

第二个：代码格式化（缩进明确，写完代码之后ctrl+alt+l）

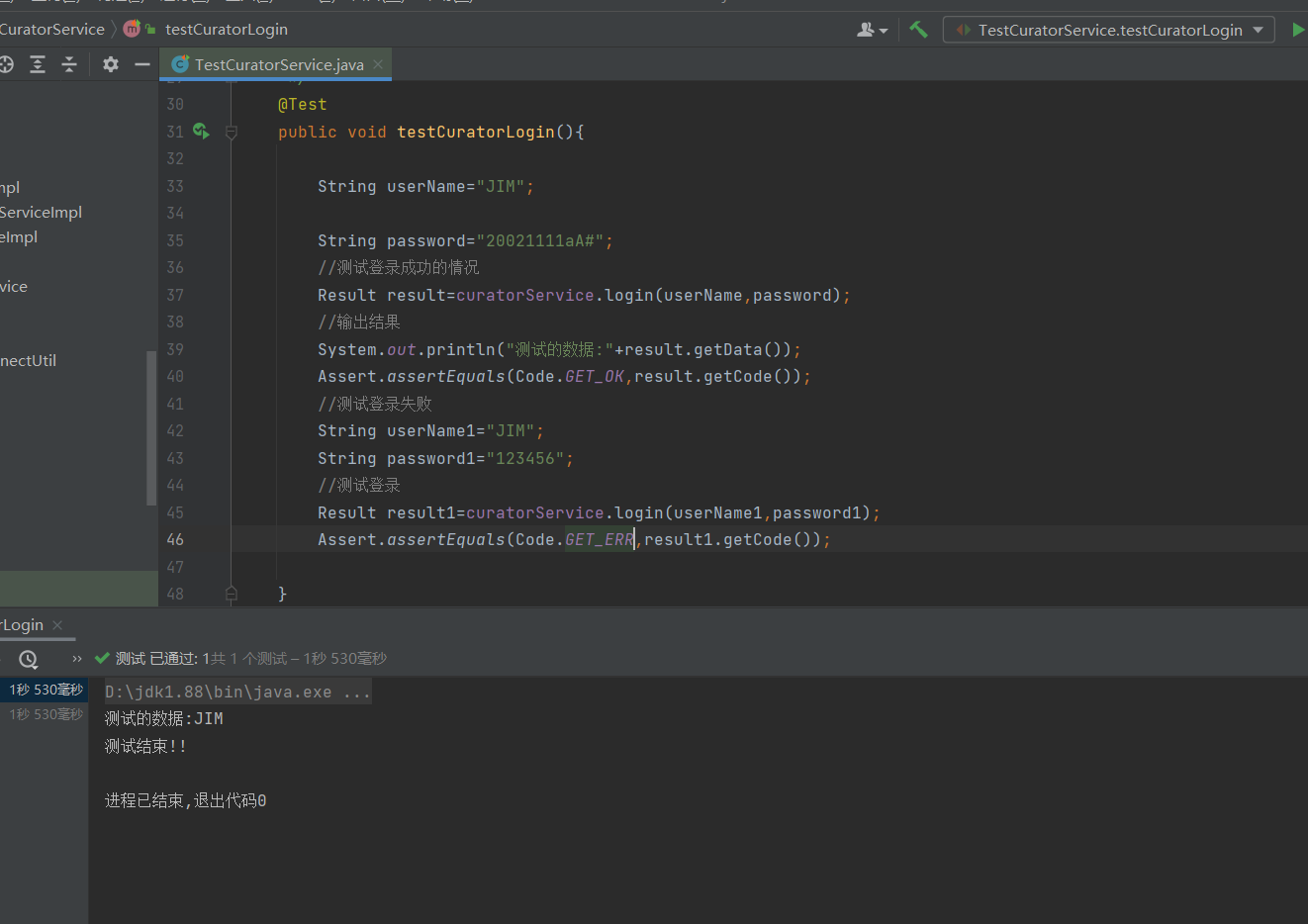
第三个: 每一个service对象和dao对象都有对应的接口以及实现类，接口都有完整的注释记录，实现类以及对应的地方都有代码注释等等的信息。

第四个：使用日志记录器在对应的可能发生异常的地方进行了日志记录，并且把记录的信息输出到对应的文档当中。

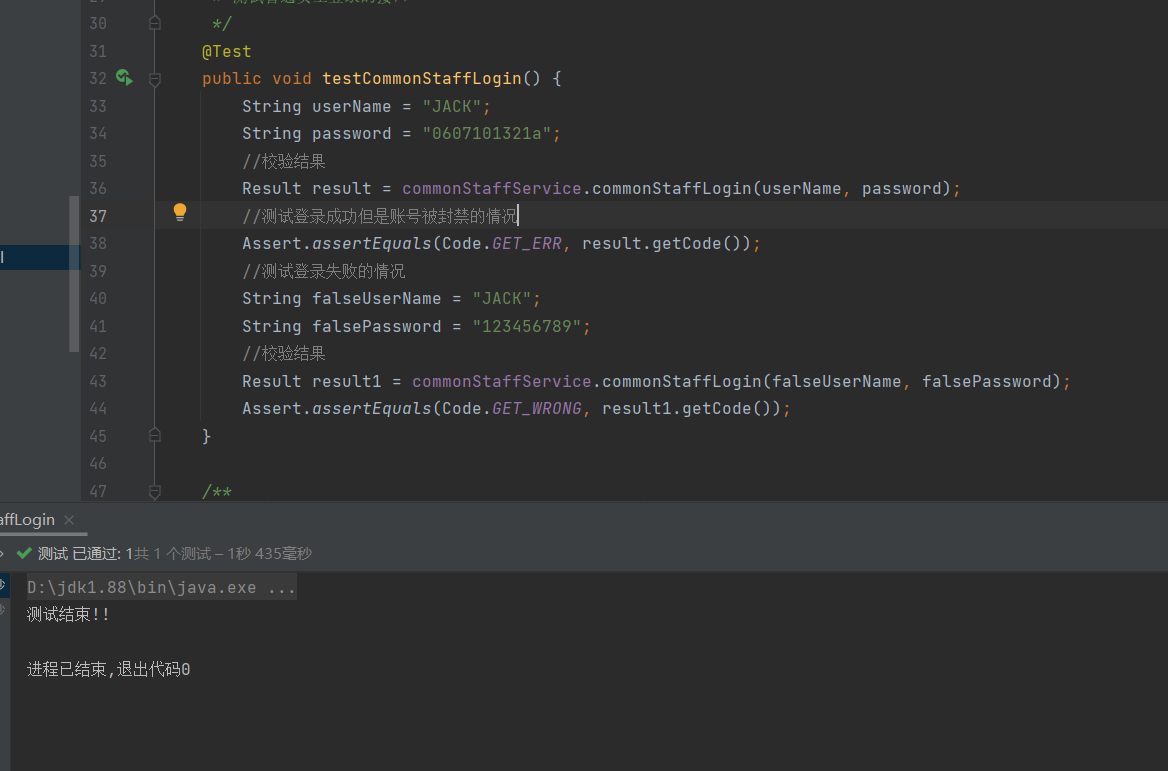
测试数据

基于Junit的测试：

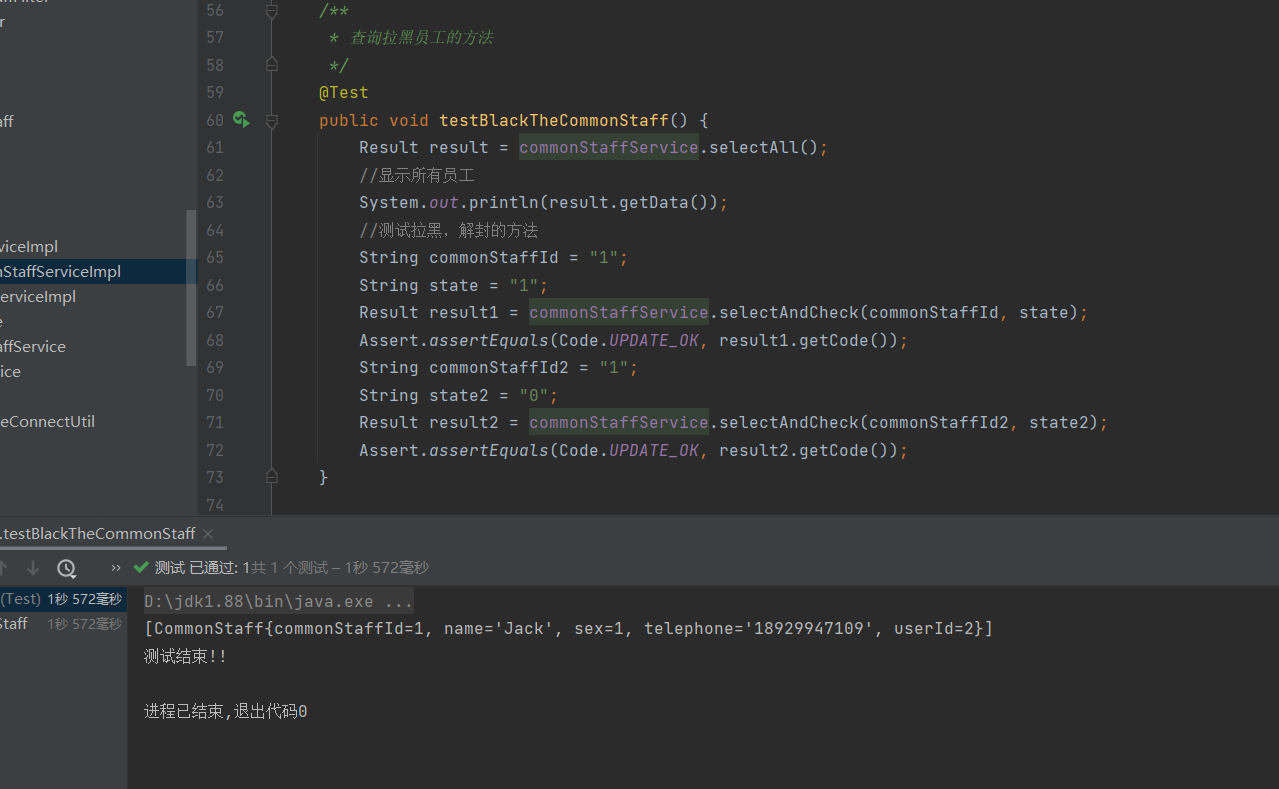
1. 测试馆长登录的Service方法：



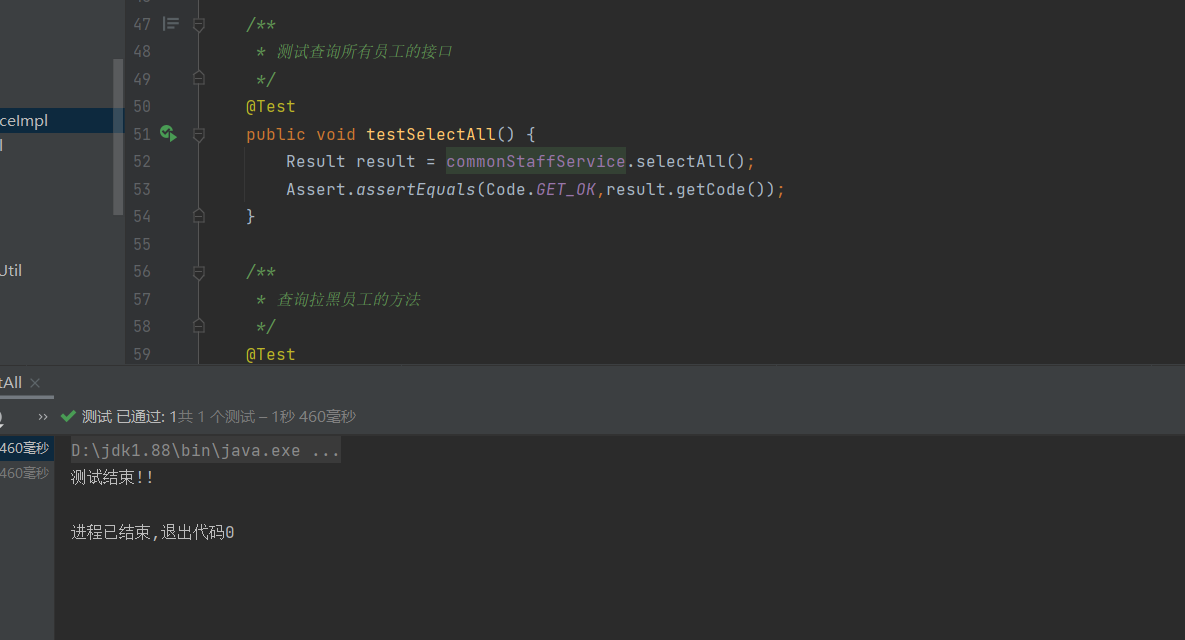
测试普通员工登录的Service方法



测试馆长管理员工的方法

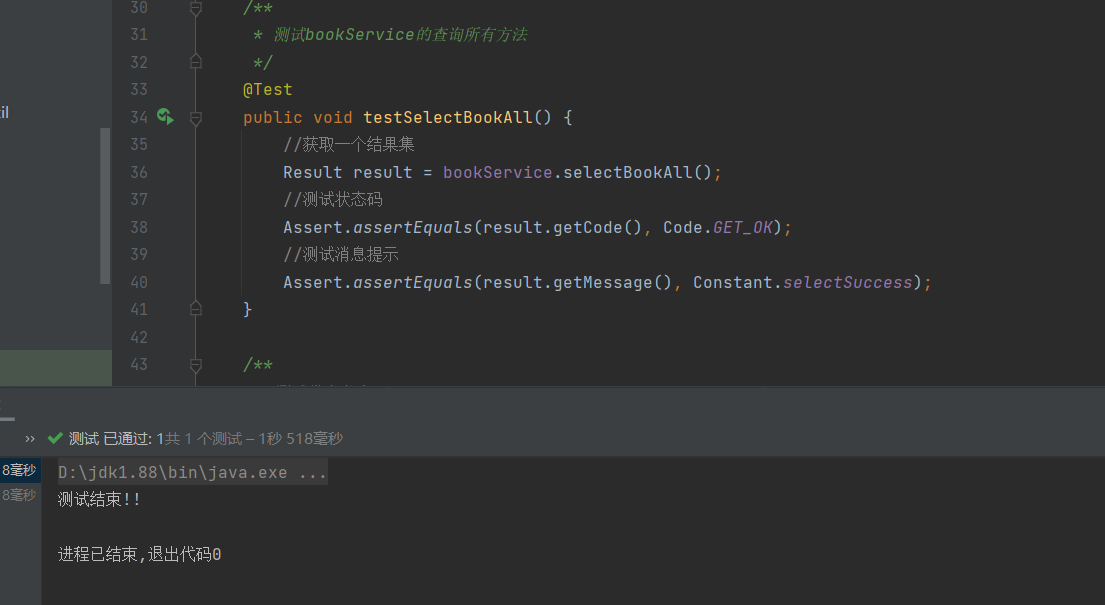


测试馆长查询所有员工的方法

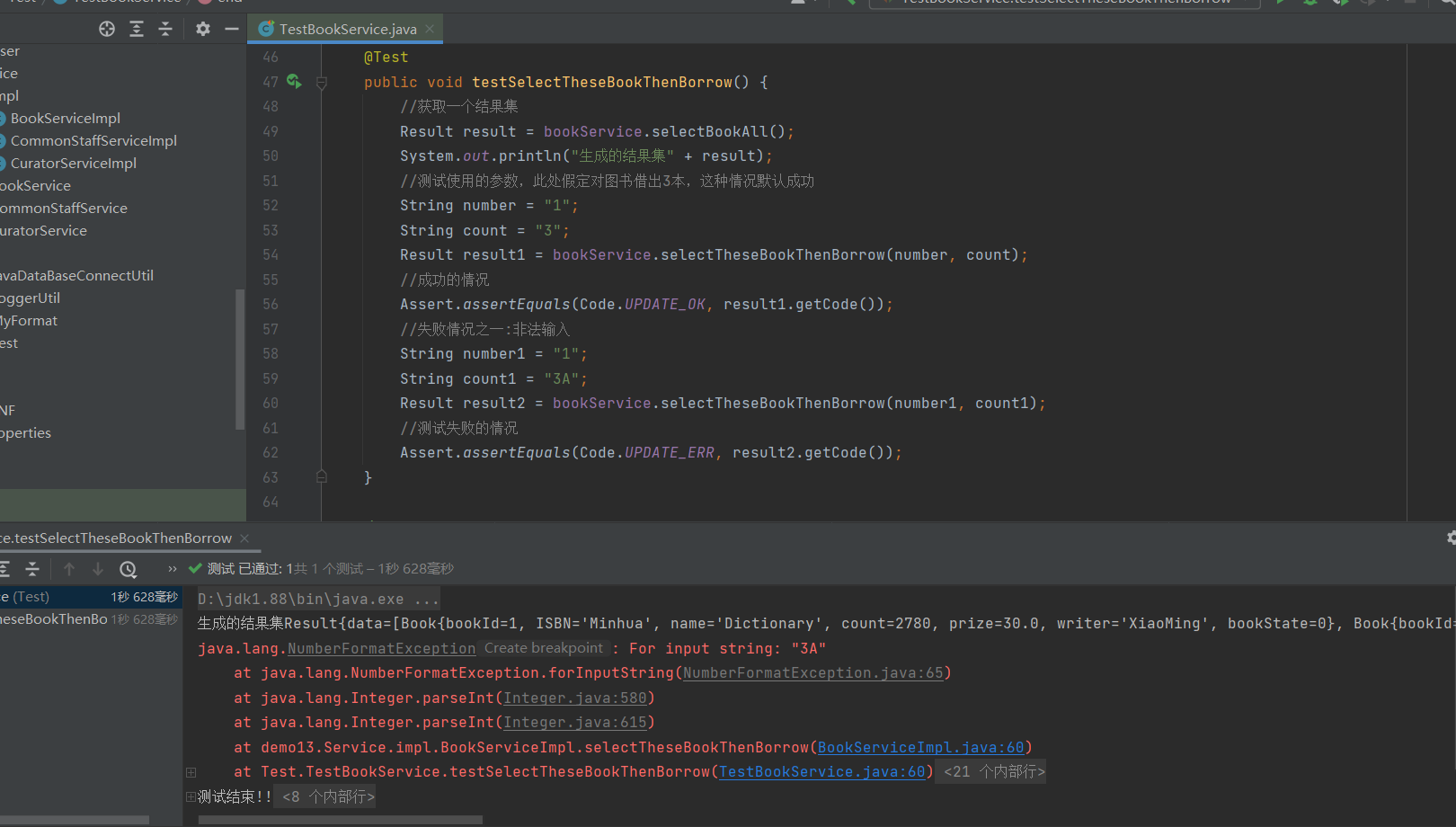


测试管理书籍的方法

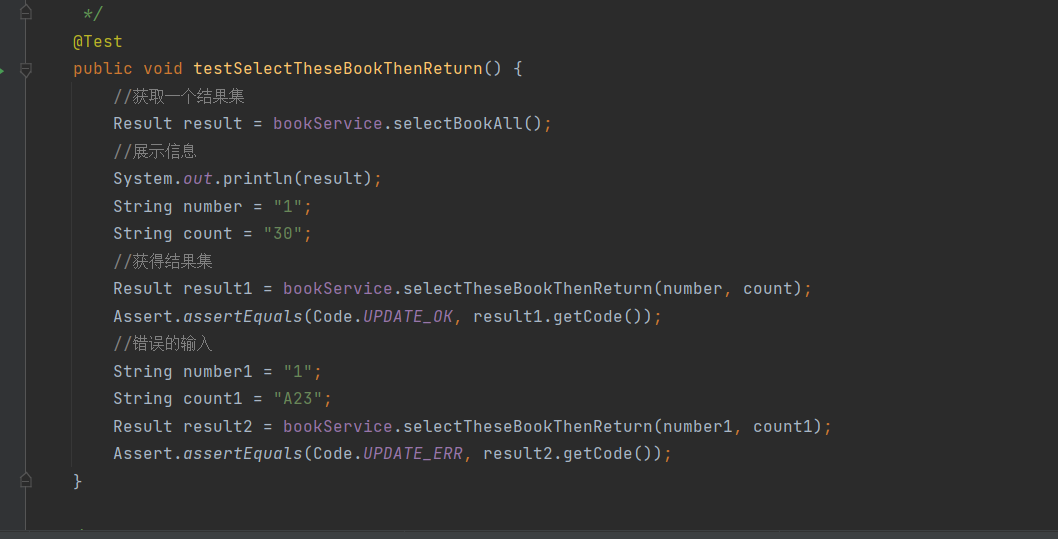
测试查询图书：



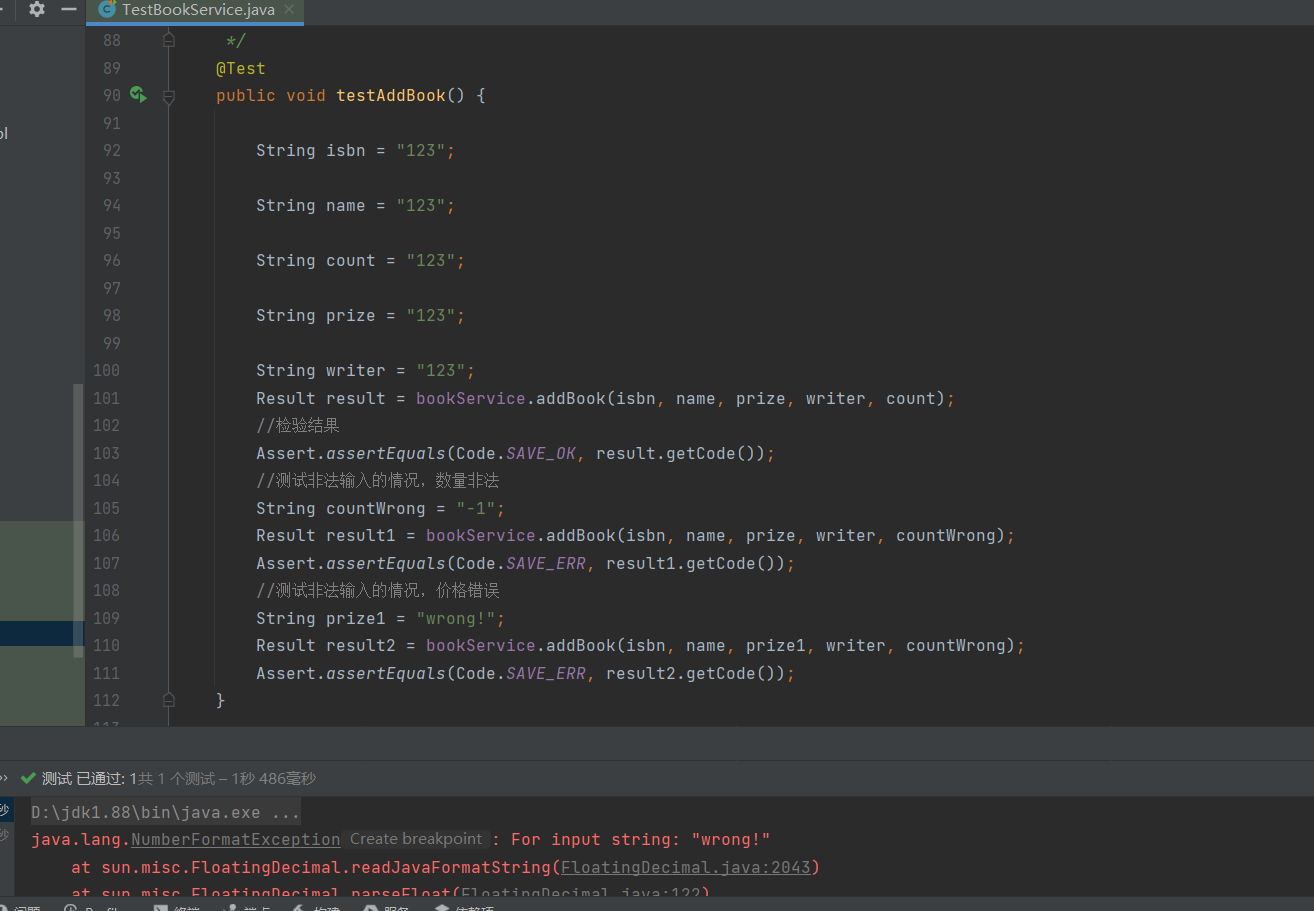
测试借出图书：

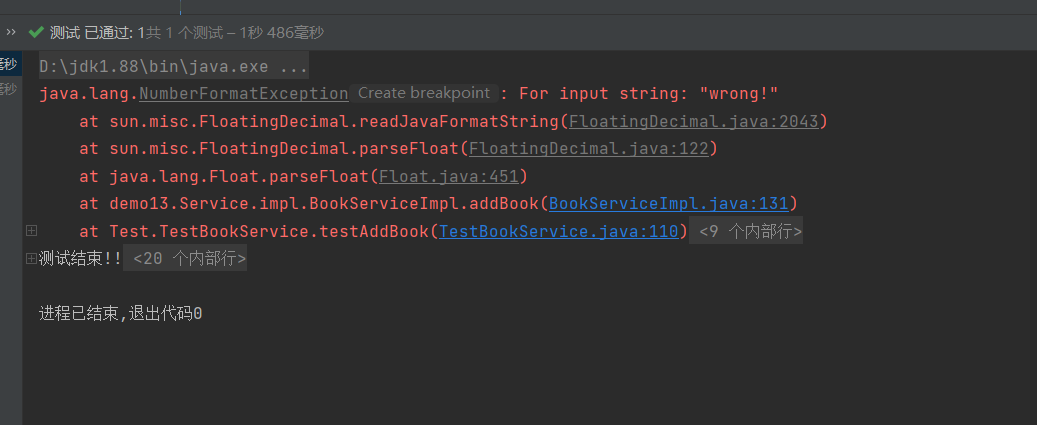


测试归还图书

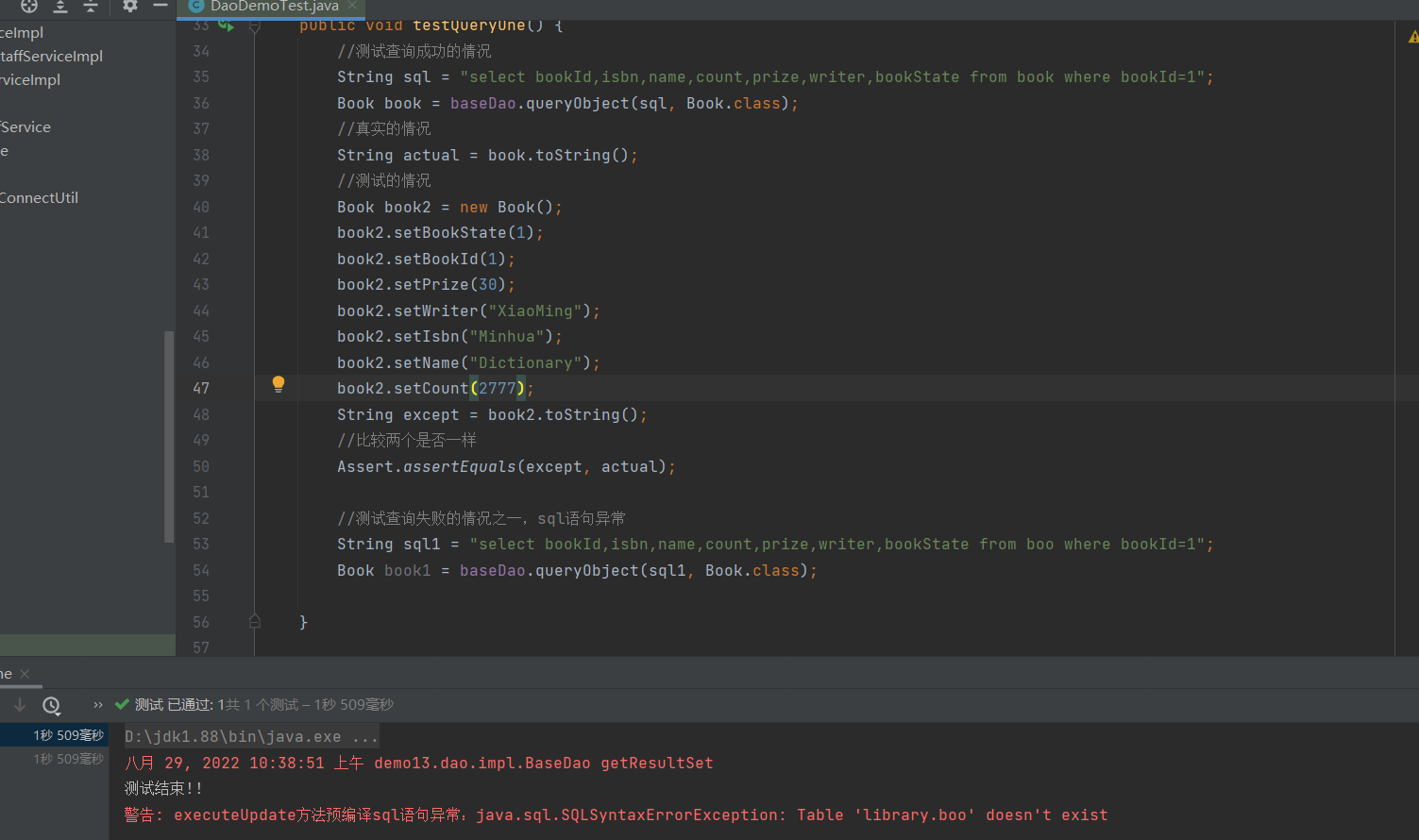


测试新增图书

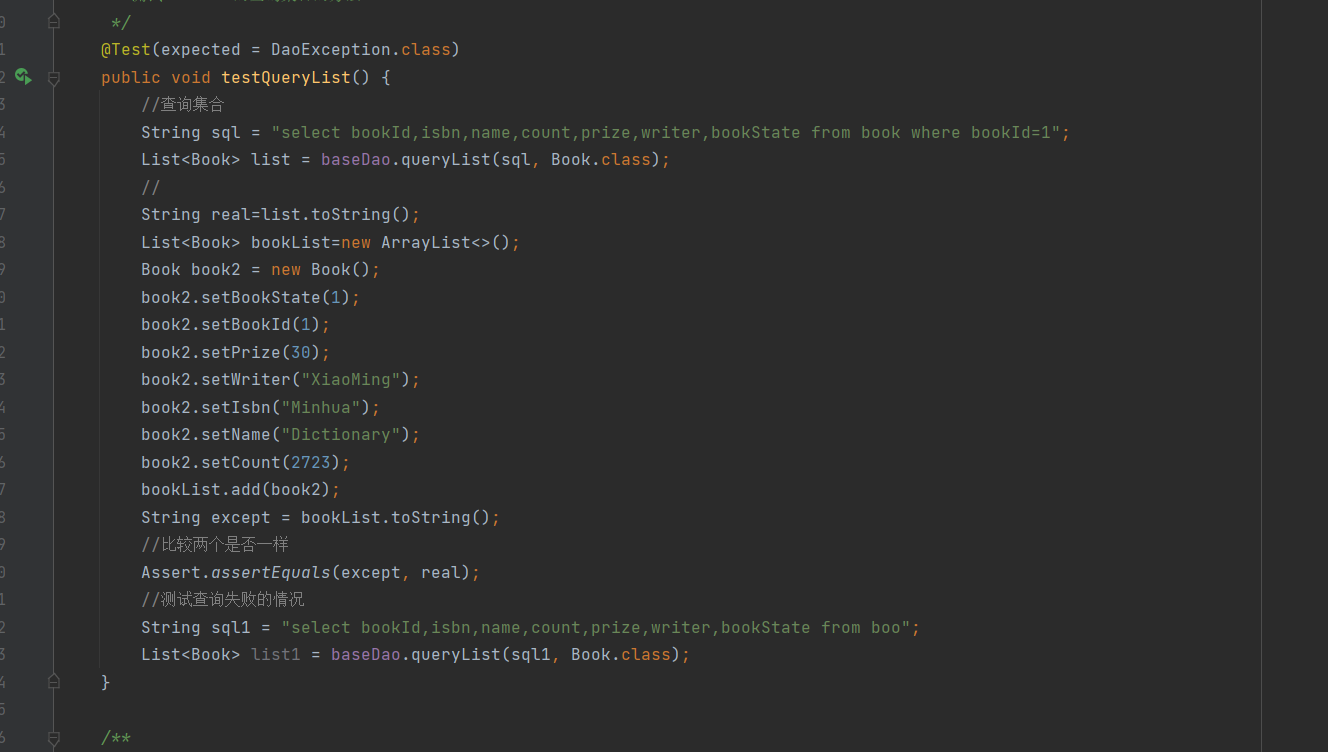




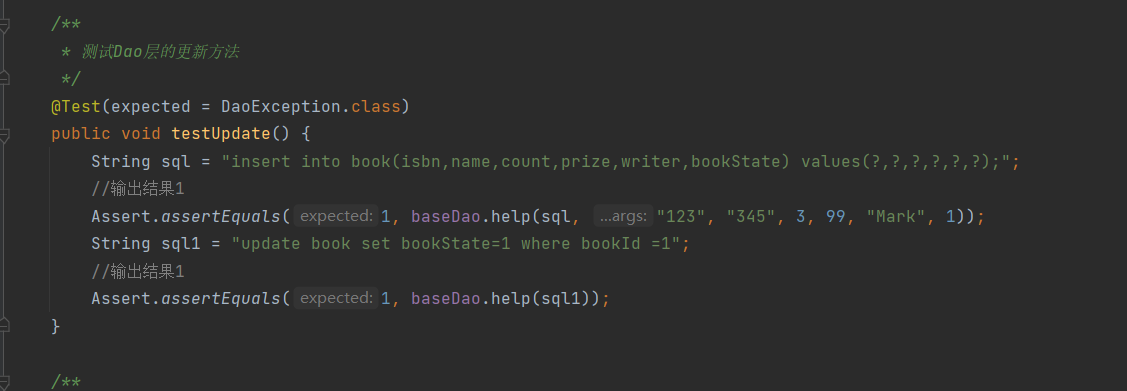
测试BaseDao的增删改查方法：

测试查询单个对象的方法

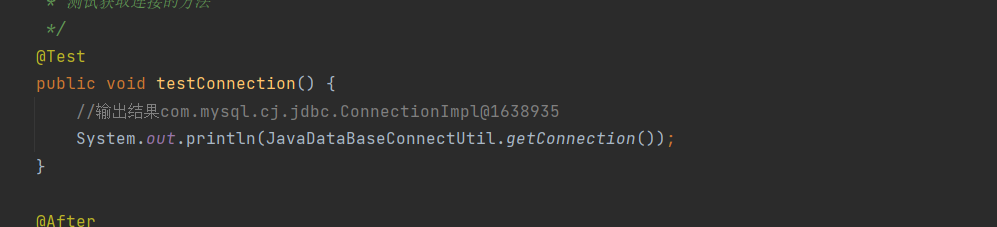
测试查询集合的方法



测试更新的方法



测试获取连接的方法



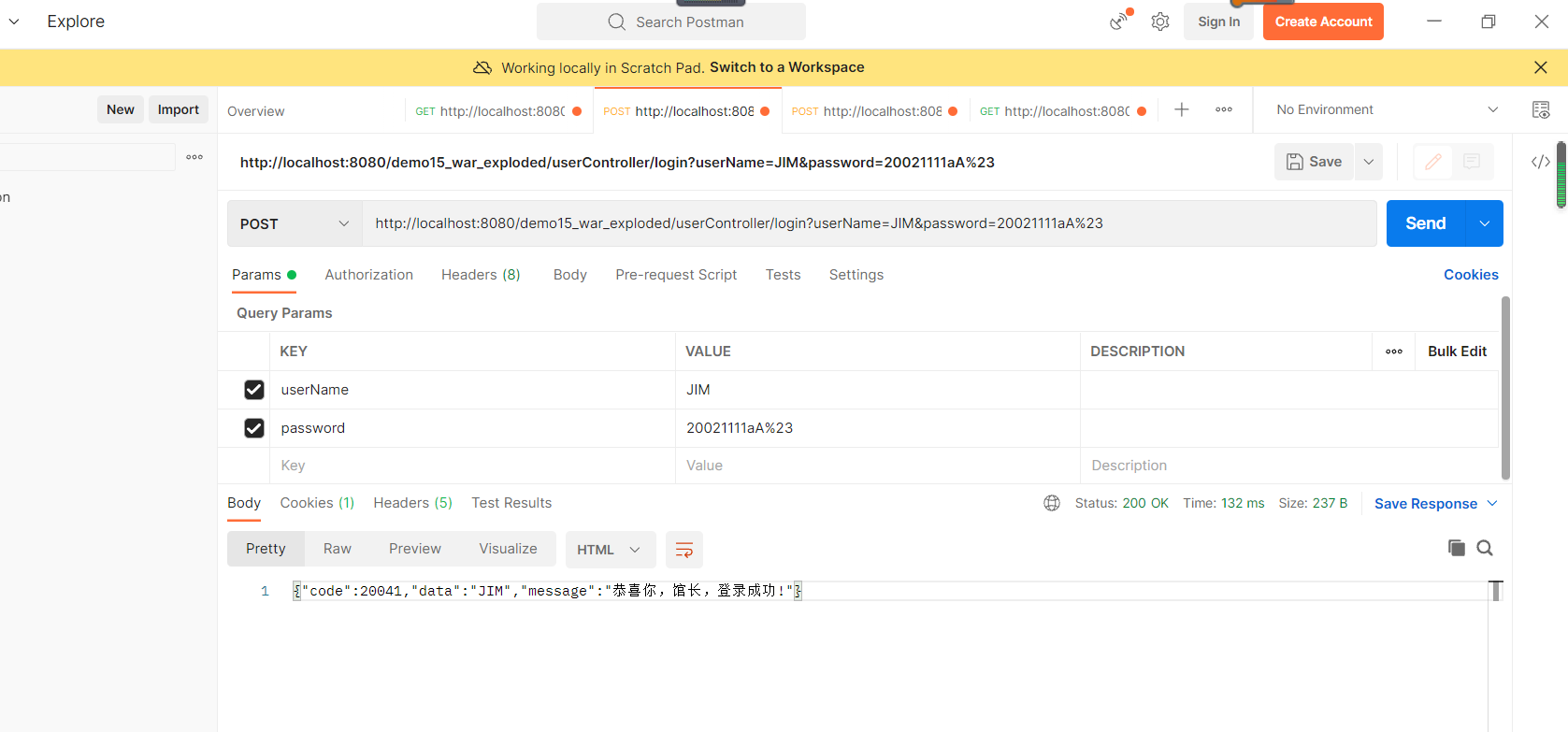
测试接口数据：

测试接口一：userController/login（测试馆长登录的接口）

测试数据：username:JIM

password:20021111aA#

测试馆长登录的接口：



测试结果：

{"code":20041,"data":"JIM","message":"恭喜你，馆长，登录成功!"}

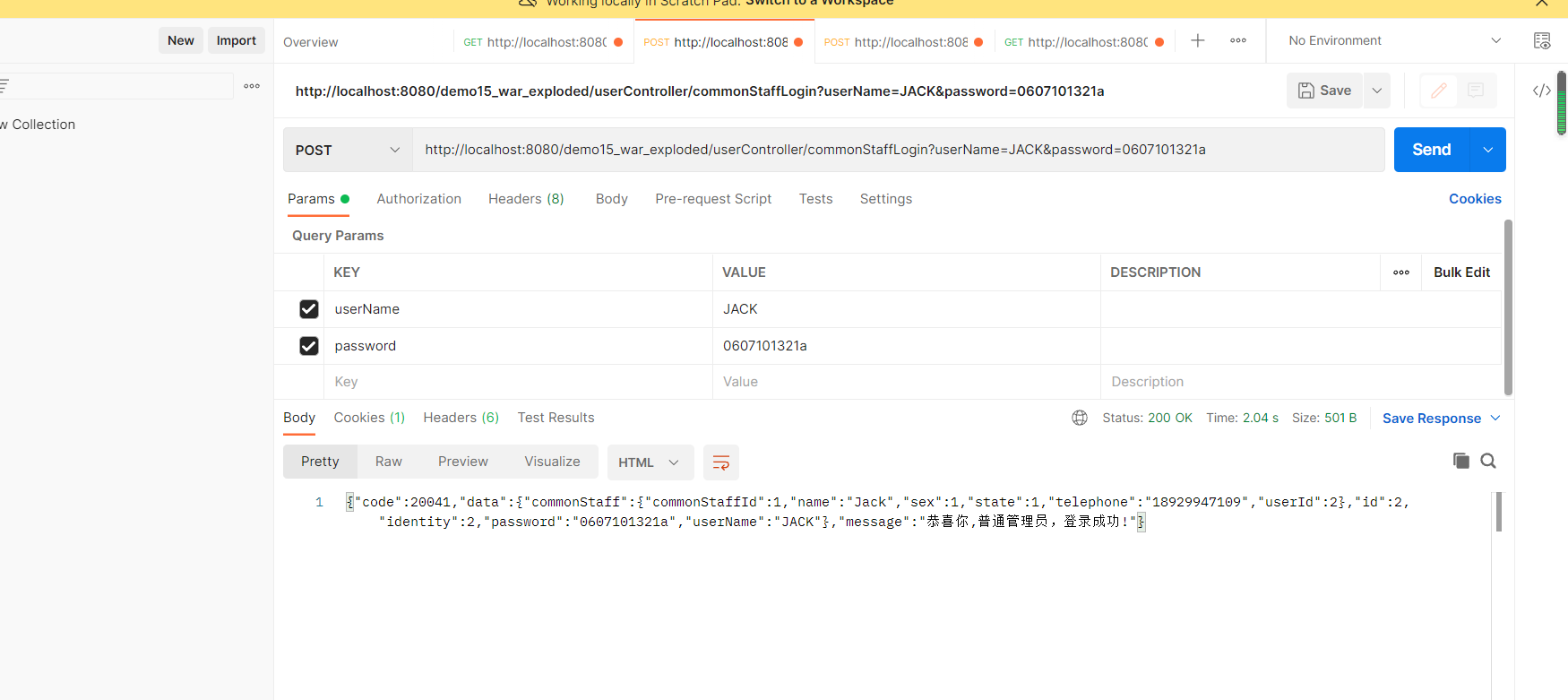
测试普通员工登录的接口：

测试路径：

<http://localhost:8080/demo15_war_exploded/userController/commonStaffLogin>

测试数据：userName:JACK

password:0607101321a



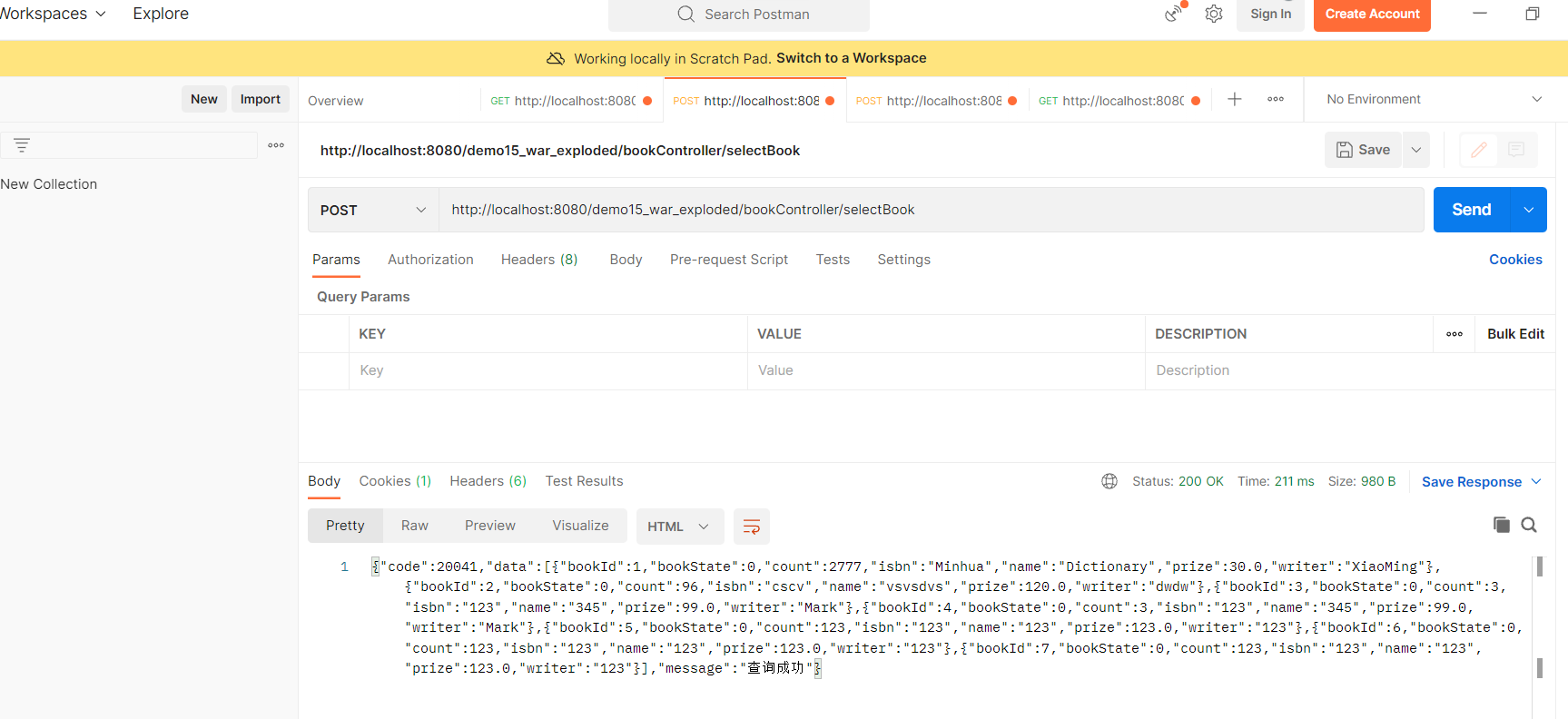
测试结果：

{"code":20041,"data":{"commonStaff":{"commonStaffId":1,"name":"Jack","sex":1,"state":1,"telephone":"18929947109","userId":2},"id":2,"identity":2,"password":"0607101321a","userName":"JACK"},"message":"恭喜你,普通管理员，登录成功!"}

测试接口三：查询图书接口(需要馆长或者员工登录之后才可以访问)

接口路径：<http://localhost:8080/demo15_war_exploded/bookController/selectBook>

测试参数：无

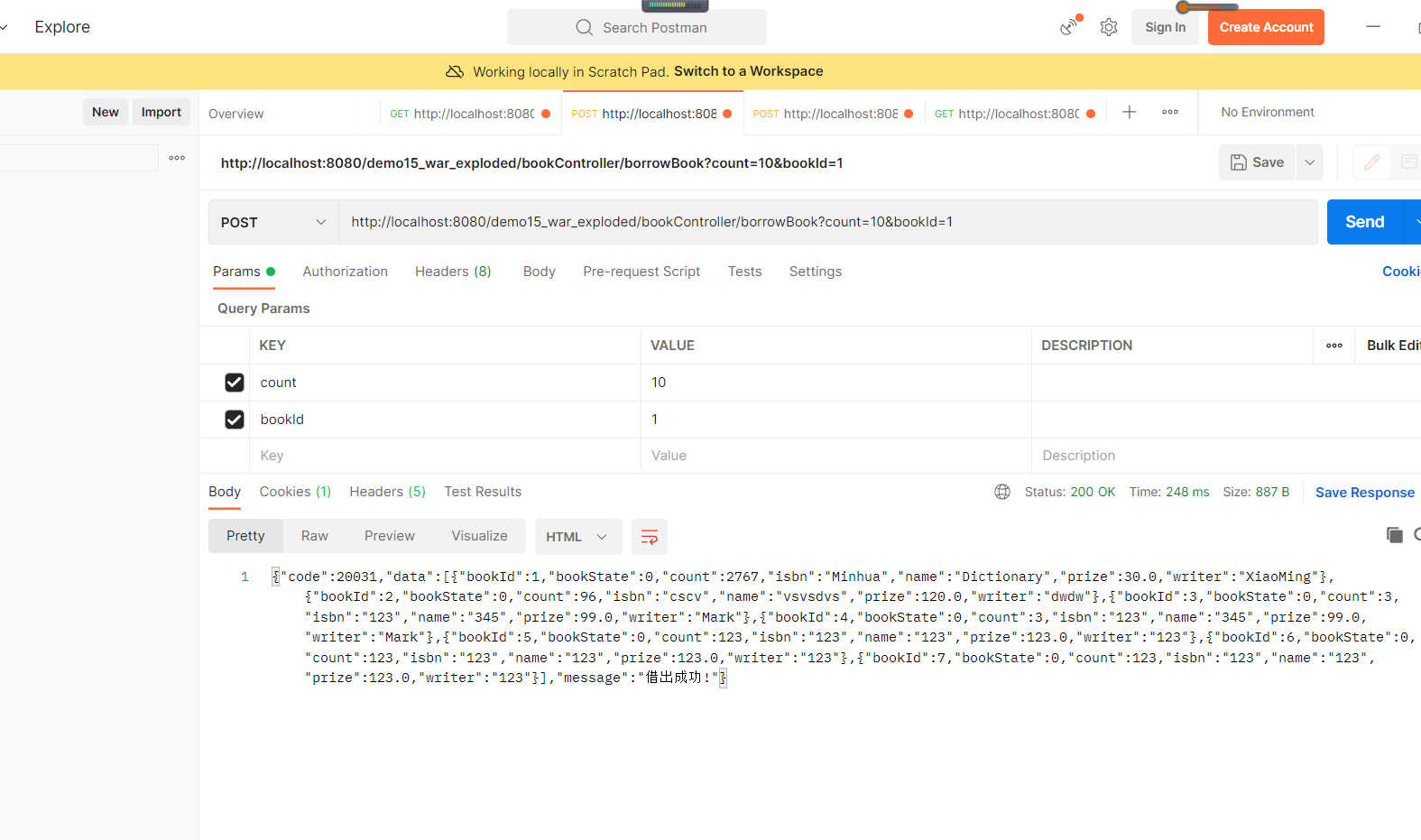


测试借出图书：

测试参数：count :10

bookId:1

测试接口：http://localhost:8080/demo15\_war\_exploded/bookController/borrowBook



测试归还图书：

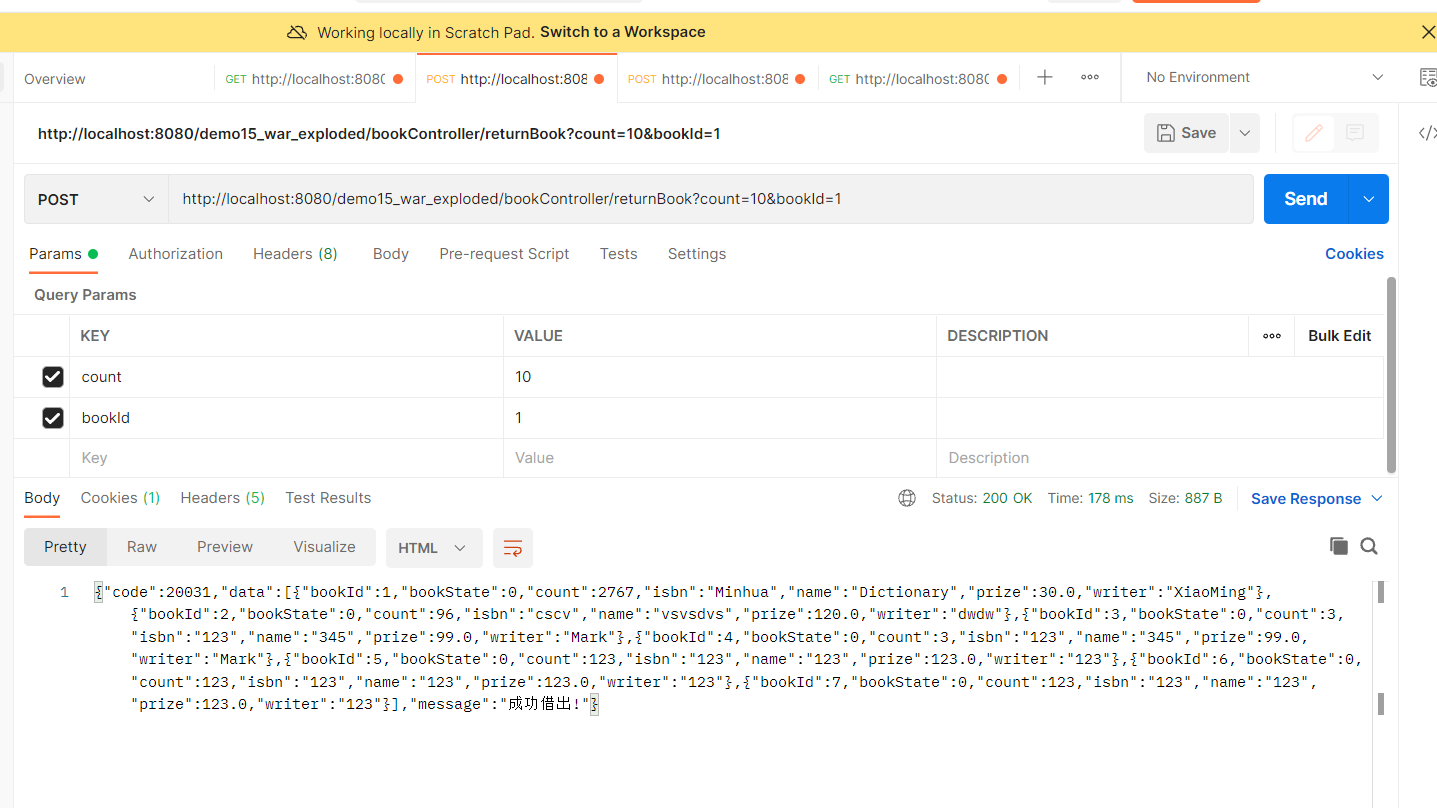
测试接口：<http://localhost:8080/demo15_war_exploded/bookController/returnBook>

参数：count:10

bookId:1

测试数据：

{"code":20031,"data":[{"bookId":1,"bookState":0,"count":2767,"isbn":"Minhua","name":"Dictionary","prize":30.0,"writer":"XiaoMing"},{"bookId":2,"bookState":0,"count":96,"isbn":"cscv","name":"vsvsdvs","prize":120.0,"writer":"dwdw"},{"bookId":3,"bookState":0,"count":3,"isbn":"123","name":"345","prize":99.0,"writer":"Mark"},{"bookId":4,"bookState":0,"count":3,"isbn":"123","name":"345","prize":99.0,"writer":"Mark"},{"bookId":5,"bookState":0,"count":123,"isbn":"123","name":"123","prize":123.0,"writer":"123"},{"bookId":6,"bookState":0,"count":123,"isbn":"123","name":"123","prize":123.0,"writer":"123"},{"bookId":7,"bookState":0,"count":123,"isbn":"123","name":"123","prize":123.0,"writer":"123"}],"message":"成功借出!"}

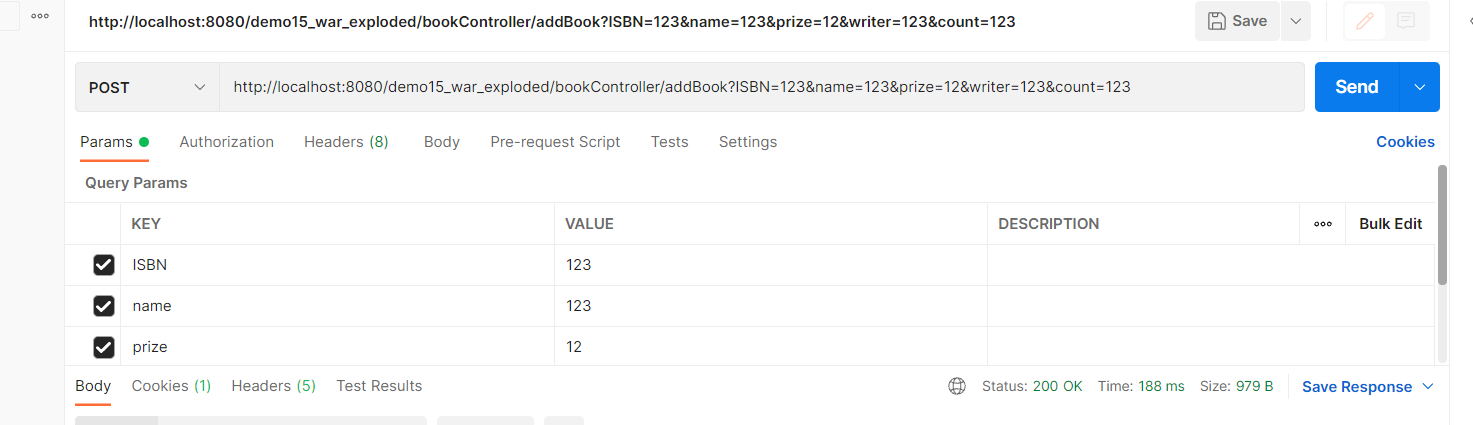


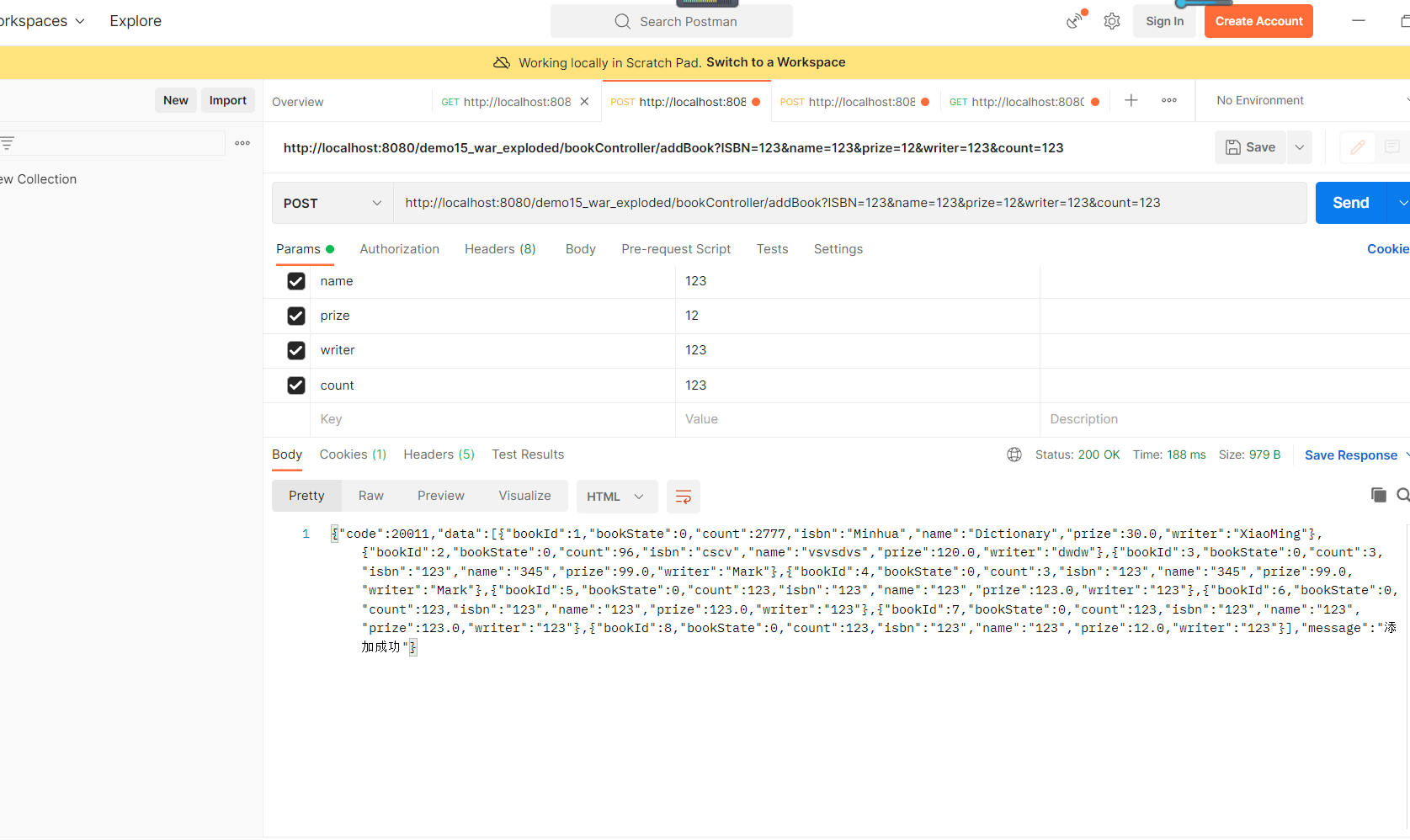
测试新增图书：

测试的接口：



测试的参数：





测试管理员工的接口：

测试路径：<http://localhost:8080/demo15_war_exploded/commonStaffServlet/selectCommonStaff>

测试结果：

{"code":20041,"data":[{"commonStaffId":1,"name":"Jack","sex":1,"state":1,"telephone":"18929947109","userId":2}],"message":"查询成功!"}

