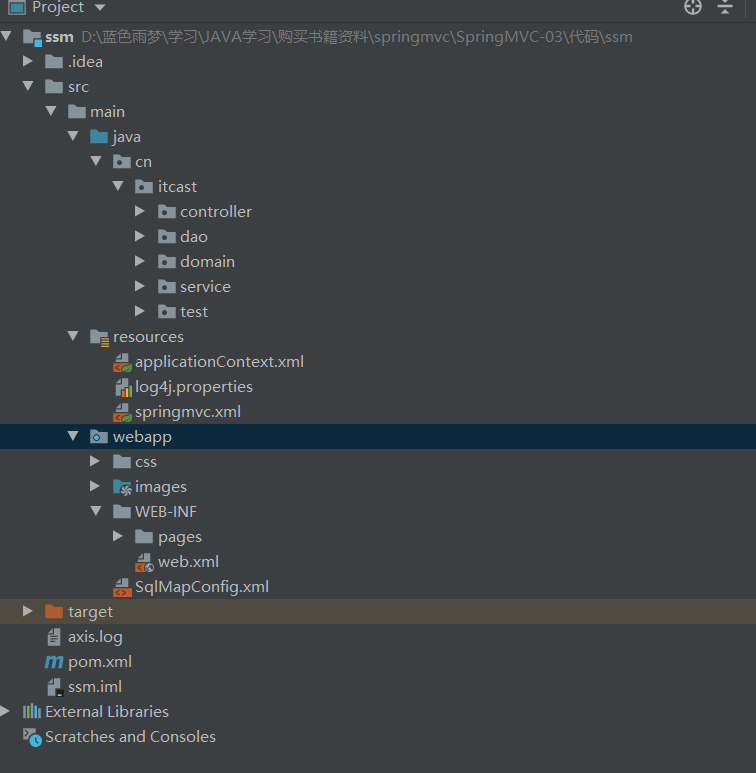
# SSM整合9

分三层：springmvc,spring,mybatis

选择配置文件+注解方式

## 整合：

目录结构：



### 思路：

1. 搭建整合环境
2. 配置spring
3. 用spring整合springmvc框架
4. 用spring整合mybatis框架

### 第一步：搭建整合环境

·ssm整合的方式：XML+注解

1. **创建maven工程：选择wabapp模式**

ssm-parent父工程：打包方式选择pom.xml

ssm-web子模块:打包方式是war

ssm-service子模块:打包方式是jar

ssm-dao子模块：打包方式是jar

ssm-domain子模块：打包方式是jar

**a2、增加Maven property:为了让工程加载得更快**

Name:archetypeCatalog

Value:internal

1. **在ssm-parent的pom.xml文件中引入坐标依赖：**

<properties>

<spring.version>5.0.2.RELEASE</spring.version>

<slf4j.version>1.6.6</slf4j.version>

<log4j.version>1.2.12</log4j.version>

<mysql.version>5.1.6</mysql.version>

<mybatis.version>3.4.5</mybatis.version>

</properties>

<dependencies><!-- spring -->

<dependency>

<groupId>org.aspectj</groupId>

<artifactId>aspectjweaver</artifactId>

<version>1.6.8</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

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

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

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

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

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

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

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

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-test</artifactId>

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

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-tx</artifactId>

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

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

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

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version><scope>compile</scope>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

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

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>servlet-api</artifactId>

<version>2.5</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>javax.servlet.jsp</groupId>

<artifactId>jsp-api</artifactId>

<version>2.0</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>jstl</groupId>

<artifactId>jstl</artifactId>

<version>1.2</version>

</dependency>

<!-- log start -->

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>${log4j.version}</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>${slf4j.version}</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>${slf4j.version}</version>

</dependency>

<!-- log end -->

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis</artifactId>

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

</dependency><dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis-spring</artifactId>

<version>1.3.0</version>

</dependency>

<dependency>

<groupId>c3p0</groupId>

<artifactId>c3p0</artifactId>

<version>0.9.1.2</version>

<type>jar</type>

<scope>compile</scope>

</dependency>

</dependencies>

<build>

<finalName>ssm</finalName>

<pluginManagement>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

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

<version>3.2</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

<encoding>UTF-8</encoding>

<showWarnings>true</showWarnings>

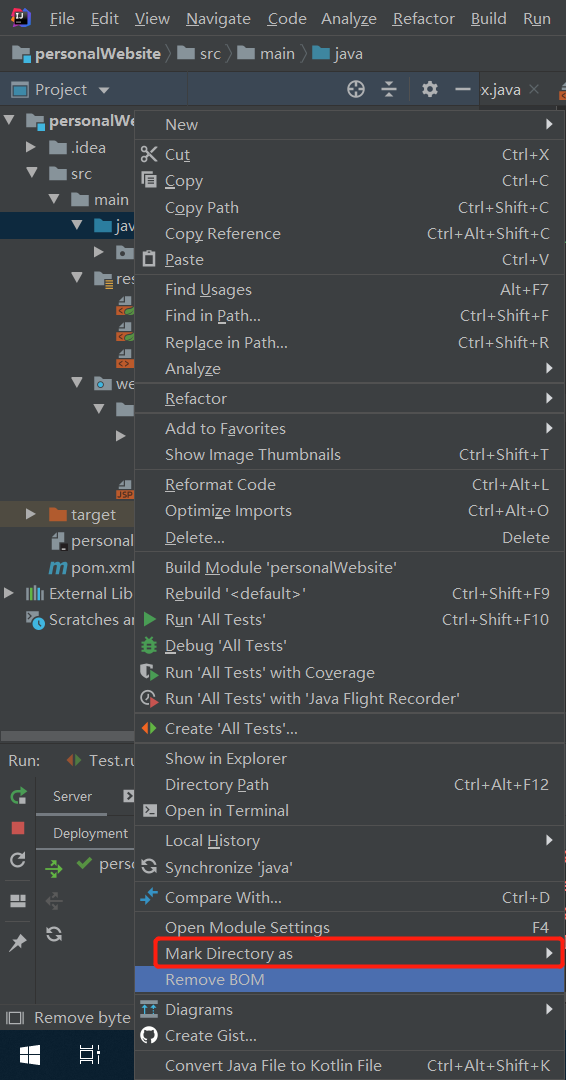
</configuration>

</plugin>

</plugins>

</pluginManagement>

</build>

**c、若无webapp文件夹，需在main中创建webapp文件夹，并在File-Project Structure-Facets-“+”-web把webapp设为源目录；  
d、若无java、resourse，需要自行创建，并且把它们设成对应的根目录：** 

e、编写实体类【在之后创建domain时，用于封装数据库传过来的数据】,要implements Serializable

需要有个序列号seriaVersionUID：用于保持版本的兼容性，在版本升级的时候反序列化 仍保持对象的唯一性

f、编写dao接口：用于之后实现与mybatis的连接，数据库操作，如基于注解的SQL

g、编写service接口以及实现类,要在实现类中加@Service(“【实现的类名】”)

### 第二步：Spring框架代码的编写

1. **在resourse目录下创建Spring Config文件applicationContext.xml配置文件，编写配置信息：**

<?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:aop="http://www.springframework.org/schema/aop"  
 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/aop  
 http://www.springframework.org/schema/aop/spring-aop.xsd  
 http://www.springframework.org/schema/tx  
 http://www.springframework.org/schema/tx/spring-tx.xsd">  
  
 <!--开启注解的扫描，希望处理service和dao，controller不需要Spring框架去处理-->  
 <context:component-scan base-package="cn.shawve" >  
 <!--配置哪些注解不扫描-->  
 <context:exclude-filter type="annotation" expression="org.springframework.stereotype.Controller" />  
 </context:component-scan>  
</beans>

1. **添加测试类：在与dao、controller等文件夹同级的文件夹下创建测试：**

public class TestSpring {  
  
 @Test  
 public void run1(){  
 // 加载配置文件

ApplicationContext ac = new ClassPathXmlApplicationContext(“classpath:applicationContext.xml”);  
 // 获取对象  
 AccountService as = (AccountService) ac.getBean("accountService");  
 // 调用方法  
 as.findAll();  
 }  
}

### 第三步：Spring整合SpringMVC框架

**1、搭建和测试SpringMVC的开发环境：**

·在web.xml中配置DispatcherServlet前端控制器：【会报红，正常现象】

<!--配置前端控制器，服务器必须加载，并加载springmvc.xml配置文件-->  
<servlet>  
 <servlet-name>dispatcherServlet</servlet-name>  
 <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  
 <!--配置初始化参数，加载springmvc.xml配置文件-->  
 <init-param>  
 <param-name>contextConfigLocation</param-name>  
 <param-value>classpath:springmvc.xml</param-value>  
 </init-param>  
 <!--启动服务器，自动创建该servlet-->  
 <load-on-startup>1</load-on-startup>  
</servlet>  
<servlet-mapping>  
 <servlet-name>dispatcherServlet</servlet-name>  
 <url-pattern>/</url-pattern>  
</servlet-mapping>

<!--解决中文乱码的过滤器[post请求传参若是中文，处理可能出现的乱码问题]-->  
<filter>  
 <filter-name>characterEncodingFilter</filter-name>  
<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>  
 <init-param>  
 <param-name>encoding</param-name>  
 <param-value>UTF-8</param-value>  
 </init-param>  
</filter>  
<filter-mapping>  
 <filter-name>characterEncodingFilter</filter-name>  
 <url-pattern>/\*</url-pattern>  
</filter-mapping>

**·在resourse目录下创建springmvc.xml配置文件，编写配置文件信息：**

<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:mvc="http://www.springframework.org/schema/mvc" xmlns:context="http://www.springframework.org/schema/context"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="  
 http://www.springframework.org/schema/beans  
 http://www.springframework.org/schema/beans/spring-beans.xsd  
 http://www.springframework.org/schema/mvc  
 http://www.springframework.org/schema/mvc/spring-mvc.xsd  
 http://www.springframework.org/schema/context  
 http://www.springframework.org/schema/context/spring-context.xsd">  
  
 <!--开启注解扫描，只扫描Controller注解-->  
 <context:component-scan base-package="cn.shawve">  
 <context:include-filter type="annotation" expression="org.springframework.stereotype.Controller" />  
 </context:component-scan>  
  
 <!--配置的视图解析器对象-->  
 <bean id="internalResourceViewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">  
 <property name="prefix" value="/WEB-INF/pages/"/>  
 <property name="suffix" value=".jsp"/>  
 </bean>  
  
 <!--在webapp下不过滤的静态资源路径-->  
 <mvc:resources location="/css/" mapping="/css/\*\*" />  
 <mvc:resources location="/images/" mapping="/images/\*\*" />  
 <mvc:resources location="/js/" mapping="/js/\*\*" />  
 <mvc:default-servlet-handler/>  
 <!--开启SpringMVC注解的支持-->  
 <mvc:annotation-driven/>  
</beans>

·测试SpringMVC是否搭建成功：  
 注意：index.jsp要放在webapp目录下，与WEB-INF同级

1. 编写index.jsp和list.jsp：

<a href = “account/findAll”> 查询所有</a>

1. 编写AccountController类：

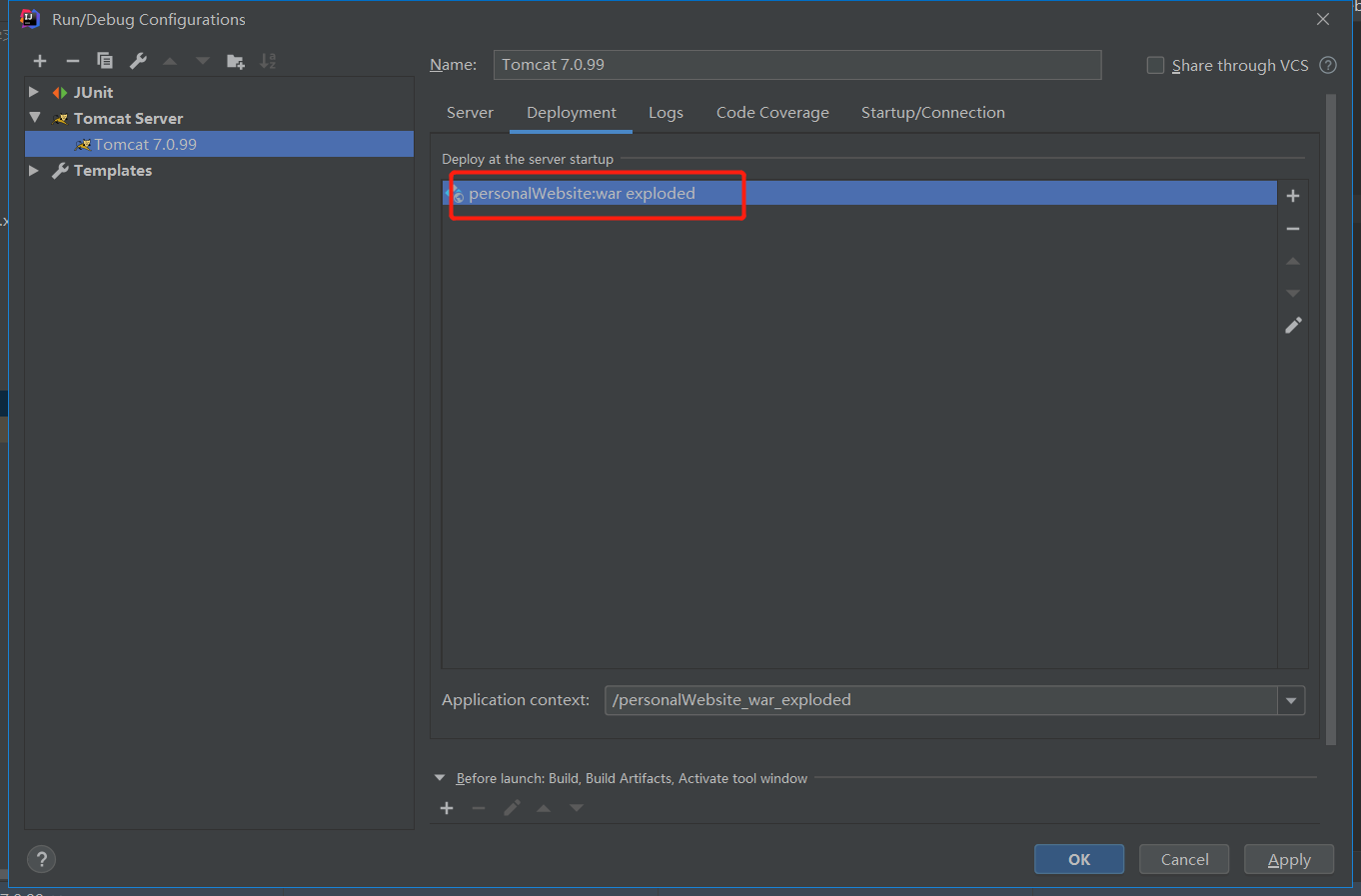
/\*\*  
 \* 查看所有商品  
 \* @param model  
 \* @return  
 \*/

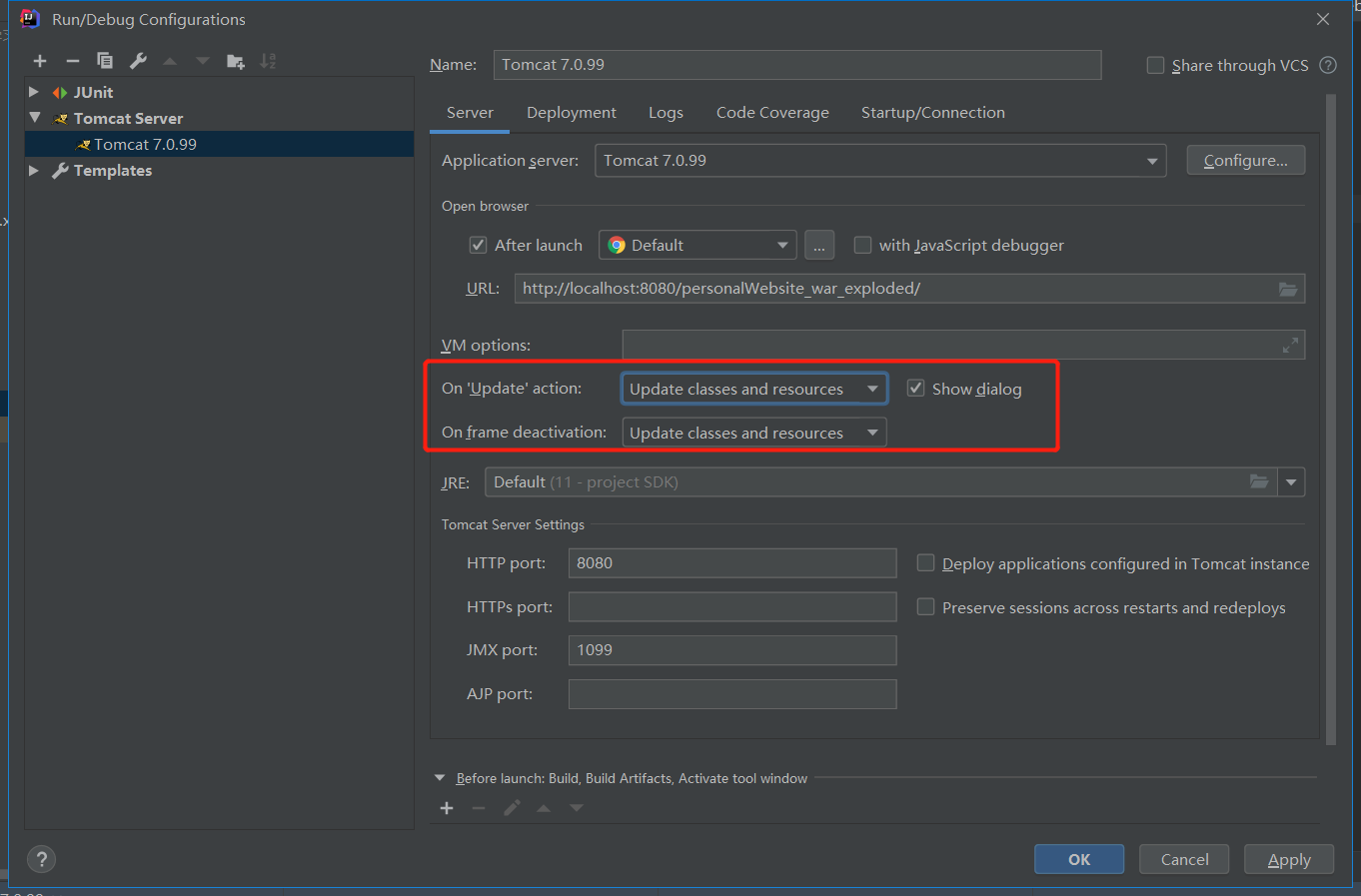
@RequestMapping(“Account”)

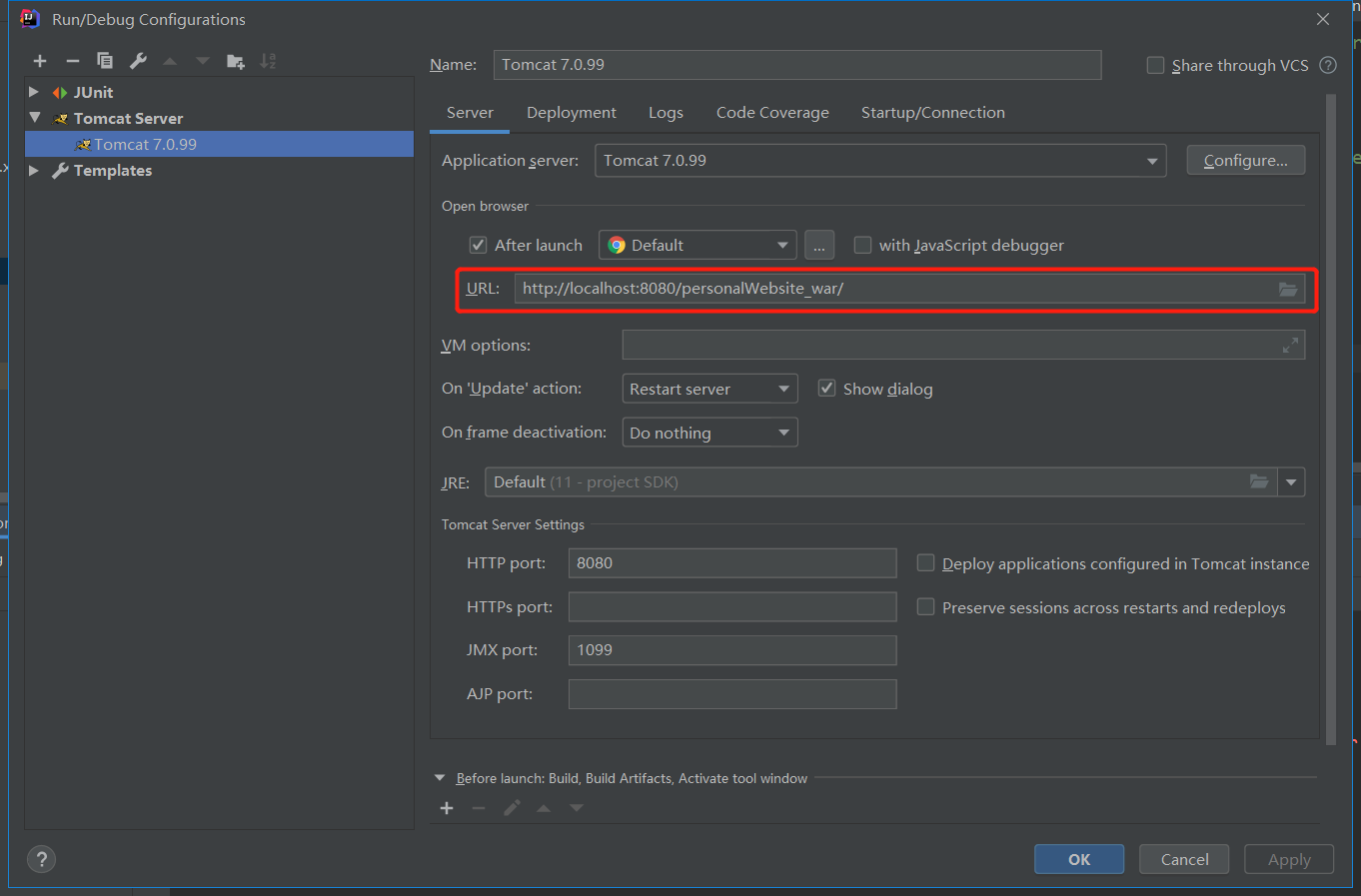
public class AccountController{  
 @RequestMapping("/findAll")  
 public String findAll(){

System.pritln.out(“执行成功”)  
 return "list.jsp";  
 }

}

1. 添加项目至tomcat服务器：  
   a、添加时选择exploded:  
   

b、为了使jsp样式修改之后直接f5刷新就可以显示修改，而不是重新启动服务器  


c、红框部分不要乱动，默认即可  


1. 运行tomcat服务器进行测试  
    5）若出现404可能是因为index.jsp不在与WEB-INF同级目录下
2. **Spring整合SpringMVC框架，在web.xml文件中配置：**

1）目的：在controller中能成功调用service对象中的方法

2）在项目启动的时候，就去加载applicationContext.xml的配置文件，在web,xml 中配置ContextLoaderListener监听器（由于该监听器只能加载WEB-INF目录下的 applicationContext.xml的配置文件，但是此时的applicationContext.xml文件在 resourse根目录下，所以需要设置配置文件的路径）

!--配置Spring的监听器，默认只加载WEB-INF目录下的applicationContext.xml配置文件-->  
<listener>  
 <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  
</listener>  
<!--设置配置文件的路径-->  
<context-param>  
 <param-name>contextConfigLocation</param-name>  
 <param-value>classpath:applicationContext.xml</param-value>  
</context-param>  
<context-param>  
 <param-name/>  
 <param-value/>  
</context-param>

1. 测试：在controller中用@Autowired来注入service对象，并调用它

@Controller  
@RequestMapping("/account")  
public class AccountController {  
  
 @Autowired  
 private AccountService accountService;  
  
 /\*\*  
 \* 查看所有商品  
 \* @param model  
 \* @return  
 \*/  
 @RequestMapping("/findAll")  
 public String findAll(Model model){  
 // 调用service的方法  
 List<Account> list = accountService.findAll();  
 model.addAttribute("list",list);  
 return "shopOprator/shopList";  
 }

}

### 第四步：Spring整合MyBatis框架

1. **在Resource目录下创建SqlMapConfig.xml配置文件，编写核心配置文件：**

<?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">  
<!--mybatis的主配置文件-->  
<configuration>  
 <!--配置环境-->  
 <environments default="mysql">  
 <!-- 配置mysql环境-->  
 <environment id="mysql">  
 <!-- 配置事务类型-->  
 <transactionManager type="JDBC"></transactionManager>  
 <!-- 配置数据源（连接池）-->  
 <dataSource type="POOLED">  
 <!-- 配置连接数据库的基本信息-->  
 <property name="driver" value="com.mysql.jdbc.Driver"/>  
 <property name="url" value="jdbc:mysql://localhost:3306/shop"/>   
 <property name="username" value="root"/>  
 <property name="password" value="root"/>  
 </dataSource>  
 </environment>  
 </environments>  
  
 <!-- 指定映射配置文件的位置，映射配置文件指的是每个dao独立的配置文件  
 如果使用注解来配置的话，此处应该使用class属性指定被注解的dao全限定类名-->  
 <mappers>  
 <mapper class="cn.shawve.dao.AccountDao"/> //注意这个路径需要修改  
 </mappers>  
</configuration>

1. **在dao文件下，如AccountDao中，方法前用对应的注解(Select、Insert等)编写SQL语句【注意在接口前添加注解@Repository】：**

// 查询所有商品  
@Select("select \* from shopList")  
public List<Account> findAll();  
  
// 增加商品  
@Insert("insert into shopList (shopName,price) values (#{shopName},#{price})")  
public void saveAccount(Account account);  
  
//删除商品  
@Delete("delete from `shopList` where `shoplist`.shopId = #{shopId}")  
public void delectAccount(int shopId);

1. **在test目录下创建测试类：**

/\*\*  
 \* 测试查询  
 \* @throws Exception  
 \*/  
@Test  
public void run1() throws Exception {  
 // 加载配置文件  
 InputStream in = Resources.getResourceAsStream("SqlMapConfig.xml");  
 // 创建SqlSessionFactory对象  
 SqlSessionFactory factory = new SqlSessionFactoryBuilder().build(in);  
 // 创建SqlSession对象  
 SqlSession session = factory.openSession();  
 // 获取到代理对象  
 AccountDao dao = session.getMapper(AccountDao.class);  
 // 查询所有数据  
 List<Account> list = dao.findAll();  
 for(Account account : list){  
 System.out.println(account);  
 }  
 // 关闭资源  
 session.close();  
 in.close();  
}

1. **整合MyBatis框架：**
2. **目的：把SqlMapConfig.xml配置文件中的内容配置到applicationContext.xml配 置文件中，因而在applicationContext.xml中配置：**

<!--Spring整合MyBatis框架-->  
<!--配置连接池-->  
<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">  
 <property name="driverClass" value="com.mysql.jdbc.Driver"/>  
 <property name="jdbcUrl" value="jdbc:mysql://localhost:3306/shop"/>  
 <property name="user" value="root"/>  
 <property name="password" value="root"/>  
</bean>  
  
<!--配置SqlSessionFactory工厂-->  
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">  
 <property name="dataSource" ref="dataSource" />  
</bean>  
  
<!--配置Dao接口所在包-->  
<bean id="mapperScanner" class="org.mybatis.spring.mapper.MapperScannerConfigurer">  
 <property name="basePackage" value="cn.shawve.dao"/>  
</bean>

1. **在AccountDao接口中添加@Repository注解至接口头部**
2. **在service中用@Autowire注入dao对象，进行测试**
3. **在applicationContext.xml中配置Spring的声明式事务管理：**

<!--配置Spring框架声明式事务管理-->  
<!--配置事务管理器-->  
<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">  
 <property name="dataSource" ref="dataSource" />  
</bean>  
  
<!--配置事务通知-->  
<tx:advice id="txAdvice" transaction-manager="transactionManager">  
 <tx:attributes>  
 <tx:method name="find\*" read-only="true"/>  
 <tx:method name="\*" isolation="DEFAULT"/>  
 </tx:attributes>  
</tx:advice>  
  
<!--配置AOP增强-->  
<aop:config>  
 <aop:advisor advice-ref="txAdvice" pointcut="execution(\* cn.shawve.service.impl.\*ServiceImpl.\*(..))"/>  
</aop:config>

**完毕！！！！**

**一些小细节：**

1. **文件位置是有要求的，不能乱写**
2. **基于注解开发时，各种注解必须写，**

**·如@Service（“<实现的类>”）**

**·@Autowired写在在引用dao、service类时**

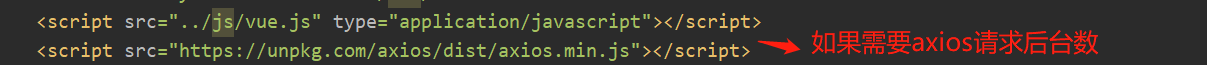
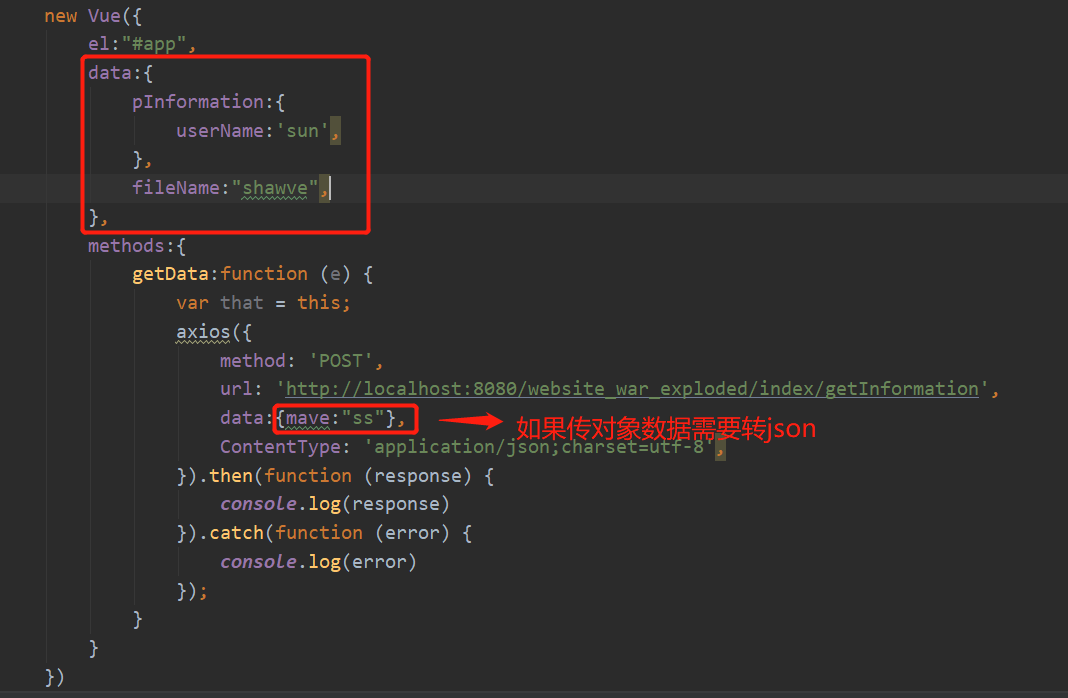
**·@Controller\@RequestMapping写在Controller类上**

**·@Repository写在dao上**

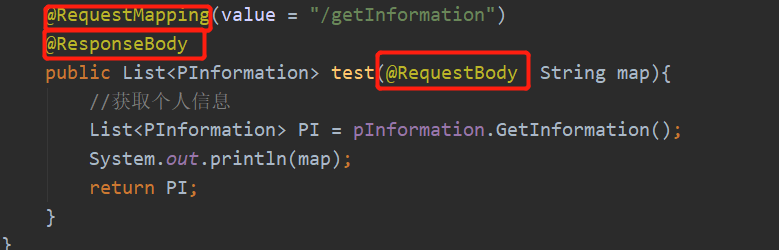
1. **注意jsp标注的完整性：isELIgnored=”false”不写将无数据传入**

<%@ **page** contentType="text/html;charset=UTF-8" language="java" isELIgnored="false" %>  
<%@ **taglib** prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

1. **若是tomcat输出台乱码：则可参照此网站：**

<https://www.cnblogs.com/shej123/p/10312806.html>  
5、引入vue的操作，直接script src即可  
  
6、关于前后端数据传递：  
 前端：  
 

后端：



ps：a、RequestMapping可以有@RequestMapping(method=”POST/GET”,value=”/\*\*\*”)的形式， 分别对应POST和GET请求，但是，如果没有写method=”POST/GET”,则可以接受GET 和POST请求

b、网页数据传输一般默认是GET，即使方法是POST请求，即【RequesMapping就好了】

c、若后端返回数据是对象，则需要配置jackson，它可以自动把对象转换成json格式

7、配置jackson:

**pom.xml:**

dependency>  
 <groupId>javax.annotation</groupId>  
 <artifactId>jsr250-api</artifactId>  
 <version>1.0</version>  
</dependency>  
<dependency>  
 <groupId>com.fasterxml.jackson.core</groupId>  
 <artifactId>jackson-core</artifactId>  
 <version>2.9.8</version>  
</dependency>  
<dependency>  
 <groupId>com.fasterxml.jackson.core</groupId>  
 <artifactId>jackson-databind</artifactId>  
 <version>2.9.8</version>  
</dependency>  
<dependency>  
 <groupId>com.fasterxml.jackson.core</groupId>  
 <artifactId>jackson-annotations</artifactId>  
 <version>2.9.8</version>  
</dependency>

**springmvc:**

<!-- 避免IE执行AJAX时,返回JSON出现下载文件 -->  
<!-- Jackson 是Spring 自带的功能 -->  
<bean id="mappingJackson2HttpMessageConverter"  
 class="org.springframework.http.converter.json.MappingJackson2HttpMessageConverter">  
 <property name="supportedMediaTypes">  
 <list>  
 <value>application/json;charset=UTF-8</value>  
 <value>application/xml;charset=UTF-8</value>  
 <value>text/html;charset=UTF-8</value>  
 <value>text/plain;charset=UTF-8</value>  
 <value>text/xml;charset=UTF-8</value>  
 <value>text/javascript;charset=UTF-8</value>  
 </list>  
 </property>  
</bean>

<!--开启SpringMVC注解的支持-->  
<mvc:annotation-driven></mvc:annotation-driven>