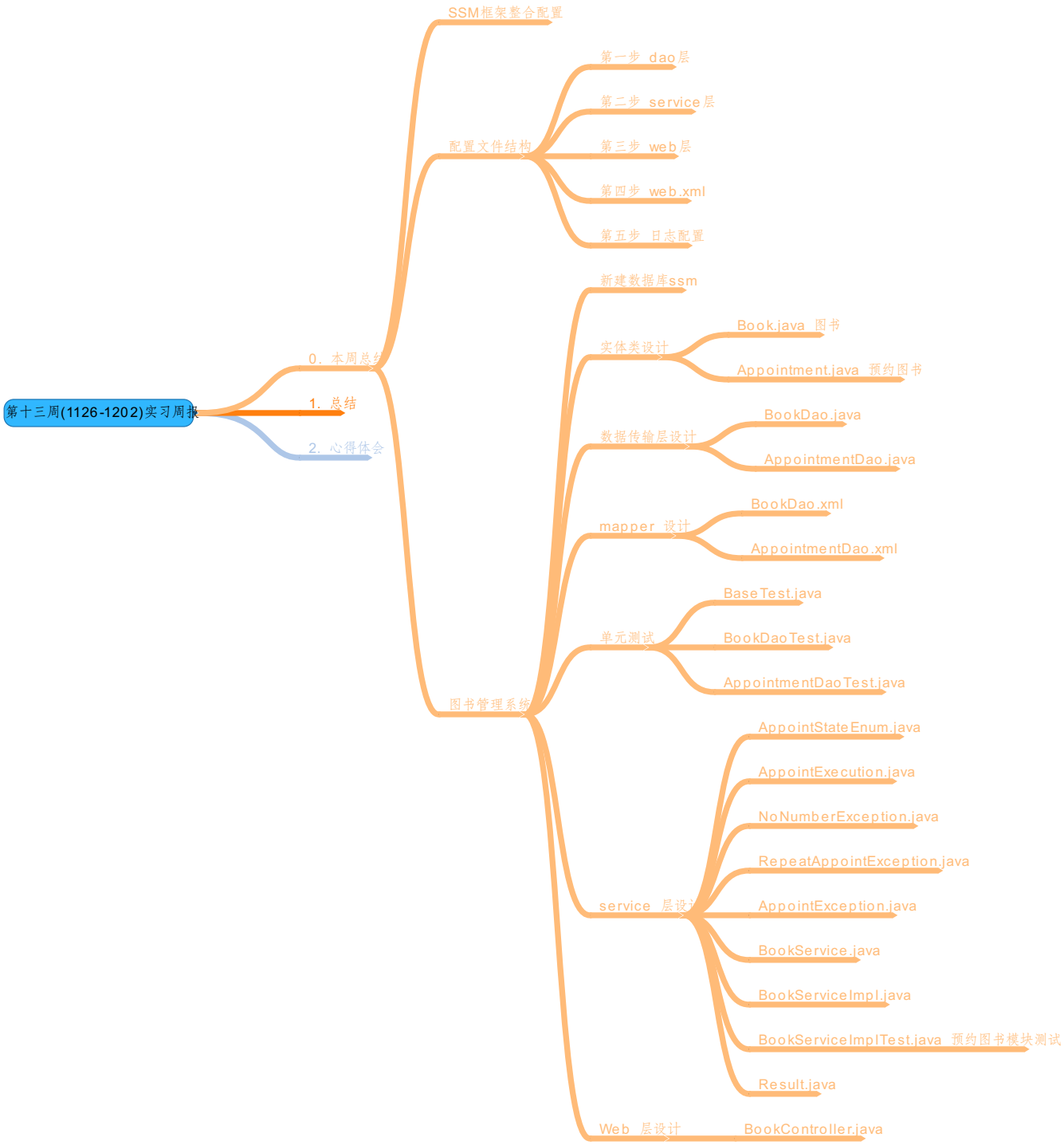
第十三周(1126-1202)

实习周报

目录



学生姓名： 曾小杰

学生班级： 计算机1505班

实习地点： 东软睿道

实习日期： 2018.11.26 - 2018.12.02

0. 本周总结

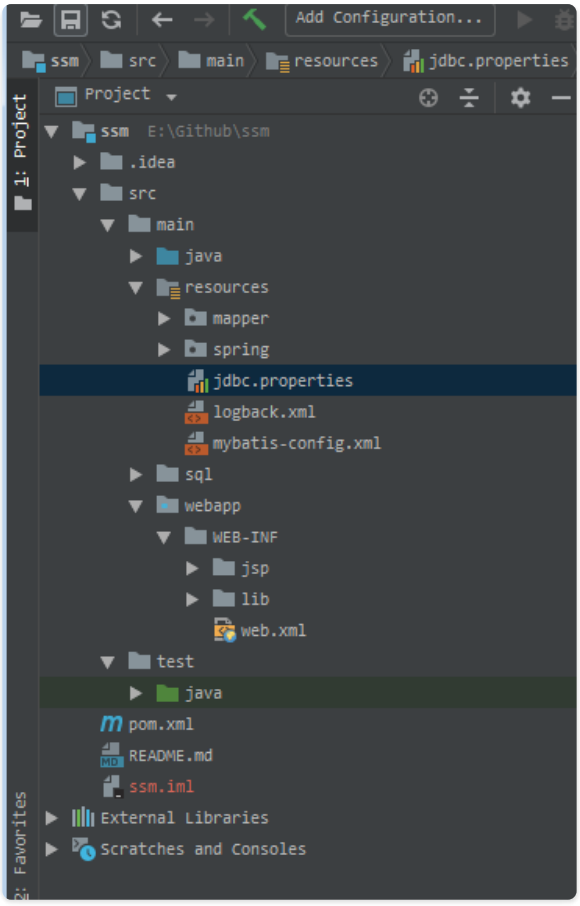
不知不觉中我学习Java（从07.21开始到现在）已经快五个月了，这段时间内学习了MySQL、Oracle、Java SE、HTML、Java WEB、Java SSM 和 linux维护的课程，其中因为秋招找工作而耽误了一个月。从刚开始的一个小白到现在，看到自己做的项目运行起来也算是一种满足吧！

在我刚开始参加招聘信息的时候，经常会看到这一点，需要具备SSM框架的技能；而且在我们的大部分教学课堂中，也会把SSM作为最核心的教学内容。 但是，我们在实际应用中发现，SpringMVC可以完全替代Struts，配合注解的方式，编程非常快捷，而且通过restful风格定义url，让地址看起来非常优雅。另外，MyBatis也可以替换Hibernate，正因为MyBatis的半自动特点，我们程序猿可以完全掌控SQL，这会让有数据库经验的程序猿能开发出高效率的SQL语句，而且XML配置管理起来也非常方便。

- SpringMVC：它用于web层，相当于controller（等价于传统的servlet和struts的action），用来处理用户请求。举个例子，用户在地址栏输入`http://网站域名/login`，那么springmvc就会拦截到这个请求，并且调用controller层中相应的方法，（中间可能包含验证用户名和密码的业务逻辑，以及查询数据库操作，但这些都不是springmvc的职责），最终把结果返回给用户，并且返回相应的页面（当然也可以只返回json/xml等格式数据）。springmvc就是做前面和后面过程的活，与用户打交道！！
- Spring：太强大了，以至于我无法用一个词或一句话来概括它。但与我们平时开发接触最多的估计就是IOC容器，它可以装载bean（也就是我们java中的类，当然也包括service dao里面的），有了这个机制，我们就不用在每次使用这个类的时候为它初始化，很少看到关键字new。另外spring的aop，事务管理等等都是我们经常用到的。
- MyBatis：如果你问我它跟鼎鼎大名的Hibernate有什么区别？我只想说，他更符合我的需求。第一，它能自由控制sql，一些对数据库有经验的人（比如我们的讲师吧）编写的代码能搞提升数据库访问的效率。第二，它可以使用xml的方式来组织管理我们的sql，因为一般程序出错很多情况下是sql出错，别人接手代码后能快速找到出错地方，甚至可以优化原来写的sql。

SSM框架整合配置

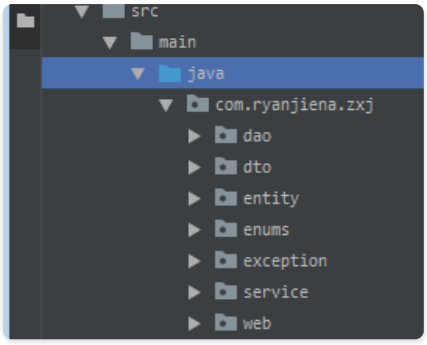
项目的目录结构（maven）



ssm project

- src：根目录，没什么好说的，下面有main和test。
  - main：主要目录，可以放java代码和一些资源文件。
    - java：存放我们的java代码，这个文件夹要使用Build Path -> Use as Source Folder，这样看包结构会方便很多，新建的包就相当于在这里新建文件夹咯。
    - resources：存放资源文件，譬如各种的spring，mybatis，log配置文件。
      - mapper：存放dao中每个方法对应的sql，在这里配置，无需写daoImpl。
      - spring：这里当然是存放spring相关的配置文件，有dao service web三层。
      - sql：其实这个可以没有，但是为了项目完整性还是加上吧。
      - webapp：这个貌似是最熟悉的目录了，用来存放我们前端的静态资源，如jsp js css。
        - resources：这里的资源是指项目的静态资源，如js css images等。
        - WEB-INF：很重要的一个目录，外部浏览器无法访问，只有内部才能访问，可以把jsp放在这里，另外就是web.xml了。你可能有疑问了，为什么上面java中的resources里面的配置文件不妨在这里，那么是不是会被外部窃取到？你想太多了，部署时候基本上只有webapp里的会直接输出到根目录，其他都会放入WEB-INF里面，项目内部依然可以使用classpath:XXX来访问，好像IDE里可以设置部署输出目录，这里扯远了
  - test：这里是单元测试分支。有时候我们需要对自己写的代码进行模块测试。
    - java：测试java代码，应遵循包名相同的原则，这个文件夹同样要使用Build Path -> Use as Source Folder，这样看包结构会方便很多。
    - resources：没什么好说的，好像也很少用到，但这个是maven的规范。

在Java 文件夹下新建包：



com.ryanjiena.zxj

在我们的Java 目录下面会存放下面几个包：

- dao：数据访问层（接口），与数据打交道，可以是数据库操作，也可以是文件读写操作，甚至是redis缓存操作，总之与数据操作有关的都放在这里，也有人叫做dal或者数据持久层都差不多意思。因为我们用的是mybatis（这样就没有daoImpl），所以可以直接在配置文件中实现接口的每个方法。
- entity：实体类，一般与数据库的表相对应，封装dao层取出来的数据为一个对象，也就是我们常说的pojo，一般只在dao层与service层之间传输。
- dto：数据传输层，刚学框架的人可能不明白这个有什么用，其实就是用于service层与web层之间传输，为什么不直接用entity（pojo）？其实在实际开发中发现，很多时间一个entity并不能满足我们的业务需求，可能呈现给用户的信息十分之多，这时候就有了dto，也相当于vo，一定不能把把这个混杂在entity里面。
- service：业务逻辑（接口），写我们的业务逻辑，也有人叫bll，在设计业务接口时候应该站在“使用者”的角度。
- serviceImpl：业务逻辑（实现），实现我们业务接口，这个里面一般写我们的事物控制语句的。
- web：控制器，springmvc就是在这里发挥作用的，一般人叫做controller控制器，相当于struts中的action。

最后一步，导入我们相应的jar包，我使用的是maven来管理我们的jar，所以只需要在pom.xml中加入相应的依赖就好了，如果不使用maven的可以自己去官网下载相应的jar，放到项目WEB-INF/lib目录下。

// pom.xml

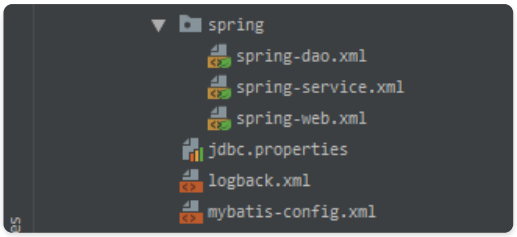
```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.soecode.ssm</groupId>
    <artifactId>ssm</artifactId>
    <packaging>war</packaging>
    <version>0.0.1-SNAPSHOT</version>
    <name>ssm Maven Webapp</name>
    <url>http://github.com/liyifeng1994/ssm</url>
    <dependencies>
        <!-- 单元测试 -->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.11</version>
        </dependency>

        <!-- 1.日志 -->
        <!-- 实现slf4j接口并整合 -->
        <dependency>
            <groupId>ch.qos.logback</groupId>
            <artifactId>logback-classic</artifactId>
            <version>1.1.1</version>
        </dependency>

        <!-- 2.数据库 -->
        <dependency>
```

```
28              <groupId>mysql</groupId>
29              <artifactId>mysql-connector-java</artifactId>
30              <version>5.1.37</version>
31              <scope>runtime</scope>
32          </dependency>
33          <dependency>
34              <groupId>c3p0</groupId>
35              <artifactId>c3p0</artifactId>
36              <version>0.9.1.2</version>
37          </dependency>
38
39          <!-- DAO: MyBatis -->
40          <dependency>
41              <groupId>org.mybatis</groupId>
42              <artifactId>mybatis</artifactId>
43              <version>3.3.0</version>
44          </dependency>
45          <dependency>
46              <groupId>org.mybatis</groupId>
47              <artifactId>mybatis-spring</artifactId>
48              <version>1.2.3</version>
49          </dependency>
50
51          <!-- 3.Servlet web -->
52          <dependency>
53              <groupId>taglibs</groupId>
54              <artifactId>standard</artifactId>
55              <version>1.1.2</version>
56          </dependency>
57          <dependency>
58              <groupId>jstl</groupId>
59              <artifactId>jstl</artifactId>
60              <version>1.2</version>
61          </dependency>
62          <dependency>
63              <groupId>com.fasterxml.jackson.core</groupId>
64              <artifactId>jackson-databind</artifactId>
65              <version>2.5.4</version>
66          </dependency>
67          <dependency>
68              <groupId>javax.servlet</groupId>
69              <artifactId>javax.servlet-api</artifactId>
70              <version>3.1.0</version>
71          </dependency>
72
73          <!-- 4.Spring -->
74          <!-- 1)Spring核心 -->
75          <dependency>
76              <groupId>org.springframework</groupId>
77              <artifactId>spring-core</artifactId>
78              <version>4.1.7.RELEASE</version>
79          </dependency>
80          <dependency>
81              <groupId>org.springframework</groupId>
82              <artifactId>spring-beans</artifactId>
83              <version>4.1.7.RELEASE</version>
84          </dependency>
85          <dependency>
86              <groupId>org.springframework</groupId>
87              <artifactId>spring-context</artifactId>
88              <version>4.1.7.RELEASE</version>
89          </dependency>
90          <!-- 2)Spring DAO层 -->
91          <dependency>
92              <groupId>org.springframework</groupId>
93              <artifactId>spring-jdbc</artifactId>
```

```
94              <version>4.1.7.RELEASE</version>
95          </dependency>
96          <dependency>
97              <groupId>org.springframework</groupId>
98              <artifactId>spring-tx</artifactId>
99              <version>4.1.7.RELEASE</version>
100         </dependency>
101         <!-- 3)Spring web -->
102         <dependency>
103             <groupId>org.springframework</groupId>
104             <artifactId>spring-web</artifactId>
105             <version>4.1.7.RELEASE</version>
106         </dependency>
107         <dependency>
108             <groupId>org.springframework</groupId>
109             <artifactId>spring-webmvc</artifactId>
110             <version>4.1.7.RELEASE</version>
111         </dependency>
112         <!-- 4)Spring test -->
113         <dependency>
114             <groupId>org.springframework</groupId>
115             <artifactId>spring-test</artifactId>
116             <version>4.1.7.RELEASE</version>
117         </dependency>
118
119         <!-- redis客户端:Jedis -->
120         <dependency>
121             <groupId>redis.clients</groupId>
122             <artifactId>jedis</artifactId>
123             <version>2.7.3</version>
124         </dependency>
125         <dependency>
126             <groupId>com.dyuproject.protostuff</groupId>
127             <artifactId>protostuff-core</artifactId>
128             <version>1.0.8</version>
129         </dependency>
130         <dependency>
131             <groupId>com.dyuproject.protostuff</groupId>
132             <artifactId>protostuff-runtime</artifactId>
133             <version>1.0.8</version>
134         </dependency>
135
136         <!-- Map工具类 -->
137         <dependency>
138             <groupId>commons-collections</groupId>
139             <artifactId>commons-collections</artifactId>
140             <version>3.2</version>
141         </dependency>
142     </dependencies>
143     <build>
144         <finalName>ssm</finalName>
145     </build>
146 </project>
147
```

配置文件结构

配置文件结构图



配置文件结构图

第一步 dao层

我们先在spring文件夹里新建spring-dao.xml文件，因为spring的配置太多，我们这里分三层，分别是dao service web。

1. 读入数据库连接相关参数（可选）
2. 配置数据连接池
   - 配置连接属性，可以不读配置项文件直接在这里写死
   - 配置c3p0，只配了几个常用的
3. 配置SqlSessionFactory对象（mybatis）
4. 扫描dao层接口，动态实现dao接口，也就是说不需要daoImpl，sql和参数都写在xml文件上.

// spring-dao.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context.xsd">
    <!-- 配置整合mybatis过程 -->
    <!-- 1.配置数据库相关参数properties的属性：${url} -->
    <context:property-placeholder location="classpath:jdbc.properties" />

    <!-- 2.数据库连接池 -->
    <bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
        <!-- 配置连接池属性 -->
        <property name="driverClass" value="${jdbc.driver}" />
        <property name="jdbcUrl" value="${jdbc.url}" />
        <property name="user" value="${jdbc.username}" />
        <property name="password" value="${jdbc.password}" />

        <!-- c3p0连接池的私有属性 -->
        <property name="maxPoolSize" value="30" />
        <property name="minPoolSize" value="10" />
        <!-- 关闭连接后不自动commit -->
        <property name="autoCommitOnClose" value="false" />
        <!-- 获取连接超时时间 -->
        <property name="checkoutTimeout" value="10000" />
        <!-- 当获取连接失败重试次数 -->
        <property name="acquireRetryAttempts" value="2" />
    </bean>

    <!-- 3.配置SqlSessionFactory对象 -->
    <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
        <!-- 注入数据库连接池 -->
        <property name="dataSource" ref="dataSource" />
        <!-- 配置MyBaties全局配置文件:mybatis-config.xml -->
        <property name="configLocation" value="classpath:mybatis-config.xml" />
        <!-- 扫描entity包 使用别名 -->
```

```
        <property name="typeAliasesPackage" value="com.soecode.lyf.entity" />
        <!-- 扫描sql配置文件:mapper需要的xml文件 -->
        <property name="mapperLocations" value="classpath:mapper/*.xml" />
    </bean>

    <!-- 4.配置扫描Dao接口包，动态实现Dao接口，注入到spring容器中 -->
    <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
        <!-- 注入sqlSessionFactory -->
        <property name="sqlSessionFactoryBeanName" value="sqlSessionFactory" />
        <!-- 给出需要扫描Dao接口包 -->
        <property name="basePackage" value="com.soecode.lyf.dao" />
    </bean>
</beans>
```

在`resources`文件夹里新建一个`jdbc.properties`文件，存放我们4个最常见的数据库连接属性.

// jdbc.properties

```
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3307/ssm?useUnicode=true&characterEncoding=utf8
jdbc.username=root
jdbc.password=root
```

因为这里用到了mybatis，所以需要配置mybatis核心文件，在recources文件夹里新建mybatis-config.xml文件。

1. 使用自增主键
2. 使用列别名
3. 开启驼峰命名转换 `create_time` -> `createTime`

// mybatis-config.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <!-- 配置全局属性 -->
    <settings>
        <!-- 使用jdbc的getGeneratedKeys获取数据库自增主键值 -->
        <setting name="useGeneratedKeys" value="true" />

        <!-- 使用列别名替换列名 默认:true -->
        <setting name="useColumnLabel" value="true" />

        <!-- 开启驼峰命名转换:Table{create_time} -> Entity{createTime} -->
        <setting name="mapUnderscoreToCamelCase" value="true" />
    </settings>
</configuration>
```

第二步 service层

刚弄好dao层，接下来到service层了。在spring文件夹里新建`spring-service.xml`文件。

1. 扫描`service`包所有注解 `@Service`
2. 配置事务管理器，把事务管理交由`spring`来完成

3. 配置基于注解的声明式事务，可以直接在方法上`@Transaction`

// spring-service.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
     http://www.springframework.org/schema/beans/spring-beans.xsd
     http://www.springframework.org/schema/context
     http://www.springframework.org/schema/context/spring-context.xsd
     http://www.springframework.org/schema/tx
     http://www.springframework.org/schema/tx/spring-tx.xsd">
    <!-- 扫描service包下所有使用注解的类型 -->
    <context:component-scan base-package="com.soecode.lyf.service" />

    <!-- 配置事务管理器 -->
    <bean id="transactionManager"
        class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <!-- 注入数据库连接池 -->
        <property name="dataSource" ref="dataSource" />
    </bean>

    <!-- 配置基于注解的声明式事务 -->
    <tx:annotation-driven transaction-manager="transactionManager" />
</beans>
```

第三步 web层

配置web层，在`spring`文件夹里新建`spring-web.xml`文件。

1. 开启`SpringMVC`注解模式，可以使用`@RequestMapping`，`@PathVariable`，`@ResponseBody`等
2. 对静态资源处理，如js，css，jpg等
3. 配置jsp 显示`ViewResolver`，例如在`controller`中某个方法返回一个string类型的”login”，实际上会返回”`/WEB-INF/login.jsp`”
4. 扫描web层 `@Controller`

// spring-web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
     http://www.springframework.org/schema/beans/spring-beans.xsd
     http://www.springframework.org/schema/context
     http://www.springframework.org/schema/context/spring-context.xsd
     http://www.springframework.org/schema/mvc
     http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd">
    <!-- 配置SpringMVC -->
    <!-- 1.开启SpringMVC注解模式 -->
    <!-- 简化配置:
         (1)自动注册DefaultAnootationHandlerMapping,AnotationMethodHandlerAdapter
         (2)提供一些列：数据绑定，数字和日期的format @NumberFormat, @DateTimeFormat,
xml,json默认读写支持
     -->
    <mvc:annotation-driven />
```

```
19
20      <!-- 2.静态资源默认servlet配置
21           (1)加入对静态资源的处理: js,gif,png
22           (2)允许使用"/"做整体映射
23       -->
24       <mvc:default-servlet-handler/>
25
26       <!-- 3.配置jsp 显示ViewResolver -->
27       <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
28        <property name="viewClass" value="org.springframework.web.servlet.view.JstlView
/>
29        <property name="prefix" value="/WEB-INF/jsp/" />
30        <property name="suffix" value=".jsp" />
31       </bean>
32
33       <!-- 4.扫描web相关的bean -->
34       <context:component-scan base-package="com.soecode.lyf.web" />
35  </beans>
```

第四步 web.xml

最后就是修改`web.xml`文件了，它在`webapp`的`WEB-INF`下。

// web.xml

```
1   <web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2       xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
3                           http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
4       version="3.1" metadata-complete="true">
5       <!-- 如果是用mvn命令生成的xml，需要修改servlet版本为3.1 -->
6       <!-- 配置DispatcherServlet -->
7       <servlet>
8           <servlet-name>mvc-dispatcher</servlet-name>
9           <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
    class>
10          <!-- 配置springMVC需要加载的配置文件
11               spring-dao.xml,spring-service.xml,spring-web.xml
12               Mybatis - > spring -> springmvc
13            -->
14          <init-param>
15              <param-name>contextConfigLocation</param-name>
16              <param-value>classpath:spring/spring-*.xml</param-value>
17          </init-param>
18      </servlet>
19      <servlet-mapping>
20          <servlet-name>mvc-dispatcher</servlet-name>
21          <!-- 默认匹配所有的请求 -->
22          <url-pattern>/</url-pattern>
23      </servlet-mapping>
24  </web-app>
25
```

第五步 日志配置

我们在项目中经常会使用到日志，所以这里还有配置日志xml，在`resources`文件夹里新建`logback.xml`文件，所

给出的日志输出格式也是最基本的控制台呼出，大家有兴趣查看logback官方文档。

// logback.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration debug="true">
    <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
        <!-- encoders are by default assigned the type
ch.qos.logback.classic.encoder.PatternLayoutEncoder -->
        <encoder>
            <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} -
%msg%n</pattern>
        </encoder>
    </appender>

    <root level="debug">
        <appender-ref ref="STDOUT" />
    </root>
</configuration>
```

图书管理系统

下面以图书管理系统中【查询图书】和【预约图书】业务来做一个demo。

新建数据库ssm

首先新建数据库名为`ssm`，再创建两张表：图书表`book`和预约图书表`appointment`，并且为`book`表初始化一些数据，sql如下。

// schema.sql

```
-- 创建图书表
CREATE TABLE `book` (
  `book_id` bigint(20) NOT NULL AUTO_INCREMENT COMMENT '图书ID',
  `name` varchar(100) NOT NULL COMMENT '图书名称',
  `number` int(11) NOT NULL COMMENT '馆藏数量',
  PRIMARY KEY (`book_id`)
) ENGINE=InnoDB AUTO_INCREMENT=1000 DEFAULT CHARSET=utf8 COMMENT='图书表';

-- 初始化图书数据
INSERT INTO `book` (`book_id`, `name`, `number`)
VALUES
    (1000, 'Java程序设计', 10),
    (1001, '数据结构', 10),
    (1002, '设计模式', 10),
    (1003, '编译原理', 10);

-- 创建预约图书表
CREATE TABLE `appointment` (
  `book_id` bigint(20) NOT NULL COMMENT '图书ID',
  `student_id` bigint(20) NOT NULL COMMENT '学号',
  `appoint_time` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP COMMENT '预约时间' ,
  PRIMARY KEY (`book_id`, `student_id`),
  INDEX `idx_appoint_time` (`appoint_time`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='预约图书表';
```

```
25
```

实体类设计

在`entity`包中添加两个对应的实体，图书实体`Book.java`和预约图书实体`Appointment.java`。

Book.java 图书

```
1  package  com.ryanjiena.zxj.entity;
2
3  /**
4    * 图书实体
5    */
6  public  class  Book  {
7
8      private  long  bookId;//  图书ID
9
10     private  String  name;//  图书名称
11
12     private  int  number;//  馆藏数量
13
14     public  Book()  {
15     }
16
17     public  Book(long  bookId,  String  name,  int  number)  {
18         this.bookId  =  bookId;
19         this.name  =  name;
20         this.number  =  number;
21     }
22
23     public  long  getBookId()  {
24         return  bookId;
25     }
26
27     public  void  setBookId(long  bookId)  {
28         this.bookId  =  bookId;
29     }
30
31     public  String getName()  {
32         return  name;
33     }
34
35     public  void  setName(String  name)  {
36         this.name  =  name;
37     }
38
39     public  int  getNumber()  {
40         return  number;
41     }
42
43     public  void  setNumber(int  number)  {
44         this.number  =  number;
45     }
46
47     @Override
48     public  String toString()  {
49         return  "Book  [bookId="  +  bookId  +  ",  name="  +  name  +  ",  number="  +  numb
   +  "]";
50     }
```

```
51
52
53  }
54
```

Appointment.java 预约图书

```
1   package  com.ryanjiena.zxj.entity;
2
3   import  java.util.Date;
4
5   /**
6    * 预约图书实体
7    */
8   public  class  Appointment  {
9
10      private  long  bookId;//  图书ID
11
12      private  long  studentId;//  学号
13
14      private  Date  appointTime;//  预约时间
15
16      //  多对一的复合属性
17      private  Book  book;//  图书实体
18
19      public  Appointment()  {
20      }
21
22      public  Appointment(long  bookId, long  studentId,  Date  appointTime)  {
23          this.bookId  =  bookId;
24          this.studentId  =  studentId;
25          this.appointTime  =  appointTime;
26      }
27
28      public  Appointment(long  bookId, long  studentId,  Date  appointTime,  Book  book)
    {
29          this.bookId  =  bookId;
30          this.studentId  =  studentId;
31          this.appointTime  =  appointTime;
32          this.book  =  book;
33      }
34
35      public  long  getBookId()  {
36          return  bookId;
37      }
38
39      public  void  setBookId(long  bookId)  {
40          this.bookId  =  bookId;
41      }
42
43      public  long  getStudentId()  {
44          return  studentId;
45      }
46
47      public  void  setStudentId(long  studentId)  {
48          this.studentId  =  studentId;
49      }
50
51      public  Date  getAppointTime()  {
52          return  appointTime;
53      }
54
```

```
55      public void setAppointTime(Date appointTime) {
56          this.appointTime = appointTime;
57      }
58
59      public Book getBook() {
60          return book;
61      }
62
63      public void setBook(Book book) {
64          this.book = book;
65      }
66
67      @Override
68      public String toString() {
69          return "Appointment [bookId=" + bookId + ", studentId=" + studentId + ",
    appointTime=" + appointTime + "]";
70      }
71
72  }
73
```

数据传输层设计

在`dao`包新建接口`BookDao.java`和`Appointment.java`

BookDao.java

```
1   package com.ryanjiena.zxj.dao;
2
3   import java.util.List;
4
5   import com.ryanjiena.zxj.entity.Book;
6   import org.apache.ibatis.annotations.Param;
7
8   public interface BookDao {
9
10      /**
11       * 通过ID查询单本图书
12       *
13       * @param id
14       * @return
15       */
16      Book queryById(long id);
17
18      /**
19       * 查询所有图书
20       *
21       * @param offset 查询起始位置
22       * @param limit 查询条数
23       * @return
24       */
25      List<Book> queryAll(@Param("offset") int offset, @Param("limit") int limit);
26
27      /**
28       * 减少馆藏数量
29       *
30       * @param bookId
31       * @return 如果影响行数等于>1，表示更新的记录行数
32       */
```

```
33      int reduceNumber(long bookId);
34
35  }
36
```

AppointmentDao.java

```
1   package com.ryanjiena.zxj.dao;
2
3   import com.ryanjiena.zxj.entity.Appointment;
4   import org.apache.ibatis.annotations.Param;
5
6   public interface AppointmentDao {
7
8       /**
9        * 插入预约图书记录
10       *
11       * @param bookId
12       * @param studentId
13       * @return 插入的行数
14       */
15      int insertAppointment(@Param("bookId") long bookId, @Param("studentId") long
    studentId);
16
17      /**
18       * 通过主键查询预约图书记录，并且携带图书实体
19       *
20       * @param bookId
21       * @param studentId
22       * @return
23       */
24      Appointment queryByKeyWithBook(@Param("bookId") long bookId, @Param("studentId"
    long studentId);
25
26  }
27
```

PS: 为什么要给方法的参数添加@Param注解呢？是因为该方法有两个或以上的参数，一定要加，不然mybatis识别不了。上面的BookDao接口的queryById方法和reduceNumber方法只有一个参数book_id，所以可以不用加@Param注解，当然加了也可以。

mapper 设计

在`mapper`目录里新建两个文件`BookDao.xml`和`AppointmentDao.xml`，分别对应上面两个`dao`接口，代码如下。

BookDao.xml

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <!DOCTYPE mapper
3       PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4       "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5   <mapper namespace="com.ryanjiena.zxj.dao.BookDao">
6       <!-- 目的：为dao接口方法提供sql语句配置 -->
```

```
    <select id="queryById" resultType="Book" parameterType="long">
        <!-- 具体的sql -->
        SELECT
            book_id,
            name,
            number
        FROM
            book
        WHERE
            book_id = #{bookId}
    </select>

    <select id="queryAll" resultType="Book">
        SELECT
            book_id,
            name,
            number
        FROM
            book
        ORDER BY
            book_id
        LIMIT #{offset}, #{limit}
    </select>

    <update id="reduceNumber">
        UPDATE book
        SET number = number - 1
        WHERE
            book_id = #{bookId}
        AND number > 0
    </update>
</mapper>
```

AppointmentDao.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper
    PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.ryanjiena.zxj.dao.AppointmentDao">
    <insert id="insertAppointment">
        <!-- ignore 主键冲突, 报错 -->
        INSERT ignore INTO appointment (book_id, student_id)
        VALUES (#{bookId}, #{studentId})
    </insert>

    <select id="queryByKeyWithBook" resultType="Appointment">
        <!-- 如何告诉MyBatis把结果映射到Appointment同时映射book属性 -->
        <!-- 可以自由控制SQL -->
        SELECT
            a.book_id,
            a.student_id,
            a.appoint_time,
            b.book_id "book.book_id",
            b.`name` "book.name",
            b.number "book.number"
        FROM
            appointment a
        INNER JOIN book b ON a.book_id = b.book_id
        WHERE
            a.book_id = #{bookId}
        AND a.student_id = #{studentId}
```

```
    </select>
</mapper>
```

PS: **总结：`namespace`是该xml对应的接口全名，`select`和`update`中的`id`对应方法名，`resultType`是返回值类型，`parameterType`是参数类型（这个其实可选），最后`#{...}`中填写的是方法的参数，看懂了是不是很简单！！我也这么觉得~ 还有一个小技巧要交给大家，就是在返回`Appointment`对象包含了一个属性名为`book`的`Book`对象，那么可以使用"`book.属性名`"的方式来取值，看上面`queryByKeyWithBook`方法的sql。**

单元测试

`dao`层写完了，接下来`test`对应的`package`写我们测试方法吧。
因为我们之后会写很多测试方法，在测试前需要让程序读入`spring-dao`和`mybatis`等配置文件，所以我这里就抽离出来一个`BaseTest`类，只要是测试方法就继承它，这样那些繁琐的重复的代码就不用写那么多了。

BaseTest.java

```
package com.ryanjiena.zxj;

import org.junit.runner.RunWith;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

/**
 * 配置spring和junit整合，junit启动时加载springIOC容器 spring-test,junit
 */
@RunWith(SpringJUnit4ClassRunner.class)
// 告诉junit spring配置文件
@ContextConfiguration({ "classpath:spring/spring-dao.xml",
"classpath:spring/spring-service.xml" })
public class BaseTest {

}

```

因为`spring-service`在`service`层的测试中会时候到，这里也一起引入算了！

新建`BookDaoTest.java`和`AppointmentDaoTest.java`两个`dao`测试文件。

BookDaoTest.java

```
package com.ryanjiena.zxj.dao;

import java.util.List;

import com.ryanjiena.zxj.BaseTest;
import com.ryanjiena.zxj.entity.Book;
import org.junit.Test;
import org.springframework.beans.factory.annotation.Autowired;

public class BookDaoTest extends BaseTest {

```

```
    @Autowired
    private BookDao bookDao;

    @Test
    public void testQueryById() throws Exception {
        long bookId = 1000;
        Book book = bookDao.queryById(bookId);
        System.out.println(book);
    }

    @Test
    public void testQueryAll() throws Exception {
        List<Book> books = bookDao.queryAll(0, 4);
        for (Book book : books) {
            System.out.println(book);
        }
    }

    @Test
    public void testReduceNumber() throws Exception {
        long bookId = 1000;
        int update = bookDao.reduceNumber(bookId);
        System.out.println("update=" + update);
    }

}

```

AppointmentDaoTest.java

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper
    PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.ryanjiena.zxj.dao.AppointmentDao">
    <insert id="insertAppointment">
        <!-- ignore 主键冲突, 报错 -->
        INSERT ignore INTO appointment (book_id, student_id)
        VALUES (#{bookId}, #{studentId})
    </insert>

    <select id="queryByKeyWithBook" resultType="Appointment">
        <!-- 如何告诉MyBatis把结果映射到Appointment同时映射book属性 -->
        <!-- 可以自由控制SQL -->
        SELECT
            a.book_id,
            a.student_id,
            a.appoint_time,
            b.book_id "book.book_id",
            b.`name` "book.name",
            b.number "book.number"
        FROM
            appointment a
        INNER JOIN book b ON a.book_id = b.book_id
        WHERE
            a.book_id = #{bookId}
        AND a.student_id = #{studentId}
    </select>
</mapper>
```

service 层设计

首先，在写我们的控制器之前，我们先定义几个预约图书操作返回码的数据字典，也就是我们要返回给客户端的信息。我们这类使用枚举类。

预约业务操作返回码说明

返回码	说明
1	预约成功
0	库存不足
-1	重复预约
-2	系统异常

新建一个包叫enums，在里面新建一个枚举类AppointStateEnum.java，用来定义预约业务的数据字典。

AppointStateEnum.java

```java
package com.ryanjiena.zxj.enums;

/**
 * 使用枚举表述常量数据字典
 */
public enum AppointStateEnum {

    SUCCESS(1, "预约成功"), NO_NUMBER(0, "库存不足"), REPEAT_APPOINT(-1, "重复预约"), INNER_ERROR(-2, "系统异常");

    private int state;

    private String stateInfo;

    private AppointStateEnum(int state, String stateInfo) {
        this.state = state;
        this.stateInfo = stateInfo;
    }

    public int getState() {
        return state;
    }

    public String getStateInfo() {
        return stateInfo;
    }

    public static AppointStateEnum stateOf(int index) {
        for (AppointStateEnum state : values()) {
            if (state.getState() == index) {
                return state;
            }
        }
        return null;
    }

}

```

接下来，在dto包下新建AppointExecution.java用来存储我们执行预约操作的返回结果。

AppointExecution.java

```java
package com.ryanjiena.zxj.dto;

import com.ryanjiena.zxj.entity.Appointment;
import com.ryanjiena.zxj.enums.AppointStateEnum;

/**
 * 封装预约执行后结果
 */
public class AppointExecution {

    // 图书ID
    private long bookId;

    // 秒杀预约结果状态
    private int state;

    // 状态标识
    private String stateInfo;

    // 预约成功对象
    private Appointment appointment;

    public AppointExecution() {
    }

    // 预约失败的构造器
    public AppointExecution(long bookId, AppointStateEnum stateEnum) {
        this.bookId = bookId;
        this.state = stateEnum.getState();
        this.stateInfo = stateEnum.getStateInfo();
    }

    // 预约成功的构造器
    public AppointExecution(long bookId, AppointStateEnum stateEnum, Appointment appointment) {
        this.bookId = bookId;
        this.state = stateEnum.getState();
        this.stateInfo = stateEnum.getStateInfo();
        this.appointment = appointment;
    }

    public long getBookId() {
        return bookId;
    }

    public void setBookId(long bookId) {
        this.bookId = bookId;
    }

    public int getState() {
        return state;
    }

    public void setState(int state) {
        this.state = state;
    }

    public String getStateInfo() {
        return stateInfo;
    }
```

```
60
61      public void setStateInfo(String stateInfo) {
62          this.stateInfo = stateInfo;
63      }
64
65      public Appointment getAppointment() {
66          return appointment;
67      }
68
69      public void setAppointment(Appointment appointment) {
70          this.appointment = appointment;
71      }
72
73      @Override
74      public String toString() {
75          return "AppointExecution [bookId=" + bookId + ", state=" + state + ",
    stateInfo=" + stateInfo + ", appointment="
76                  + appointment + "]";
77      }
78
79  }
80
```

接着，在`exception`包下新建三个文件`NoNumberException.java`，`RepeatAppointException.java`，`AppointException.java`,预约业务异常类（都需要继承`RuntimeException`），分别是无库存异常、重复预约异常、预约未知错误异常，用于业务层非成功情况下的返回（即成功返回结果，失败抛出异常）。

NoNumberException.java

```
1   package com.ryanjiena.zxj.exception;
2
3   /**
4    * 库存不足异常
5    */
6   public class NoNumberException extends RuntimeException {
7
8       public NoNumberException(String message) {
9           super(message);
10      }
11
12      public NoNumberException(String message, Throwable cause) {
13          super(message, cause);
14      }
15
16  }
17
```

RepeatAppointException.java

```
1   package com.ryanjiena.zxj.exception;
2
3   /**
4    * 重复预约异常
5    */
6   public class RepeatAppointException extends RuntimeException {
7
8       public RepeatAppointException(String message) {
```

```
9           super(message);
10      }
11
12      public RepeatAppointException(String message, Throwable cause) {
13          super(message, cause);
14      }
15
16  }
17
```

AppointException.java

```
1   package com.ryanjiena.zxj.exception;
2
3   /**
4    * 预约业务异常
5    */
6   public class AppointException extends RuntimeException {
7
8       public AppointException(String message) {
9           super(message);
10      }
11
12      public AppointException(String message, Throwable cause) {
13          super(message, cause);
14      }
15
16  }
17
```

终于可以编写业务代码了，在`service`包下新建`BookService.java`图书业务接口。

BookService.java

```
1   package com.ryanjiena.zxj.service;
2
3   import java.util.List;
4
5   import com.ryanjiena.zxj.dto.AppointExecution;
6   import com.ryanjiena.zxj.entity.Book;
7
8   /**
9    * 业务接口：站在"使用者"角度设计接口 三个方面：方法定义粒度，参数，返回类型（return 类型/异
    常）
10   */
11  public interface BookService {
12
13      /**
14       * 查询一本图书
15       *
16       * @param bookId
17       * @return
18       */
19      Book getById(long bookId);
20
21      /**
22       * 查询所有图书
23       *
```

```
     * @return
     */
    List<Book> getList();

    /**
     * 预约图书
     *
     * @param bookId
     * @param studentId
     * @return
     */
    AppointExecution appoint(long bookId, long studentId);

}
```

终于可以编写业务代码了，在service包下新建BookService.java图书业务接口。

BookServiceImpl.java

```
package com.ryanjiena.zxj.service.impl;

import java.util.List;

import com.ryanjiena.zxj.dao.AppointmentDao;
import com.ryanjiena.zxj.dao.BookDao;
import com.ryanjiena.zxj.dto.AppointExecution;
import com.ryanjiena.zxj.entity.Appointment;
import com.ryanjiena.zxj.entity.Book;
import com.ryanjiena.zxj.enums.AppointStateEnum;
import com.ryanjiena.zxj.exception.AppointException;
import com.ryanjiena.zxj.exception.NoNumberException;
import com.ryanjiena.zxj.exception.RepeatAppointException;
import com.ryanjiena.zxj.service.BookService;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

@Service
public class BookServiceImpl implements BookService {

    private Logger logger = LoggerFactory.getLogger(this.getClass());

    // 注入Service依赖
    @Autowired
    private BookDao bookDao;

    @Autowired
    private AppointmentDao appointmentDao;


    @Override
    public Book getById(long bookId) {
        return bookDao.queryById(bookId);
    }

    @Override
    public List<Book> getList() {
        return bookDao.queryAll(0, 1000);
    }
```

```
43
44      @Override
45      @Transactional
46      /**
47       * 使用注解控制事务方法的优点:  1.开发团队达成一致约定，明确标注事务方法的编程风格
48       * 2.保证事务方法的执行时间尽可能短，不要穿插其他网络操作，RPC/HTTP请求或者剥离到事务方法外部
49       * 3.不是所有的方法都需要事务，如只有一条修改操作，只读操作不需要事务控制
50       */
51      public AppointExecution appoint(long bookId, long studentId) {
52          try {
53              // 减库存
54              int update = bookDao.reduceNumber(bookId);
55              if (update <= 0) {// 库存不足
56                  throw new NoNumberException("no number");
57              } else {
58                  // 执行预约操作
59                  int insert = appointmentDao.insertAppointment(bookId, studentId);
60                  if (insert <= 0) {// 重复预约
61                      throw new RepeatAppointException("repeat appoint");
62                  } else {// 预约成功
63                      Appointment appointment = appointmentDao.queryByKeyWithBook(boo
    studentId);
64                      return new AppointExecution(bookId, AppointStateEnum.SUCCESS,
    appointment);
65                  }
66              }
67          } catch (NoNumberException e1) {
68              throw e1;
69          } catch (RepeatAppointException e2) {
70              throw e2;
71          } catch (Exception e) {
72              logger.error(e.getMessage(), e);
73              // 所有编译期异常转换为运行期异常
74              throw new AppointException("appoint inner error:" + e.getMessage());
75          }
76      }
77
78  }
79
```

BookServiceImplTest.java 预约图书模块测试

```
1   package com.ryanjiena.zxj.service.impl;
2
3   import static org.junit.Assert.fail;
4
5   import com.ryanjiena.zxj.dto.AppointExecution;
6   import org.junit.Test;
7   import org.springframework.beans.factory.annotation.Autowired;
8
9   import com.ryanjiena.zxj.BaseTest;
10  import com.ryanjiena.zxj.service.BookService;
11
12  public class BookServiceImplTest extends BaseTest {
13
14      @Autowired
15      private BookService bookService;
16
17      @Test
18      public void testAppoint() throws Exception {
19          long bookId = 1001;
20          long studentId = 12345678910L;
```

```
        AppointExecution execution = bookService.appoint(bookId, studentId);
        System.out.println(execution);
    }

}

```

Result.java

在dto包里新建一个封装json返回结果的类Result.java，设计成泛型。

```
package com.ryanjiena.zxj.dto;

/**
 * 封装json对象，所有返回结果都使用它
 */
public class Result<T> {

    private boolean success;// 是否成功标志

    private T data;// 成功时返回的数据

    private String error;// 错误信息

    public Result() {
    }

    // 成功时的构造器
    public Result(boolean success, T data) {
        this.success = success;
        this.data = data;
    }

    // 错误时的构造器
    public Result(boolean success, String error) {
        this.success = success;
        this.error = error;
    }

    public boolean isSuccess() {
        return success;
    }

    public void setSuccess(boolean success) {
        this.success = success;
    }

    public T getData() {
        return data;
    }

    public void setData(T data) {
        this.data = data;
    }

    public String getError() {
        return error;
    }

    public void setError(String error) {
        this.error = error;
    }
```

```
52
53      @Override
54      public  String toString()  {
55          return  "JsonResult  [success="  +  success  +  ",  data="  +  data  +  ",  error="
error  +  "]";
56      }
57
58  }
59
```

Web 层设计

最后我们写`web`层，也就是`controller`，我们在`web`包下新建`BookController.java`文件。

BookController.java

```
1   package  com.ryanjiena.zxj.web;
2
3   import  java.util.List;
4
5   import  com.ryanjiena.zxj.dto.Result;
6   import  com.ryanjiena.zxj.entity.Book;
7   import  com.ryanjiena.zxj.enums.AppointStateEnum;
8   import  org.slf4j.Logger;
9   import  org.slf4j.LoggerFactory;
10  import  org.springframework.beans.factory.annotation.Autowired;
11  import  org.springframework.stereotype.Controller;
12  import  org.springframework.ui.Model;
13  import  org.springframework.web.bind.annotation.PathVariable;
14  import  org.springframework.web.bind.annotation.RequestMapping;
15  import  org.springframework.web.bind.annotation.RequestMethod;
16  import  org.springframework.web.bind.annotation.ResponseBody;
17  import  org.springframework.web.bind.annotation.RequestParam;
18
19  import  com.ryanjiena.zxj.dto.AppointExecution;
20  import  com.ryanjiena.zxj.exception.NoNumberException;
21  import  com.ryanjiena.zxj.exception.RepeatAppointException;
22  import  com.ryanjiena.zxj.service.BookService;
23
24  @Controller
25  @RequestMapping("/book")  //  url:/模块/资源/{id}/细分  /seckill/list
26  public  class  BookController  {
27
28      private  Logger  logger  =  LoggerFactory.getLogger(this.getClass());
29
30      @Autowired
31      private  BookService  bookService;
32
33      @RequestMapping(value  =  "/list",  method  =  RequestMethod.GET)
34      private  String list(Model  model)  {
35          List<Book>  list  =  bookService.getList();
36          model.addAttribute("list",  list);
37          //  list.jsp  +  model  =  ModelAndView
38          return  "list";//  WEB-INF/jsp/"list".jsp
39      }
40
41      @RequestMapping(value  =  "/{bookId}/detail",  method  =  RequestMethod.GET)
42      private  String detail(@PathVariable("bookId")  Long  bookId,  Model  model)  {
43          if  (bookId  ==  null)  {
```

```
44              return "redirect:/book/list";
45          }
46          Book book = bookService.getById(bookId);
47          if (book == null) {
48              return "forward:/book/list";
49          }
50          model.addAttribute("book", book);
51          return "detail";
52      }
53
54      // ajax json
55      @RequestMapping(value = "/{bookId}/appoint", method = RequestMethod.POST,
    produces = {
56              "application/json; charset=utf-8" })
57      @ResponseBody
58      private Result<AppointExecution> appoint(@PathVariable("bookId") Long bookId,
    @RequestParam("studentId") Long studentId) {
59          if (studentId == null || studentId.equals("")) {
60              return new Result<>(false, "学号不能为空");
61          }
62          AppointExecution execution = null;
63          try {
64              execution = bookService.appoint(bookId, studentId);
65          } catch (NoNumberException e1) {
66              execution = new AppointExecution(bookId, AppointStateEnum.NO_NUMBER);
67          } catch (RepeatAppointException e2) {
68              execution = new AppointExecution(bookId, AppointStateEnum.REPEAT_APPOIN
69          } catch (Exception e) {
70              execution = new AppointExecution(bookId, AppointStateEnum.INNER_ERROR);
71          }
72          return new Result<AppointExecution>(true, execution);
73      }
74
75  }
76
```

1. 总结

在项目开发的时候,由于自己的错误导致代码一直运行不出来。最后只能找讲师，讲师给我提示用模块单元测试来解决。听到之后才发现自己真的好蠢，学习了单元测试尽然不用……最后，修改BookController几处错误，代码终于完美运行!

1. `detail`方法不是返回`json`的，故不用加`@ResponseBody`注解
2. `appoint`方法应该加上`@ResponseBody`注解
3. 另外`studentId`参数注解应该是`@RequestParam`
4. 至于`controller`测试，测试appoint方法可不必写`jsp`，用`curl`就行，比如

```
1  curl -H "Accept: application/json; charset=utf-8" -d "studentId=1234567890"
   localhost:8080/book/1003/appoint
```

2. 心得体会

经过这一周的培训，让我了解到自己的不足，在今后的学习中，我会更加的努力的奋斗下去，完善自我。

1. 继续学习，不断提升自己的理论素养；
2. 制定相应的实习计划，及时对所学知识进行巩固；
3. 加强信心，坚持下去!

成功根本没有什么秘诀可言，如果真是有的话，就是两个：第一个就是坚持到底，永不放弃；第二个是当你想放弃的时候，回过头来看看第一个秘诀：坚持到底，永不放弃。