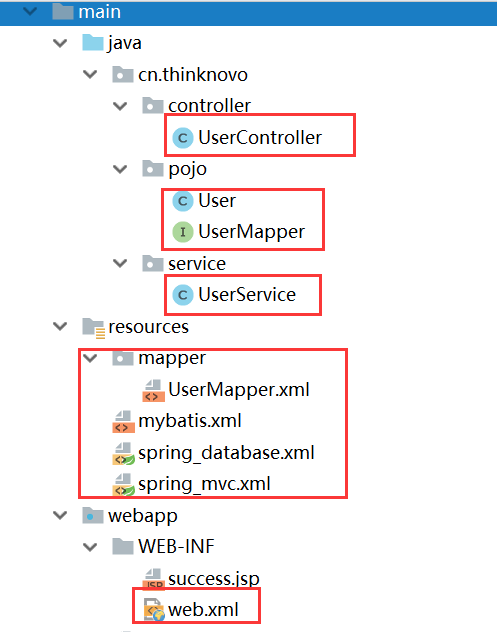
**Spring+ Spring mvc+MyBatis整合**

1. **目录结构**



(1) 修改web.xml配置文件的内容

(2) 添加mybatis、spring mvc、spring database的xml文件，放置在资源文件夹根目录，也可以在这里创建分别的目录

(3) 把mybatis的mapper.xml文件放置到资源文件夹中，不再与pojo同包

(4) 创建controller、pojo、service包，分别创建controller类、pojo对象和接口、service类

**PS：实例工程参考ssm1000，把mybatis、spring mvc、spring database的xml文件通过脚手架的导入**

1. **修改web.xml文件**

*<?***xml version="1.0" encoding="UTF-8"***?>*<**web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_4\_0.xsd"  
 version="4.0"**>  
 *<!--Spring核心监听器 -->  
 <!--在服务器启动时加载Spring容器，且只会加载一次 -->* <**context-param**>  
 <**param-name**>contextConfigLocation</**param-name**>  
 <**param-value**>classpath:spring\_database.xml</**param-value**>  
 </**context-param**>  
 <**listener**>  
 <**listener-class**>org.springframework.web.context.ContextLoaderListener</**listener-class**>  
 </**listener**>  
  
 *<!--配置springmvc -->* <**servlet**>  
 <**servlet-name**>dispatcherServlet</**servlet-name**>  
 <**servlet-class**>org.springframework.web.servlet.DispatcherServlet</**servlet-class**>  
 *<!--配置初始化参数：作用是配置SpringMVC配置文件的位置和名称 -->* <**init-param**>  
 <**param-name**>contextConfigLocation</**param-name**>  
 <**param-value**>classpath:spring\_mvc.xml</**param-value**>  
 </**init-param**>  
 <**load-on-startup**>1</**load-on-startup**>  
 </**servlet**>  
  
 <**servlet-mapping**>  
 <**servlet-name**>dispatcherServlet</**servlet-name**>  
 <**url-pattern**>/</**url-pattern**>  
 </**servlet-mapping**>  
  
 *<!--配置由Spring 提供的针对中文乱码的编码过滤器 -->  
 <!-- 编码过滤器 -->* <**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**>  
</**web-app**>

1. **导入spring-mvc.xml文件**

*<?***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.xsd"**>  
  
 *<!--1.开启Springioc 自动扫描注解包 只对mvc的包做扫描 -->* <**context:component-scan base-package="cn.thinknovo.controller"**/>  
  
 *<!--2. 开启注解，对jackson数据支持 -->* <**mvc:annotation-driven**/>  
   
 *<!--3.配置视图解析器 -->   
 <!-- 配置视图解析器:如何把handle方法返回值解析为实际的物理视图  
 name="prefix"代表前面的目录，为空代表根目录 name="suffix"代表返回的地址为后缀为.JSP的文件 -->* <**bean class="org.springframework.web.servlet.view.InternalResourceViewResolver"**>  
 <**property name="prefix" value="/WEB-INF/"** />  
 <**property name="suffix" value=".jsp"** />  
 </**bean**>  
</**beans**>

**PS：这里的scan扫描需要单独说明一下，为什么只扫描了mvc的包：**

在使用注解处理的时候，需要注意一点，web服务器启动的时候，spring-mvc.xml会优先于spring.xml启动，如果工程整包被扫描，那么service会优先于事务先启动，这样就导致事务无法注入管理，所以这里只能扫描mvc包，然后在spring.xml文件中扫描service包，这样就是先启动spring的事务管理注入，再注入service，最后注入controller，这个时候，事务才进入了spring 的IOC管理。**结论就是：为了避免这种问题出现，需要在spring-mvc.xml中只扫描controller的包，而在spring.xml的配置中只扫描service的包**

是否被spring的IOC管理，可以在调用批量增、删、改的时候，通过后台打印查看，没有管理的提示是：

JDBC Connection [com.mysql.jdbc.JDBC4Connection@7103e78b] will not be managed by Spring

并且每操作一次数据库，都会创建一个新的session，所以当某一条数据出错时，不会回滚。

如果纳入管理的提示是：

JDBC Connection [com.mysql.jdbc.JDBC4Connection@29fd1532] will be managed by Spring

并且每次操作不会创建新的session，所以当某一条数据出错时，会回滚所有提交内容。

还有两种情况需要在配置的时候注意：

(1)如果只在spring的配置扫描所有包，mvc将无法请求对应的controller；

(2)如果在mvc中配置了controller的扫描，在spring的配置中扫描所有包，会出现controller被创建两次；

**PS：事务需要在spring.xml文件进行配置，后面的spring.xml文件中会注释说明**

1. **导入spring-database.xml文件**

*<?***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"  
 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/mvc  
 http://www.springframework.org/schema/mvc/spring-mvc.xsd  
 http://www.springframework.org/schema/tx  
 http://www.springframework.org/schema/tx/spring-tx.xsd"**>  
  
 *<!--1. 配置数据源:c3p0 -->* <**bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource"**>  
 <**property name="driverClassName" value="com.mysql.jdbc.Driver"** />  
 <**property name="url" value="jdbc:mysql://localhost:3306/hiber"** />  
 <**property name="username" value="root"** />  
 <**property name="password" value="thinknovo"** />  
 </**bean**>  
   
 *<!--2. 配置mybatis的SqlSession的工厂: SqlSessionFactoryBean dataSource:引用数据源 -->* <**bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean"**>  
 *<!--dataSource属性指定要用到的连接池-->* <**property name="dataSource" ref="dataSource"** />  
 *<!--configLocation属性指定mybatis的核心配置文件-->* <**property name="configLocation" value="classpath:mybatis.xml"** />  
 *<!-- 自动扫描mapping.xml文件 -->* <**property name="mapperLocations" value="classpath:mapper/\*.xml"**/>  
 </**bean**>  
   
 *<!--3. 自动扫描mybatis中pojo接口包，  
 并且自动注入第二项创建的sessionfactory，初始化所有的mapper接口对象 -->* <**bean class="org.mybatis.spring.mapper.MapperScannerConfigurer"**>  
 <**property name="basePackage" value="cn.thinknovo.pojo"** />  
 </**bean**>  
  
 *<!--4. 扫描service注解 -->* <**context:component-scan base-package="cn.thinknovo.service"**/>  
  
 *<!--5. 配置事务管理器 -->* <**bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager"**>  
 <**property name="dataSource" ref="dataSource"** />  
 </**bean**>  
  
 *<!--6. 开启注解进行事务管理 transaction-manager：引用上面定义的事务管理器-->* <**tx:annotation-driven transaction-manager="transactionManager"** />  
   
</**beans**>

1. **导入mybatis.xml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"*>***<**configuration**>  
 <**settings**>  
 *<!--如果value为true的时候，当返回内容为空时，那么返回的是一个空的对象，而不是null  
 如果这里的value为false(默认值)，mybatis在没有查询到内容的时候返回null-->* <**setting name="returnInstanceForEmptyRow" value="true"**/>  
 *<!--显示sql语句，此行配置如果删除，则不在后台显示SQL语句-->* <**setting name="logImpl" value="STDOUT\_LOGGING"**/>  
 </**settings**>  
  
 <**mappers**>  
 *<!-- 自动扫描包内的Mapper接口与配置文件  
 接口文件名要与mapper文件名相同  
 纳入spring管理之后，不再需要扫描-->  
 <!--<package name="cn/thinknovo/pojo"/>-->* </**mappers**>  
</**configuration**>

**PS：引入mybatis.xml只是为了添加辅助参数配置项，比如显示sql或者控制返回空值等，如果项目中不需要这些配置，直接可以删除掉此文件**

1. **创建pojo对象**

**public class** User {  
 **private int id**;  
 **private** String **username**;  
 **private** String **password**;  
  
 **public int** getId() {  
 **return id**;  
 }  
  
 **public void** setId(**int** id) {  
 **this**.**id** = id;  
 }  
  
 **public** String getUsername() {  
 **return username**;  
 }  
  
 **public void** setUsername(String username) {  
 **this**.**username** = username;  
 }  
  
 **public** String getPassword() {  
 **return password**;  
 }  
  
 **public void** setPassword(String password) {  
 **this**.**password** = password;  
 }  
}

**PS：这里的pojo对象需要数据库对应的表结构。**

1. **创建mapper.xml**

<**mapper namespace="cn.thinknovo.pojo.UserMapper"**>  
 <**select id="selectUser" resultType="cn.thinknovo.pojo.User"**>  
 select *\** from user;  
 </**select**>  
  
 <**update id="updateUser" parameterType="cn.thinknovo.pojo.User"**>  
 update user set **username**=#{username} where **id**=#{id};  
 </**update**>  
  
 <**insert id="insertUser" parameterType="cn.thinknovo.pojo.User"**>  
 insert into user values(null,#{username},#{password});  
 </**insert**>  
</**mapper**>

1. **创建mapper接口**

**public interface** UserMapper {  
 List<User> selectUser();  
 **void** updateUser(User user);  
 **void** insertUser(User user);  
}

1. **创建service**

@Service  
**public class** UserService {  
  
 **public** UserService() {  
 System.***out***.println(**"--------------------------UserService构造函数"**);  
 }  
  
 @Autowired  
 **private** UserMapper **userMapper**;  
  
 **public** UserMapper getUserMapper() {  
 **return userMapper**;  
 }  
  
 **public** List<User> selectUser() {  
 **return userMapper**.selectUser();  
 }  
  
 **public void** updateUser(User user) {  
 **userMapper**.updateUser(user);  
 }  
  
 */\*  
 在@Transactional注解中如果不配置rollbackFor属性,那么事务只在RuntimeException的时候才会回滚,  
 加上rollbackFor=Exception.class,可以让事务在遇到任意异常时都回滚  
 \*/* @Transactional(rollbackFor = Exception.**class**)  
 **public void** insertUser(List<User> userList) **throws** Exception {  
 **for** (User user : userList) {  
 **userMapper**.insertUser(user);  
 }  
 }  
}

1. **创建controller**

@Controller  
**public class** UserController {  
  
 **public** UserController() {  
 System.***out***.println(**"--------------------------UserAction构造函数"**);  
 }  
  
 @Autowired  
 **private** UserService **userService**;  
  
 @RequestMapping(value = **"/selectUser"**, method = RequestMethod.***GET***)  
 **public** String selectUser(ModelMap modelMap) {  
 System.***out***.println(**"UserAction -------------------------- selectUser"**);  
  
 List<User> list = **new** ArrayList<>();  
 User user = **new** User();  
 user.setUsername(**"kkk"**);  
 user.setPassword(**"3330"**);  
 list.add(user);  
  
 user = **new** User();  
 user.setUsername(**"jjj"**);  
 user.setPassword(**"4440"**);  
 list.add(user);  
  
 user = **new** User();  
 user.setUsername(**"aaa"**);  
 user.setPassword(**"5550"**);  
 list.add(user);  
  
 **try** {  
 **userService**.insertUser(list);  
 } **catch** (Exception e) {  
 System.***out***.println(**"---------------------------------------------------sql出错"**);  
 e.printStackTrace();  
 }  
   
*// userService.updateUser(user);* modelMap.addAttribute(**"list\_user"**, **"en~~~"**);  
 **return "success"**;  
 }  
}

现在可以通过controller的访问路径在浏览器输入进行测试了。基于事务问题，可以批量处理下增、删、改，看看事务的自动管理，单条记录是没有事务可言的，因为sql出错，不能写入，但是批量处理的时候，如果前面的插入了，后面出错，就应该全部回滚。