# SSM整合&Maven聚合工程

## 需求

整合SSM三个框架，实现对用户数据的CRUD

学习目标：

1. spring和Mybatis的整合
2. spring和springMVC的整合
3. 使用SpringMVC+Mybatis实现数据库的CRUD

## 使用技术

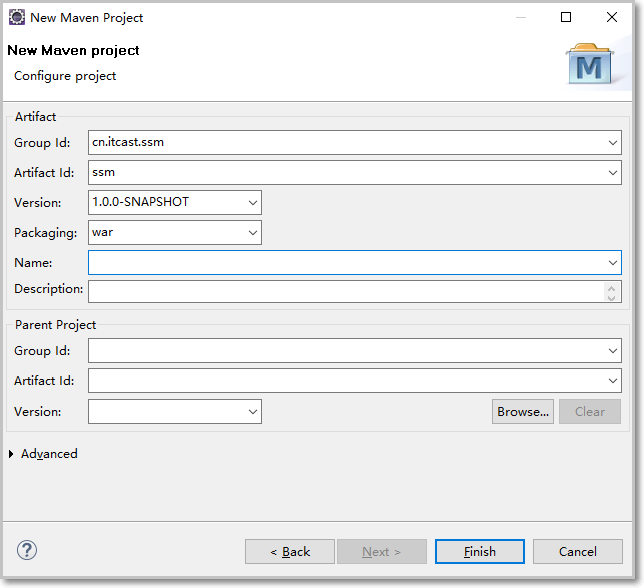
Spring + SpringMVC + Mybatis

数据库：mysql+easyui

# 创建工程---环境整理

## 创建工程

创建war工程并生成web.xml:



## 继承itcast-parent

继承itcast-parent并且在pom.xml中引入所需依赖：

|  |
| --- |
| <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/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <groupId>cn.itcast.ssm</groupId>  <artifactId>ssm</artifactId>  <version>1.0.0-SNAPSHOT</version>  <packaging>war</packaging>  <parent>  <groupId>cn.itcast.parent</groupId>  <artifactId>itcast-parent</artifactId>  <version>0.0.1-SNAPSHOT</version>  </parent>  <dependencies>  <!-- 单元测试 -->  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <scope>test</scope>  </dependency>  <!-- Spring -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-webmvc</artifactId>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-jdbc</artifactId>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-aspects</artifactId>  </dependency>  <!-- Mybatis -->  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis</artifactId>  </dependency>  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis-spring</artifactId>  </dependency>  <!-- MySql -->  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  </dependency>  <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-log4j12</artifactId>  </dependency>  <!-- Jackson Json处理工具包 -->  <dependency>  <groupId>com.fasterxml.jackson.core</groupId>  <artifactId>jackson-databind</artifactId>  </dependency>  <!-- 连接池 -->  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid</artifactId>  </dependency>  <!-- JSP相关 -->  <dependency>  <groupId>jstl</groupId>  <artifactId>jstl</artifactId>  </dependency>  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>servlet-api</artifactId>  <scope>provided</scope>  </dependency>  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>jsp-api</artifactId>  <scope>provided</scope>  </dependency>  </dependencies>  <build>  <plugins>  <!-- 配置Tomcat插件 -->  <plugin>  <groupId>org.apache.tomcat.maven</groupId>  <artifactId>tomcat7-maven-plugin</artifactId>  <configuration>  <port>8080</port>  <path>/</path>  </configuration>  </plugin>  </plugins>  </build>  </project> |

# Mybatis独立测试

使用mybatis进行CRUD,为了操作方便，都会先创建一个pojo对象用来封装数据

## 步骤一：创建pojo对象

由于使用的数据库表还是之前mybatis中的表，因此只要对照着tb\_user创建pojo对象即可：

**public** **class** User **implements** Serializable{

**private** Long id;

// 用户名

**private** String userName;

// 密码

**private** String password;

// 姓名

**private** String name;

// 年龄

**private** Integer age;

// 性别，1男性，2女性

**private** Integer sex;

// 出生日期

**private** Date birthday;

// 创建时间

**private** Date created;

// 更新时间

**private** Date updated;

**public** Long getId() {

**return** id;

}

**public** **void** setId(Long 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;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** Integer getAge() {

**return** age;

}

**public** **void** setAge(Integer age) {

**this**.age = age;

}

**public** Integer getSex() {

**return** sex;

}

**public** **void** setSex(Integer sex) {

**this**.sex = sex;

}

**public** Date getBirthday() {

**return** birthday;

}

**public** **void** setBirthday(Date birthday) {

**this**.birthday = birthday;

}

**public** Date getCreated() {

**return** created;

}

**public** **void** setCreated(Date created) {

**this**.created = created;

}

**public** Date getUpdated() {

**return** updated;

}

**public** **void** setUpdated(Date updated) {

**this**.updated = updated;

}

@Override

**public** String toString() {

**return** "User [id=" + id + ", userName=" + userName + ", password=" + password + ", name=" + name

+ ", age=" + age + ", sex=" + sex + ", birthday=" + birthday + ", created=" + created

+ ", updated=" + updated + "]";

}

}

## 步骤二：创建mapper接口



内容如下:

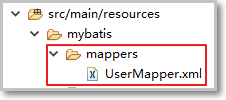
**public** **interface** UserMapper {

User queryUserById(@Param("id")Long id);

}

## 步骤三：创建UserMapper.xml

在resources下创建一个mybatis目录，在mybatis目录下再创建一个mappers目录：专门用来管理映射文件的



注意需要修改的地方：

<?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标签:配置各类映射 namespace：名称空间，由于映射文件有多个，为了防止crud语句的唯一标识被重复，可以设置空间名称。 -->

<mapper namespace=*"cn.itcast.ssm.mapper.UserMapper"*>

<select id=*"queryUserById"* resultType=*"User"*>

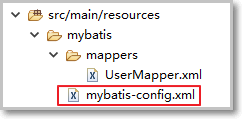
select \* from tb\_user where id = #{id}

</select>

</mapper>

## 步骤四：创建mybatis-config.xml

再resources中的mybatis目录中创建mybatis-config.xml配置文件：



mybatis-config.xml:注意需要修改的地方

<?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>

<!-- 引入外部资源文件 resource:默认引入classpath路径下的资源文件 -->

<properties resource=*"jdbc.properties"*></properties>

<settings>

<!-- 设置驼峰参数 -->

<setting name=*"mapUnderscoreToCamelCase"* value=*"true"* />

</settings>

<!-- 配置别名 -->

<typeAliases>

<!-- typeAlias:用来配置别名，方便映射文件使用，type:类的全限定类名,alias:别名 -->

<!-- <typeAlias type="cn.itcast.pojo.User" alias="User"/> -->

<!-- 配置包扫描：解决多个pojo别名问题，默认将类名作为笔名，大小写不敏感 -->

<package name=*"cn.itcast.pojo"*/>

</typeAliases>

<!-- 配置环境：可以配置多个环境，default：配置某一个环境的唯一标识，表示默认使用哪个环境 -->

<environments default=*"development"*>

<!-- 配置环境,id:环境的唯一标识 -->

<environment id=*"development"*>

<!-- 事务管理器，type:使用jdbc的事务管理器 -->

<transactionManager type=*"JDBC"* />

<!-- 数据源，type:池类型的数据源 -->

<dataSource type=*"POOLED"*>

<!-- 配置连接信息 -->

<property name=*"driver"* value=*"${jdbc.driverClass}"* />

<property name=*"url"* value=*"${jdbc.url}"* />

<property name=*"username"* value=*"${jdbc.username}"* />

<property name=*"password"* value=*"${jdbc.password}"* />

</dataSource>

</environment>

</environments>

<!-- 配置映射文件：用来配置sql语句和结果集类型等 -->

<mappers>

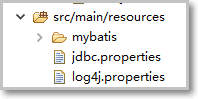
<mapper resource=*"mybatis/mappers/UserMapper.xml"*/>

</mappers>

</configuration>

## 步骤五：添加properties

在resources中添加jdbc.propertie和log4j的资源文件：



jdbc.properties内容如下:

jdbc.driverClass=com.mysql.jdbc.Driver

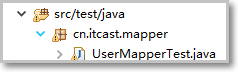
jdbc.url=jdbc:mysql://localhost:3306/mybatis

jdbc.username=root

jdbc.password=root

## 步骤六：测试

在测试包中创建一个测试类进行测试：



**public** **class** UserMapperTest {

@Test

**public** **void** testQueryUserById() **throws** IOException {

String resource ="mybatis/mybatis-config.xml";

InputStream inputStream = Resources.*getResourceAsStream*(resource);

SqlSessionFactory sqlSessionFactory = **new** SqlSessionFactoryBuilder().build(inputStream);

SqlSession sqlSession = sqlSessionFactory.openSession(**true**);

UserMapper userMapper = sqlSession.getMapper(UserMapper.**class**);

User user = userMapper.queryUserById(1l);

System.***out***.println(user);

}

}

# Spring和mybatis整合

Mybatis可以操作数据库，其实就是通过SqlSessionFactory来获取sqlSession，然后通过sqlSession来操作数据库，而之前的SqlSessionFactory都是我们手动初始化出来的。而所谓spring整合mybatis,其实就是将SqlSessionFactory交给spring的IOC容器来进行初始化话。

Mybatis和spring的整合包提供了一个SqlSessionFactoryBean，该对象是一个工厂bean,实现了FactoryBean,可以通过其getObject方法来返回一个SqlSessionFactory。

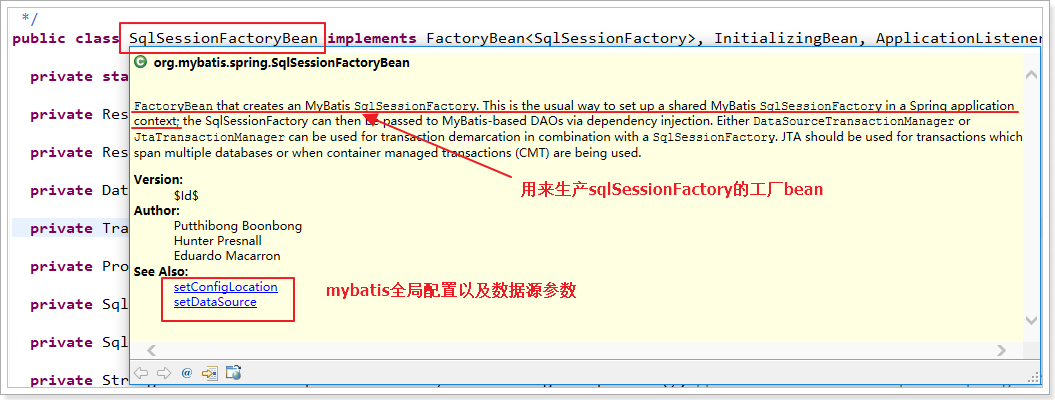
Mybatis和spring的整合依赖：

<dependency>

<groupId>org.mybatis</groupId>

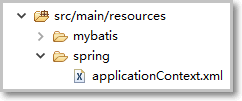
<artifactId>mybatis-spring</artifactId>

</dependency>



## 步骤一：配置applicationContext.xml

创建配置文件：



通过SqlSessionFactoryBean初始化SqlSessionFactory

<?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"*>

<!-- 加载资源文件 -->

<context:property-placeholder location=*"classpath:jdbc.properties"*/>

<!-- 装配sqlSessionFactory -->

<bean id=*"sqlSessionFactory"* class=*"org.mybatis.spring.SqlSessionFactoryBean"*>

<!-- 注入mybatis的核心配置文件 -->

<property name=*"configLocation"* value=*"classpath:mybatis/mybatis-config.xml"*></property>

<!-- 注入mybatis的数据源 -->

<property name=*"dataSource"* ref=*"dataSource"*></property>

<!-- 配置别名扫描 -->

<property name=*"typeAliasesPackage"* value=*"cn.itcast.pojo"*></property>

<!-- 配置映射文件管理：mybatis/mappers目录下的所有的映射文件 -->

<property name=*"mapperLocations"* value=*"classpath:mybatis/mappers/\*\*/\*.xml"*></property>

</bean>

<!-- 装配数据源 -->

<bean id=*"dataSource"* class=*"com.alibaba.druid.pool.DruidDataSource"*>

<property name=*"driverClassName"* value=*"${jdbc.driverClass}"*></property>

<property name=*"url"* value=*"${jdbc.url}"*></property>

<property name=*"username"* value=*"${jdbc.username}"*></property>

<property name=*"password"* value=*"${jdbc.password}"*></property>

</bean>

</beans>

## 步骤二：修改mybatis-config.xml

由于数据源已经交由spring管理，因此在该配置文件中的数据源就可以不用配置了，删除即可。

<?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>

<!-- 开启驼峰匹配 -->

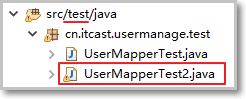
<setting name=*"mapUnderscoreToCamelCase"* value=*"true"*/>

</settings>

</configuration>

## ※步骤三：测试

再次创建一个测试方法：



**public** **class** UserMapperTest2 {

@Test

**public** **void** testQueryUserById() {

ApplicationContext ac = **new** ClassPathXmlApplicationContext("spring/applicationContext.xml");

SqlSessionFactory sqlSessionFactory = (SqlSessionFactory) ac.getBean("sqlSessionFactory");

SqlSession sqlSession = sqlSessionFactory.openSession(**true**);

UserMapper userMapper = sqlSession.getMapper(UserMapper.**class**);

User user = userMapper.queryUserById(1l);

System.***out***.println(user);

}

}

## 整合Mapper动态代理类

### 装配单个mapper

我们在进行测试的时候还是需要自己从容器中获取SqlSessionFactory，然后再通过SqlSessionFactory获取SqlSession,最后再通过SqlSession获取UserMapper，这样操作还是有点麻烦。

如果能将一个UserMapper装配到sping容器中，那么我们是不是就可以直接从容器中获取UserMapper并进行操作了呢？

如果SqlSessionFactory可以交给spring管理，那mapper接口行不行呢？

Spring提供了一个叫做MapperFactoryBean，专门用来管理mapper。其实底层还是通过sqlSession.getMapper(Xxx.class);来获取一个mapper的代理对象。

通过MapperFactoryBean将UserMapper装配到spring容器中:可参照资料中的Mybatis-Spring-1.2.2中的index.html中的第二章入门

而MapperFactoryBean实际上还是最终通过SqlSessionFactory获取SqlSession,最后再通过SqlSession获取UserMapper的

修改applicationContext.xml配置如下：

<?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"*

xmlns:aop=*"http://www.springframework.org/schema/aop"*

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-4.0.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*

*http://www.springframework.org/schema/aop*

*http://www.springframework.org/schema/aop/spring-aop.xsd"*>

<bean id=*"sqlSessionFactory"* class=*"org.mybatis.spring.SqlSessionFactoryBean"*>

<property name=*"configLocation"* value=*"classpath:mybatis/mybatis-config.xml"*></property>

<property name=*"dataSource"* ref=*"dataSource"*></property>

</bean>

<bean id=*"dataSource"* class=*"com.alibaba.druid.pool.DruidDataSource"*>

<property name=*"driverClassName"* value=*"${jdbc.driverClass}"*></property>

<property name=*"url"* value=*"${jdbc.url}"*></property>

<property name=*"username"* value=*"${jdbc.username}"*></property>

<property name=*"password"* value=*"${jdbc.password}"*></property>

</bean>

<!--

location：加载资源文件

-->

<context:property-placeholder location=*"classpath:jdbc.properties"*/>

<bean id=*"userMapper"* class=*"org.mybatis.spring.mapper.MapperFactoryBean"*>

<property name=*"mapperInterface"* value=*"cn.itcast.mapper.UserMapper"*></property>

<property name=*"sqlSessionFactory"* ref=*"sqlSessionFactory"*></property>

</bean>

</beans>

可以将userMapper的获取方式修改如下：

@Test

**public** **void** testQueryUserById() {

/\*ApplicationContext ac = new ClassPathXmlApplicationContext("spring/applicationContext.xml");

SqlSessionFactory sqlSessionFactory = (SqlSessionFactory) ac.getBean("sqlSessionFactory");

SqlSession sqlSession = sqlSessionFactory.openSession(true);

UserMapper userMapper = sqlSession.getMapper(UserMapper.class);

User user = userMapper.queryUserById(1l);

System.out.println(user);\*/

ApplicationContext ac = **new** ClassPathXmlApplicationContext("spring/applicationContext.xml");

UserMapper userMapper = (UserMapper) ac.getBean("userMapper");

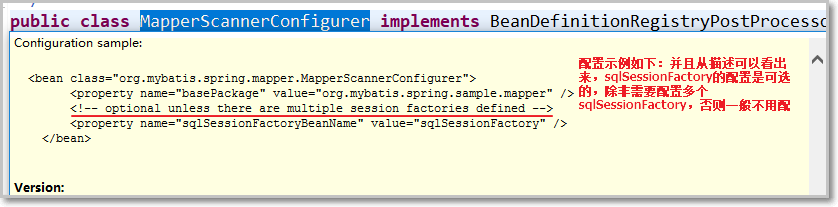
User user = userMapper.queryUserById(1l);

System.***out***.println(user);

}

### 装配多个mapper

如果整个工程中有多个mapper接口，那么如果每一个接口都配置一个MapperFactoryBean，显然太麻烦，不太合理，因此在spring和mybatis的整合包中还提供了一个MapperScannerConfigurer对象,该对象中提供了通过配置包路径来扫描所有mapper接口的方式。



从以上注释，我们可以将mapper的配置优化如下：

<?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"*

xmlns:aop=*"http://www.springframework.org/schema/aop"*

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-4.0.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*

*http://www.springframework.org/schema/aop*

*http://www.springframework.org/schema/aop/spring-aop.xsd"*>

<bean id=*"sqlSessionFactory"* class=*"org.mybatis.spring.SqlSessionFactoryBean"*>

<property name=*"configLocation"* value=*"classpath:mybatis/mybatis-config.xml"*></property>

<property name=*"dataSource"* ref=*"dataSource"*></property>

</bean>

<bean id=*"dataSource"* class=*"com.alibaba.druid.pool.DruidDataSource"*>

<property name=*"driverClassName"* value=*"${jdbc.driverClass}"*></property>

<property name=*"url"* value=*"${jdbc.url}"*></property>

<property name=*"username"* value=*"${jdbc.username}"*></property>

<property name=*"password"* value=*"${jdbc.password}"*></property>

</bean>

<!--

location：加载资源文件

-->

<context:property-placeholder location=*"classpath:jdbc.properties"*/>

<!-- 装配userMapper到spring容器中 -->

<!-- <bean id="userMapper" class="org.mybatis.spring.mapper.MapperFactoryBean">

<property name="mapperInterface" value="cn.itcast.mapper.UserMapper"></property>

<property name="sqlSessionFactory" ref="sqlSessionFactory"></property>

</bean> -->

<!-- 配置mapper扫描：可以一次性扫描多个mapper接口，由于id时唯一的，不能代表多个接口类型，因此不要id -->

<bean class=*"org.mybatis.spring.mapper.MapperScannerConfigurer"*>

<!-- 配置mapper接口所在包路径 -->

<property name=*"basePackage"* value=*"cn.itcast.mapper"*></property>

</bean>

</beans>

### 执行测试

在测试时，可以通过对象的字节码来直接获取UserMapper：

@Test

**public** **void** testQueryUserById() {

/\*ApplicationContext ac = new ClassPathXmlApplicationContext("spring/applicationContext.xml");

SqlSessionFactory sqlSessionFactory = (SqlSessionFactory) ac.getBean("sqlSessionFactory");

SqlSession sqlSession = sqlSessionFactory.openSession(true);

UserMapper userMapper = sqlSession.getMapper(UserMapper.class);

User user = userMapper.queryUserById(1l);

System.out.println(user);\*/

/\*ApplicationContext ac = new ClassPathXmlApplicationContext("spring/applicationContext.xml");

UserMapper userMapper = (UserMapper) ac.getBean("userMapper");

User user = userMapper.queryUserById(1l);

System.out.println(user);\*/

ApplicationContext ac = **new** ClassPathXmlApplicationContext("spring/applicationContext.xml");

UserMapper userMapper = ac.getBean(UserMapper.**class**);

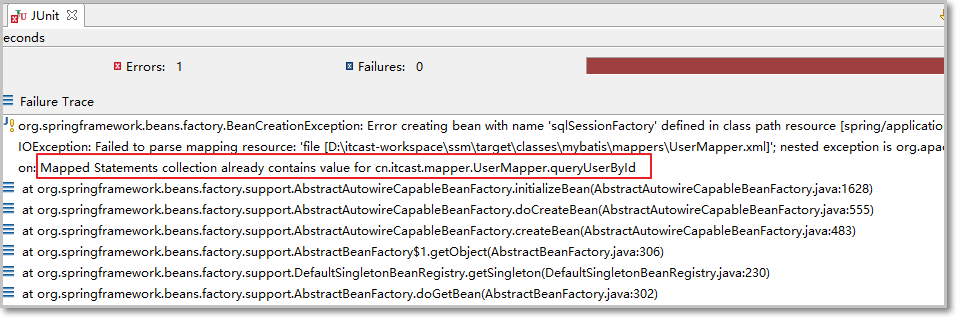
User user = userMapper.queryUserById(1l);

System.***out***.println(user);

}

## 常见错误

报错如下:



因为虽然将管理映射文件在applicationContext.xml中进行了配置，但是没有将mybatis-config.xml中的管理映射文件的配置删除，管理了两个相同的映射文件导致，因此错误提示说已经存在一个一个叫queryUserById的statements。

## 最终配置文件

### applicationContext.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: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"*>

<!-- 加载资源文件 -->

<context:property-placeholder location=*"classpath:jdbc.properties"*/>

<!-- 装配sqlSessionFactory -->

<bean id=*"sqlSessionFactory"* class=*"org.mybatis.spring.SqlSessionFactoryBean"*>

<!-- 注入mybatis的核心配置文件 -->

<property name=*"configLocation"* value=*"classpath:mybatis/mybatis-config.xml"*></property>

<!-- 注入mybatis的数据源 -->

<property name=*"dataSource"* ref=*"dataSource"*></property>

<!-- 配置别名扫描 -->

<property name=*"typeAliasesPackage"* value=*"cn.itcast.pojo"*></property>

<!-- 配置映射文件管理 -->

<property name=*"mapperLocations"* value=*"classpath:mybatis/mappers/\*\*/\*.xml"*></property>

</bean>

<!-- 装配数据源 -->

<bean id=*"dataSource"* class=*"com.alibaba.druid.pool.DruidDataSource"*>

<property name=*"driverClassName"* value=*"${jdbc.driverClass}"*></property>

<property name=*"url"* value=*"${jdbc.url}"*></property>

<property name=*"username"* value=*"${jdbc.username}"*></property>

<property name=*"password"* value=*"${jdbc.password}"*></property>

</bean>

<!-- 使用mapper的包扫描，一次性装配多个mapper的动态代理类 -->

<bean class=*"org.mybatis.spring.mapper.MapperScannerConfigurer"*>

<property name=*"basePackage"* value=*"cn.itcast.mapper"*></property>

</bean>

</beans>

### 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>

<settings>

<!-- 开启驼峰匹配 -->

<setting name=*"mapUnderscoreToCamelCase"* value=*"true"*/>

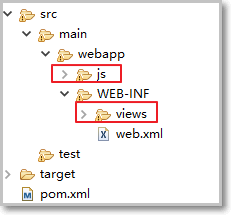
</settings>

</configuration>

# Spring和springmvc整合

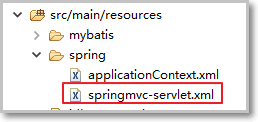
## 数据准备

将资料中的js和静态资源复制到工程中：



## Spring和springmvc整合

### 步骤一：创建springmvc-servlet.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/mvc*

*http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd*

*http://www.springframework.org/schema/context*

*http://www.springframework.org/schema/context/spring-context.xsd"*>

</beans>

### 步骤二：配置web.xml

在web.xml中需要配置post请求乱码过滤器、spring监听器、springmvc的控制器。

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns=*"http://java.sun.com/xml/ns/javaee"*

xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*

version=*"2.5"*>

<display-name>ssm</display-name>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

<welcome-file>index.htm</welcome-file>

<welcome-file>index.jsp</welcome-file>

<welcome-file>default.html</welcome-file>

<welcome-file>default.htm</welcome-file>

<welcome-file>default.jsp</welcome-file>

</welcome-file-list>

<!-- 编码过滤器，以UTF8编码 -->

<filter>

<filter-name>encodingFilter</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>UTF8</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>encodingFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<!--Spring的监听器 -->

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:spring/applicationContext.xml</param-value>

</context-param>

<!-- 配置SpringMVC -->

<servlet>

<servlet-name>springmvc</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:spring/springmvc-servlet.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>springmvc</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

</web-app>

## 页面显示

需求：显示users.jsp页面

### 步骤一：创建controller

@Controller

@RequestMapping("user")

**public** **class** UserController {

@RequestMapping("users")

**public** String toUsers(){

**return** "users";

}

}

### 步骤二：配置springmvc-servlet.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/mvc*

*http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd*

*http://www.springframework.org/schema/context*

*http://www.springframework.org/schema/context/spring-context.xsd"*>

<!-- 开启注解扫描 -->

<context:component-scan base-package=*"cn.itcast.web"*></context:component-scan>

<!-- 配置注解驱动，替代推荐使用的映射器以及适配器，json转换器 -->

<mvc:annotation-driven />

<!-- 配置视图解析器 -->

<bean class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

<property name=*"prefix"* value=*"/WEB-INF/views/"*></property>

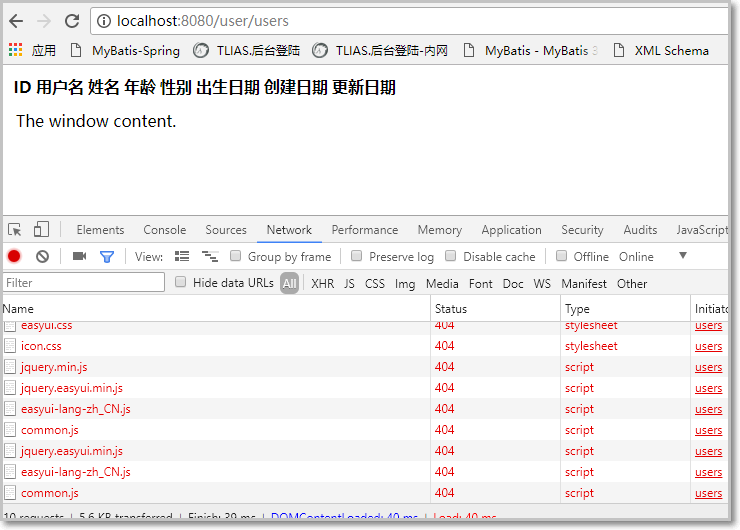
<property name=*"suffix"* value=*".jsp"*></property>

</bean>

</beans>

### 步骤三：测试访问

发现页面显示后没有任何样式：



原因:静态资源都被拦截掉了。

解决方案：在springmvc-servlet.xml中配置静态资源放行。

<?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/mvc*

*http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd*

*http://www.springframework.org/schema/context*

*http://www.springframework.org/schema/context/spring-context.xsd"*>

<!-- 开启注解扫描 -->

<context:component-scan base-package=*"cn.itcast.web"*></context:component-scan>

<!-- 配置注解驱动，替代推荐使用的映射器以及适配器，json转换器 -->

<mvc:annotation-driven />

<!-- 配置视图解析器 -->

<bean class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

<property name=*"prefix"* value=*"/WEB-INF/views/"*></property>

<property name=*"suffix"* value=*".jsp"*></property>

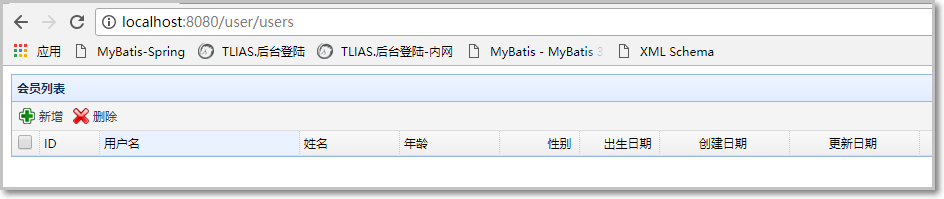
</bean>

<!-- 放行静态资源 -->

<mvc:default-servlet-handler/>

</beans>

再次测试：

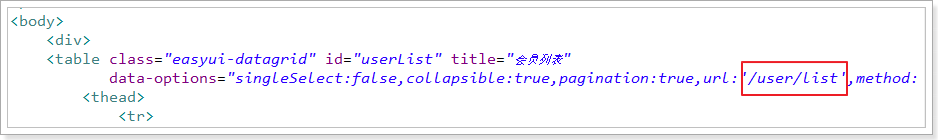


## 查询所有数据

分析：要查询用户信息，首先搞清楚以下几个问题

1. 请求路径是什么
2. 请求路径对应的Controller方法的返回值，应该是什么样的

Jsp页面获取数据的请求路径，参见jsp页面中的url：



### 步骤一：编写controller

@Controller

@RequestMapping("user")

**public** **class** UserController {

@Autowired

**private** IUserService userService;

@RequestMapping("users")

**public** String toUsers(){

**return** "users";

}

@RequestMapping("list")

@ResponseBody

**public** List<User> queryUserList(){

List<User> users = userService.queryUserList();

**return** users;

}

}

### 步骤二：编写service

Service接口：

**public** **interface** IUserService {

List<User> queryUserList();

}

Service实现类:

@Service

**public** **class** UserServiceImpl **implements** IUserService{

@Autowired

**private** UserMapper userMapper;

@Override

**public** List<User> queryUserList() {

**return** userMapper.queryUserList();

}

}

### 步骤三：编写userMapper和映射文件

userMapper接口：

**public** **interface** UserMapper {

User queryUserById(Long id);

List<User> queryUserList();

}

UserMapper映射文件:

<?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">

<!-- 用来编写各类声明和自定义映射等配置 namespace属性:名称空间，用来区分映射文件的 -->

<mapper namespace=*"cn.itcast.mapper.UserMapper"*>

<select id=*"queryUserById"* resultType=*"User"*>

select \* from tb\_user where id = #{id}

</select>

<select id=*"queryUserList"* resultType=*"User"*>

select \* from tb\_user

</select>

</mapper>

### 步骤四：spring扫描service层

<?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"* xmlns:aop=*"http://www.springframework.org/schema/aop"*

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-4.0.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*

*http://www.springframework.org/schema/aop*

*http://www.springframework.org/schema/aop/spring-aop.xsd"*>

<!-- 扫描service -->

<context:component-scan base-package=*"cn.itcast.service"*></context:component-scan>

<bean id=*"sqlSessionFactory"* class=*"org.mybatis.spring.SqlSessionFactoryBean"*>

<property name=*"configLocation"* value=*"classpath:mybatis/mybatis-config.xml"*></property>

<property name=*"dataSource"* ref=*"dataSource"*></property>

<!-- mapperLocations:配置映射文件 value：配置映射文件路径，只要是在根目录下的mybatis中的mappers目录下的任意多级目录（0个或多个）下的任意一个映射文件都进行扫描 -->

<property name=*"mapperLocations"* value=*"classpath:mybatis/mappers/\*\*/\*.xml"*></property>

<!-- 配置别名扫描的包 -->

<property name=*"typeAliasesPackage"* value=*"cn.itcast.pojo"*></property>

</bean>

<bean id=*"dataSource"* class=*"com.alibaba.druid.pool.DruidDataSource"*>

<property name=*"driverClassName"* value=*"${jdbc.driverClass}"*></property>

<property name=*"url"* value=*"${jdbc.url}"*></property>

<property name=*"username"* value=*"${jdbc.username}"*></property>

<property name=*"password"* value=*"${jdbc.password}"*></property>

</bean>

<!-- location：加载资源文件 -->

<context:property-placeholder location=*"classpath:jdbc.properties"* />

<!-- 配置mapper扫描：可以一次性扫描多个mapper接口 -->

<bean class=*"org.mybatis.spring.mapper.MapperScannerConfigurer"*>

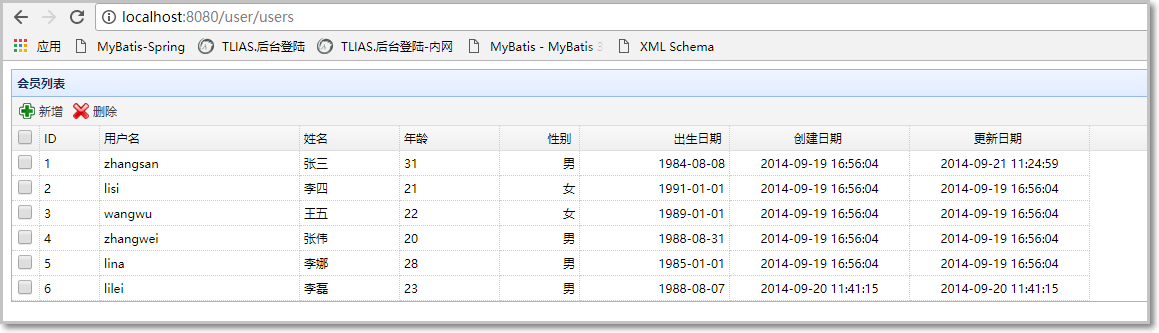
<!-- 配置mapper接口所在包路径 -->

<property name=*"basePackage"* value=*"cn.itcast.mapper"*></property>

</bean>

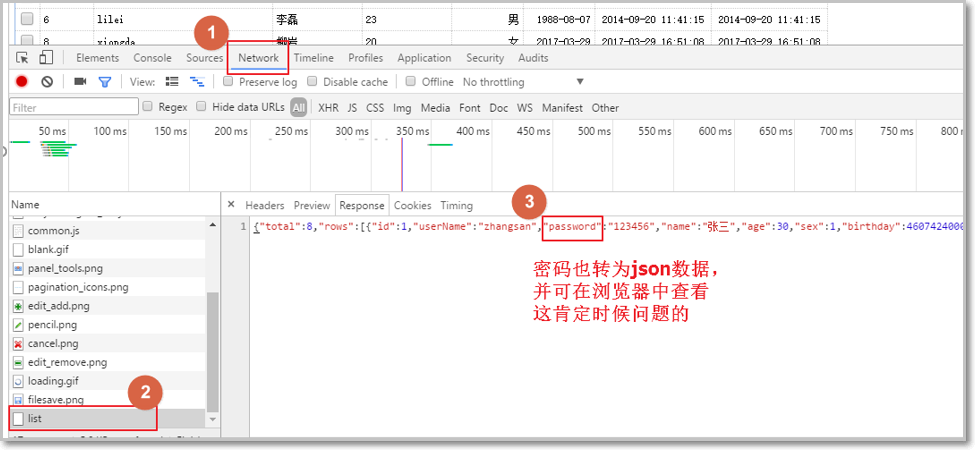
</beans>

### 步骤五：测试



### ※解决密码问题

问题：



解决方案：在实体类（user）中的对应属性上，添加@JsonIgnore注解



## Service层整合事务管理

为了保证数据的一致性及原子性，Service的增删改操作应该添加事务。

需求：添加一个新增两个用户的方法，测试事务是否添加成功。

### 步骤一：service接口和实现类

在UserService接口中添加方法：

/\*\*

\* 测试事务

\* **@param** user1

\* **@param** user2

\*/

**public** **void** addUsers(User user1, User user2);

service实现类：

@Override

**public** **void** addUsers(User user1, User user2) {

//新增用户一

userMapper.addUser(user1);

//新增用户二

userMapper.addUser(user2);

}

### 步骤二：mapper接口和映射文件

UserMapper接口:

**public** **interface** UserMapper {

User queryUserById(Long id);

List<User> queryUserList();

**void** addUser(User user);

}

UserMapper映射文件：

<insert id=*"addUser"*>

INSERT INTO tb\_user (

user\_name ,

password ,

name ,

age ,

sex ,

birthday ,

created ,

updated

)

VALUES

(

#{userName},

#{password},

#{name},

#{age},

#{sex},

#{birthday},

now(),

now()

);

</insert>

### 步骤三：配置事务

<?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"* xmlns:aop=*"http://www.springframework.org/schema/aop"*

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-4.0.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*

*http://www.springframework.org/schema/aop*

*http://www.springframework.org/schema/aop/spring-aop.xsd"*>

<!-- 扫描service -->

<context:component-scan base-package=*"cn.itcast.service"*></context:component-scan>

<bean id=*"sqlSessionFactory"* class=*"org.mybatis.spring.SqlSessionFactoryBean"*>

<property name=*"configLocation"* value=*"classpath:mybatis/mybatis-config.xml"*></property>

<property name=*"dataSource"* ref=*"dataSource"*></property>

<!-- mapperLocations:配置映射文件 value：配置映射文件路径，只要是在根目录下的mybatis中的mappers目录下的任意多级目录（0个或多个）下的任意一个映射文件都进行扫描 -->

<property name=*"mapperLocations"* value=*"classpath:mybatis/mappers/\*\*/\*.xml"*></property>

<!-- 配置别名扫描的包 -->

<property name=*"typeAliasesPackage"* value=*"cn.itcast.pojo"*></property>

</bean>

<bean id=*"dataSource"* class=*"com.alibaba.druid.pool.DruidDataSource"*>

<property name=*"driverClassName"* value=*"${jdbc.driverClass}"*></property>

<property name=*"url"* value=*"${jdbc.url}"*></property>

<property name=*"username"* value=*"${jdbc.username}"*></property>

<property name=*"password"* value=*"${jdbc.password}"*></property>

</bean>

<!-- location：加载资源文件 -->

<context:property-placeholder location=*"classpath:jdbc.properties"* />

<!-- 配置mapper扫描：可以一次性扫描多个mapper接口 -->

<bean class=*"org.mybatis.spring.mapper.MapperScannerConfigurer"*>

<!-- 配置mapper接口所在包路径 -->

<property name=*"basePackage"* value=*"cn.itcast.mapper"*></property>

</bean>

<bean id=*"transactionManager"* class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>

<property name=*"dataSource"* ref=*"dataSource"*></property>

</bean>

<tx:advice id=*"txAdvice"* transaction-manager=*"transactionManager"*>

<tx:attributes>

<tx:method name=*"\*"*/>

</tx:attributes>

</tx:advice>

<aop:config>

<aop:pointcut expression=*"execution(\* cn.itcast.service.impl.\*.\*(..))"* id=*"pt1"*/>

<aop:advisor advice-ref=*"txAdvice"* pointcut-ref=*"pt1"*/>

</aop:config>

</beans>

### 步骤四：测试

@Test

**public** **void** testAddUsers(){

ApplicationContext ac = **new** ClassPathXmlApplicationContext("spring/applicationContext.xml");

IUserService userService = (IUserService) ac.getBean("userServiceImpl");

User user1 = **new** User();

user1.setName("用户1");

user1.setPassword("123456");

user1.setUserName("admin1");

user1.setAge(18);

user1.setSex(1);

user1.setBirthday(**new** Date());

User user2 = **new** User();

user2.setName("用户2");

user2.setPassword("123456");

user2.setUserName("admin2");

user2.setAge(18);

user2.setSex(1);

user2.setBirthday(**new** Date());

userService.addUsers(user1, user2);

}

### 步骤五：使用注解方式的事务

声明注解方式的事务

<!-- 声明要用注解方式的事务管理 -->

<tx:annotation-driven/>

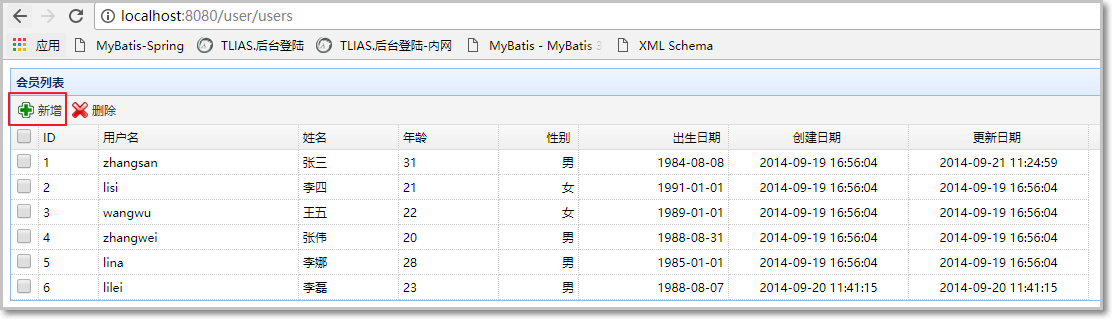
在类上使用注解:



## 新增用户页面显示

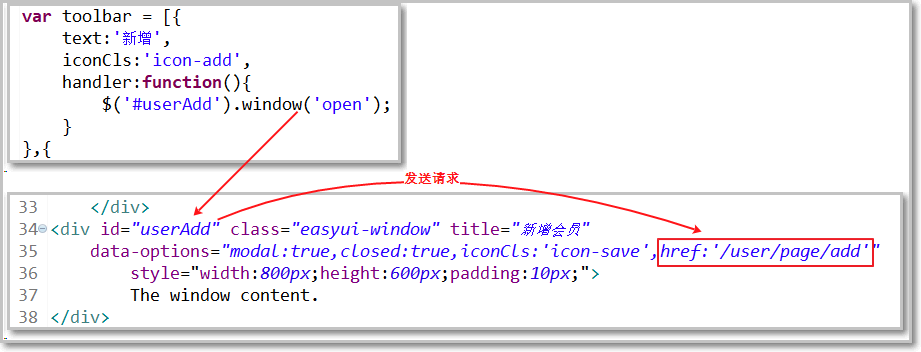
### 分析

需求：当点击新增按钮后弹出一个窗口用来新增用户数据。



分析：

1、首先分析新增按钮的请求是如何发送的？



### 步骤一：编写Controller

编写一个Controller的跳转方法，将页面进行回显：

@RequestMapping("page/add")

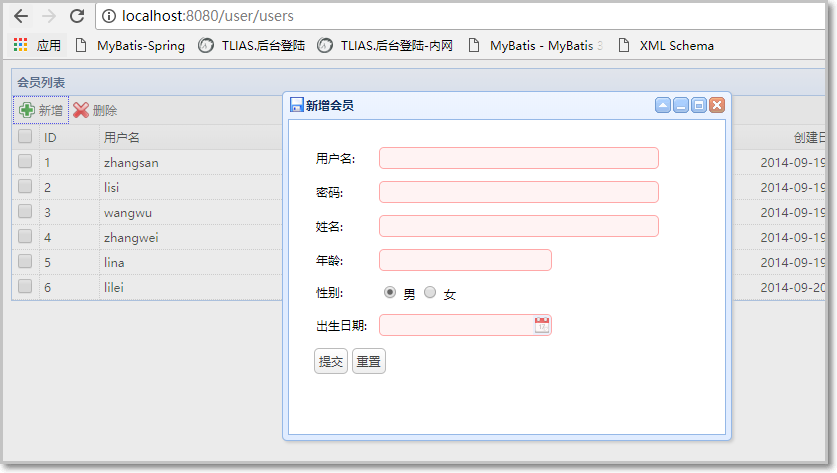
**public** String toPage() {

**return** "user-add";

}

### 步骤二：测试

测试成功：



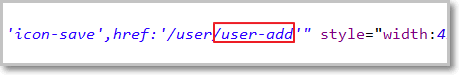
### 问题

如果有很多页面需要跳转的话，每一个页面都需要在Controller中编写一个跳转页面的方法，那就太麻烦了。

能否编写一个通用的页面跳转的方法呢？

### 步骤三：编写通用页面跳转的方法

修改页面：



将之前的方法修改如下:

@Controller

@RequestMapping("user")

**public** **class** UserController {

@Autowired

**private** IUserService userService;

/\*@RequestMapping("users")

public String toUsers(){

return "users";

}

@RequestMapping("page/add")

public String toPage() {

return "user-add";

}\*/

@RequestMapping("{pageName}")

**public** String toPage(@PathVariable("pageName") String pageName) {

**return** pageName;

}

@RequestMapping("list")

@ResponseBody

**public** List<User> queryUserList(){

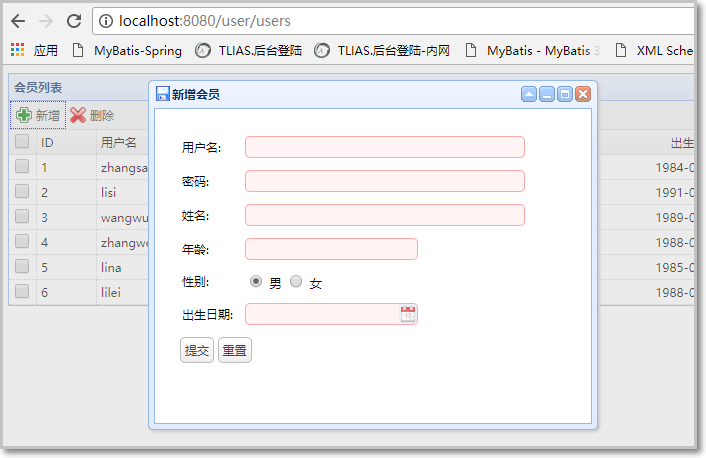
List<User> users = userService.queryUserList();

**return** users;

}

}

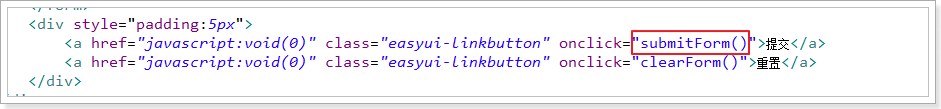
### 步骤四：测试访问



## 新增用户

### 分析

user-add.jsp中，提交事件：



触发submitForm()函数：



submitForm()函数，首先做表单验证；验证通过发送ajax请求，请求路径为：/user/save，请求参数：form表单的序列化为字符串，响应数据：{status：200}

表单验证的依据：



### 步骤一：编写UserController

将可能发生的异常捕获一下，并设置返回值的状态吗为500，使得前台页面获取500状态之后可以提示用户新增失败！

根据请求路径、响应数据以及请求参数，编写Controller方法，并对异常做处理：

@RequestMapping("save")

@ResponseBody

**public** Map<String,Integer> saveUser(User user){

Map<String,Integer> map = **new** HashMap<String,Integer>();

**try** {

Boolean flag = userService.saveUser(user);

**if** (flag) {

map.put("status", 200);

}**else** {

map.put("status", 500);

}

} **catch** (Exception e) {

map.put("status", 500);

e.printStackTrace();

}

**return** map;

}

### 步骤二：编写UserService及其实现

UserService：

**public** **interface** IUserService {

List<User> queryUserList();

**void** addUsers(User user1,User user2);

Boolean saveUser(User user);

}

UserServiceImpl：

Mybatis的statement在进行增删改的时候，都会返回一个int类型值，表示操作了几条数据.

由于我们是进行新增操作，如果成功必然是返回一条记录，我们可以判断新增操作的返回值是否为1来决定service层的新增方法的返回值是否为true.

前提：需要将UserMapper接口中的新增方法的返回值改成 int 类型。

@Override

**public** Boolean saveUser(User user) {

//设置创建实际和更新时间

**int** count = userMapper.addUser(user);

**if** (count==1) {

**return** **true**;

}

**return** **false**;

}

### 步骤三：修改UserMapper接口

将新增方法的返回值设置为int类型：

**public** **interface** UserMapper {

User queryUserById(Long id);

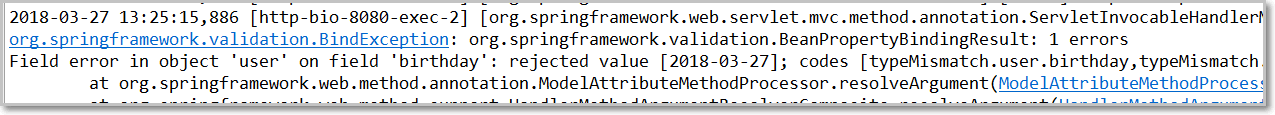
List<User> queryUserList();

**int** addUser(User user);

}

### 步骤四：测试新增

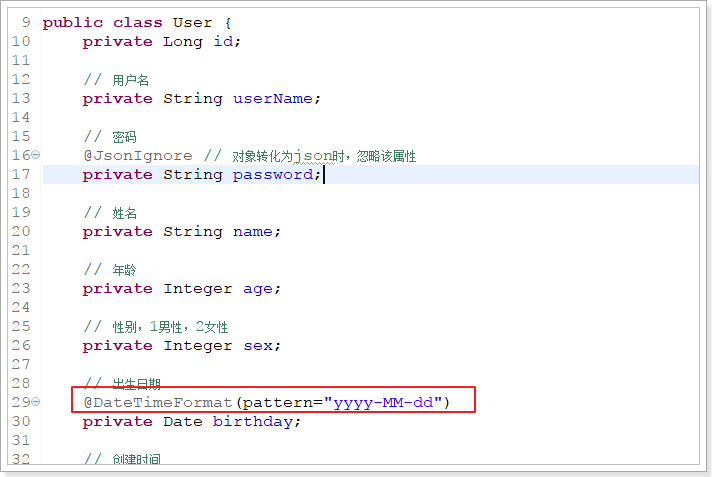
新增用户，程序写完添加用户，无法保存。控制台报异常：类型转换错误



可以使用@DateTimeFormat对接收的参数进行格式的转换。

### ※步骤五：@DateTimeFormat的使用

解决方案：



# 聚合（多模块）

## 概念

### 多模块（聚合）

对于一个大型的项目，如果我们直接作为一个工程开发，由于相互之间的依赖我们只能从头到尾由一组人开发，否则就会出现一个类好多人开发，相互更改的混乱局面，这样会容易造成各种各样的冲突，这个时候我们就将项目进行了横向和纵向的拆分。

所谓的横向的拆分就是我们平常说的三层架构，将项目分成了web层，service层、dao层（web层也被叫做表现层，service层也被叫做业务层，dao层也被持久层），可以理解为将一个功能模块的不同调用过程进行了水平方向的拆分。

所谓的纵向拆分就是将一个项目的多个功能模块进行了拆分，可以理解成为了完成一个系统，深度（纵向）分析需要有哪些功能，然后将这些功能独立出来，进行了（纵向）拆分。

拆分后，每个功能模块进行了单独的开发之后，在整合项目时就需要有一个能够整合这些项目或者模块的工程，这就是所谓聚合工程的意义。

　　建立聚合工程需要注意：

　　1.该聚合项目本身也做为一个Maven项目，它必须有自己的POM。

　　2.它的打包方式必须为： pom

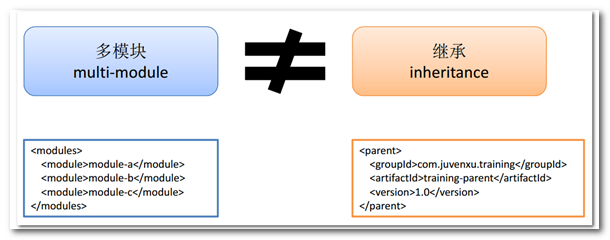
　　3.引入了新的元素：module，子模块

### 总结

　　对于聚合模块来说，它知道有哪些被聚合的模块。

　　对于继承关系的子POM来说，它必须知道自己的父POM是谁。

在一些最佳实践中我们会发现：一个POM既是聚合POM，又是父POM，这么做主要是为了方便。



多模块解决的问题： 将maven项目分解，将一个大工程分成几个小的工程便于开发，为了统一管理和开发方便。

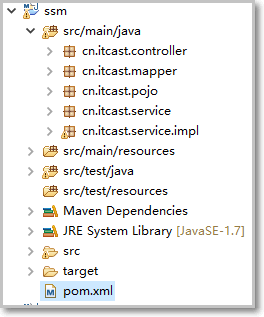
继承解决的问题 ： 使用pom配置，为了复用和整合，多个模块之间有关系，主要是指子模块和父工程之间的继承关系。例如：可以将公用的jar包放置到父工程下，子工程继承就可以使用。

提示：两者虽然概念不同，但在企业中会一起使用。

## 构建示例和项目分解

【示例】使用eclipse工具进行父子工程构建

需求: ssm工程使用maven来进行聚合和继承的重构。



更加项目的包结构，抽取成以下：

ssm\_parent:父工程（pom工程）

ssm\_pojo:存放实体（jar工程）

ssm\_mapper：存放接口（jar工程）

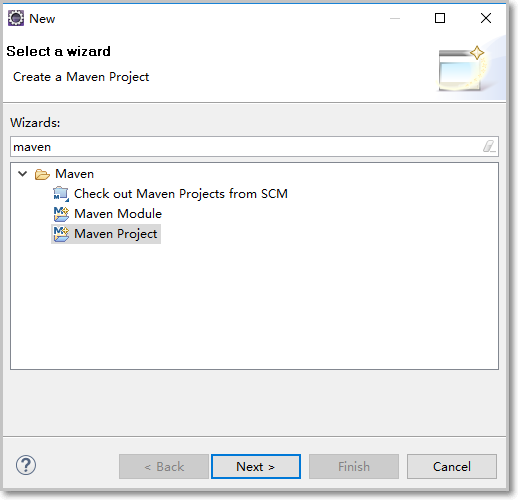
ssm\_service：存放Service（jar工程）

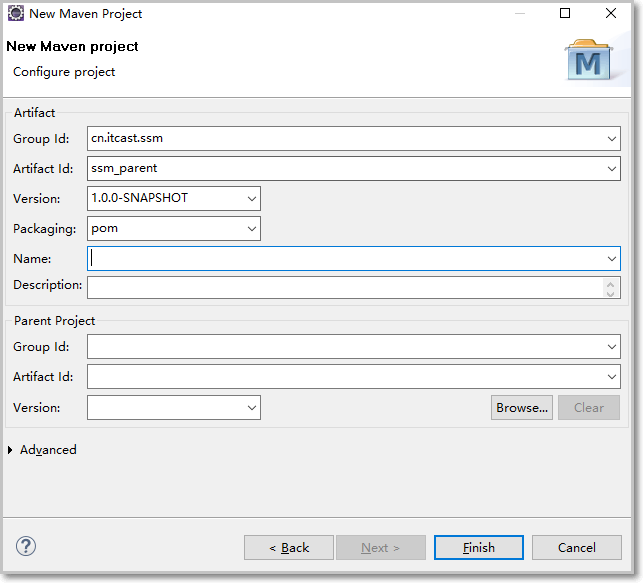
ssm\_web：存放controller（war工程）

### 步骤一：构建父工程

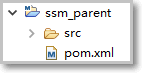
目标：构建一个父工程，多个子工程，让父工程管理子工程。

**注意:父工程打包方式必须为pom.**





生成工程如下：



### 步骤二：添加依赖

提示：父工程的作用就是用来操作子工程的，用来被继承的，最终聚合其他子工程。

一般在开发时都会统一版本和依赖，我们通常只需要复制即可。

<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/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<groupId>cn.itcast.ssm</groupId>

<artifactId>ssm\_parent</artifactId>

<version>1.0.0-SNAPSHOT</version>

<packaging>pom</packaging>

<!-- 版本管理 -->

<properties>

<junit.version>4.10</junit.version>

<spring.version>4.3.13.RELEASE</spring.version>

<mybatis.version>3.2.8</mybatis.version>

<mybatis.spring.version>1.2.2</mybatis.spring.version>

<mybatis.paginator.version>1.2.15</mybatis.paginator.version>

<mysql.version>5.1.32</mysql.version>

<slf4j.version>1.6.4</slf4j.version>

<jackson.version>2.9.0</jackson.version>

<druid.version>1.0.9</druid.version>

<httpclient.version>4.3.5</httpclient.version>

<jstl.version>1.2</jstl.version>

<servlet-api.version>2.5</servlet-api.version>

<jsp-api.version>2.0</jsp-api.version>

<joda-time.version>2.5</joda-time.version>

<commons-lang3.version>3.3.2</commons-lang3.version>

<commons-io.version>1.3.2</commons-io.version>

</properties>

<dependencies>

<!-- 单元测试 -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.9</version>

<scope>test</scope>

</dependency>

<!-- springmvc -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

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

</dependency>

<!-- spring中mybatis事务管理器 -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

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

</dependency>

<!-- spring切面 -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aspects</artifactId>

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

</dependency>

<!-- Jackson Json处理工具包 -->

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

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

</dependency>

<!-- Mybatis -->

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis</artifactId>

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

</dependency>

<!-- mybatis和spring整合 -->

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis-spring</artifactId>

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

</dependency>

<!-- 通用mapper -->

<dependency>

<groupId>com.github.abel533</groupId>

<artifactId>mapper</artifactId>

<version>2.3.4</version>

</dependency>

<!-- 分页插件 -->

<dependency>

<groupId>com.github.pagehelper</groupId>

<artifactId>pagehelper</artifactId>

<version>3.7.5</version>

</dependency>

<dependency>

<groupId>com.github.jsqlparser</groupId>

<artifactId>jsqlparser</artifactId>

<version>0.9.1</version>

</dependency>

<!-- 阿里连接池 -->

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>druid</artifactId>

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

</dependency>

<!-- 数据库 -->

<dependency>

<groupId>mysql</groupId>

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

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

</dependency>

<!-- 日志相关 -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>1.7.2</version>

</dependency>

<!-- poi报表 -->

<dependency>

<groupId>org.apache.poi</groupId>

<artifactId>poi</artifactId>

<version>3.10.1</version>

</dependency>

<!-- servlet相关 -->

<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>2.1</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

<version>1.2</version>

</dependency>

<!-- 时间操作组件 -->

<dependency>

<groupId>joda-time</groupId>

<artifactId>joda-time</artifactId>

<version>${joda-time.version}</version>

</dependency>

<!-- Apache工具组件 -->

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-lang3</artifactId>

<version>${commons-lang3.version}</version>

</dependency>

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-io</artifactId>

<version>${commons-io.version}</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

<version>3.2</version>

<configuration>

<target>1.8</target>

<source>1.8</source>

</configuration>

</plugin>

</plugins>

<!-- 受版本管理的tomcat插件，要用的时候在子工程中配置即可 -->

<pluginManagement>

<plugins>

<!-- 配置Tomcat插件 -->

<plugin>

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

<artifactId>tomcat7-maven-plugin</artifactId>

<version>2.2</version>

</plugin>

</plugins>

</pluginManagement>

</build>

</project>

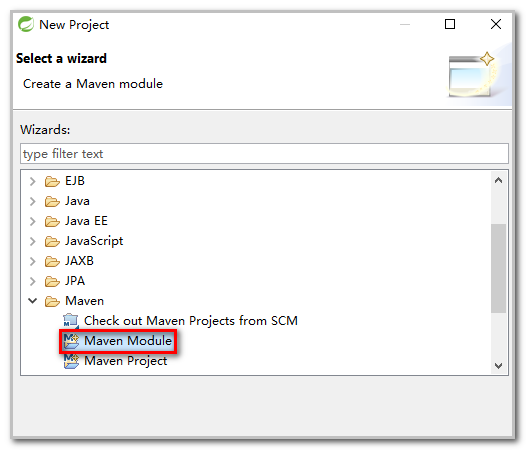
### 步骤三：建立子模块

#### ssm\_pojo

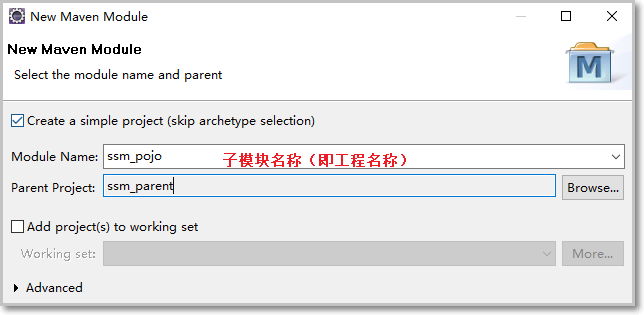
该模块用来存放实体类

右键父工程ssm-parent->new->project

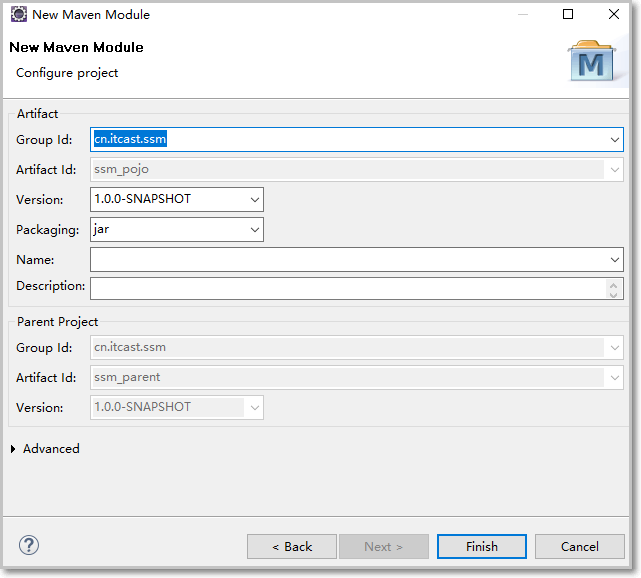
选择子模块:



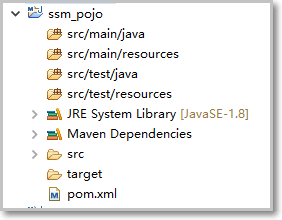
设置工程名称：



设置子模块坐标：



生成如下：



ssm\_pojo中pom.xml如下：

parent的中的坐标其实就是父工程的坐标，这样就相当于进行了继承。

一旦继承了父工程，那么父工程中的所有的依赖都可以在子工程中使用

<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/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>cn.itcast.ssm</groupId>

<artifactId>ssm\_parent</artifactId>

<version>1.0.0-SNAPSHOT</version>

</parent>

<artifactId>ssm\_pojo</artifactId>

</project>

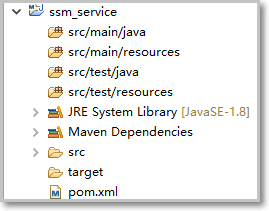
一旦继承了父工程，那么子工程中可以使用所有父工程中的依赖.

再次查看父工程ssm-parent的pom：在pom.xml的最下方发现多了一个子工程。



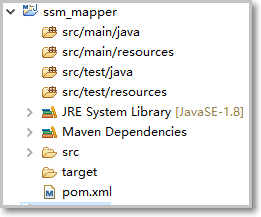
#### ssm\_service

过程同上：



#### ssm\_mapper

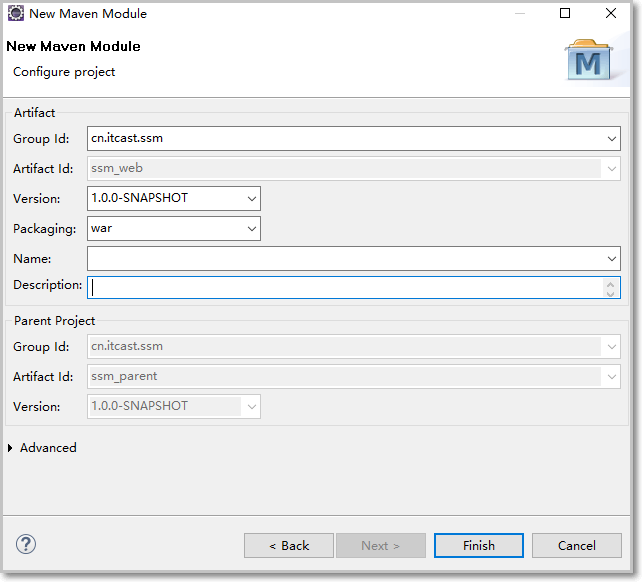
过程同上:



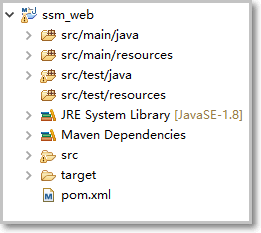
#### ssm\_web

第一步：创建子工程

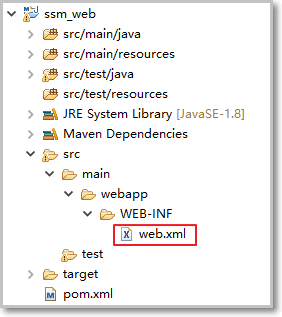
由于是我们最终要将工程进行发布，因此就需要一个web工程：ssm\_web的打包方式是war



生成如下:此时生成的工程没有WEB-INF目录和web.xml，因此需要手动生成。

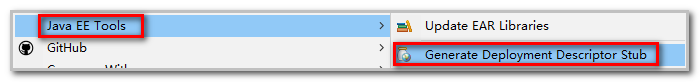


第二步：添加WEB-INF/web.xml



使用eclipse添加web.xml:

右键工程->Java EE Tools ->Generate Deployment Descriptor Stub



第三步:配置tomcat7的maven插件

最终需要将工程发布到tomcat上运行，而发布到tomcat上的工程一般都是war工程，因此需要将tomcat7的maven插件在ssm\_contoller中的pom.xml中进行配置：

<build>

<plugins>

<!-- 配置Tomcat插件 -->

<plugin>

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

<artifactId>tomcat7-maven-plugin</artifactId>

<configuration>

<port>8080</port>

<path>/</path>

</configuration>

</plugin>

</plugins>

</build>

注意:如果创建出来的子工程报红，右键该工程-maven-Update Project…即可。

### 步骤四：层之间的依赖设置

层和层之间是相互调用的关系，由于这些都是不同的工程，因此要设置相互之间的依赖才能进行调用。

思考：怎么进行依赖？

web需要依赖service ,service需要依赖dao, Dao需要依赖entity,而由于传递性依赖的特性，entity也就能被其他层调用了。

注意：在执行添加依赖之前，先执行Maven install,将所有工程安装到本地并重构索引，这样方便使用索引添加依赖，并且可以通过坐标找到本地仓库中的依赖。

#### mapper依赖pojo

打开mapper的pom，添加pojo的依赖：

添加ssm\_pojo依赖：

<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/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>cn.itcast.ssm</groupId>

<artifactId>ssm\_parent</artifactId>

<version>1.0.0-SNAPSHOT</version>

</parent>

<artifactId>ssm\_mapper</artifactId>

<dependencies>

<dependency>

<groupId>cn.itcast.ssm</groupId>

<artifactId>ssm\_pojo</artifactId>

<version>1.0.0-SNAPSHOT</version>

</dependency>

</dependencies>

</project>

#### Service依赖mapper

操作同上

在service中添加mapper的依赖：

<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/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>cn.itcast.ssm</groupId>

<artifactId>ssm\_parent</artifactId>

<version>1.0.0-SNAPSHOT</version>

</parent>

<artifactId>ssm\_service</artifactId>

<dependencies>

<dependency>

<groupId>cn.itcast.ssm</groupId>

<artifactId>ssm\_mapper</artifactId>

<version>1.0.0-SNAPSHOT</version>

</dependency>

</dependencies>

</project>

#### web依赖service

在controller中添加service的依赖：

<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/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>cn.itcast.ssm</groupId>

<artifactId>ssm\_parent</artifactId>

<version>1.0.0-SNAPSHOT</version>

</parent>

<artifactId>ssm\_web</artifactId>

<packaging>war</packaging>

<build>

<plugins>

<plugin>

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

<artifactId>tomcat7-maven-plugin</artifactId>

<configuration>

<port>8080</port>

<path>/</path>

</configuration>

</plugin>

</plugins>

</build>

<dependencies>

<dependency>

<groupId>cn.itcast.ssm</groupId>

<artifactId>ssm\_service</artifactId>

<version>1.0.0-SNAPSHOT</version>

</dependency>

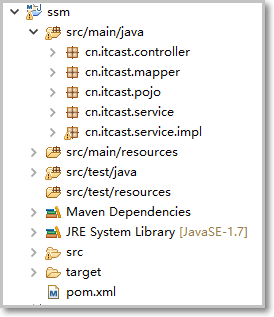
</dependencies>

</project>

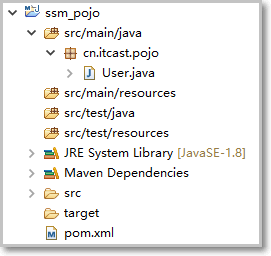
### 步骤五：项目分解

将之前的ssm工程中各层的类都分解到对应的子工程中

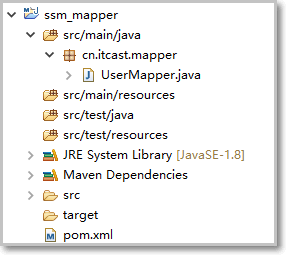
注意：在执行添加依赖之前，先执行Maven install,将所有工程安装到本地并重构索引，这样方便使用索引添加依赖，并且可以通过坐标找到本地仓库中的依赖。



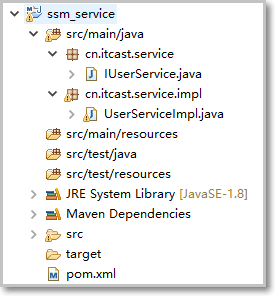
#### ssm\_pojo



#### ssm\_mapper



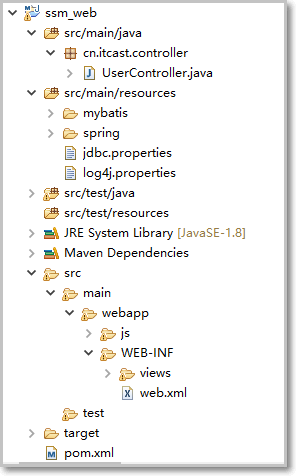
#### ssm\_service



#### ssm\_web

分拆到ssm-controller中时，注意：

1. 配置文件和日志
2. js和页面
3. Web.xml中的相关配置



### 步骤六：测试启动

#### 添加tomcat插件

ssm\_web是一个web工程，因此需要在ssm\_web中添加tomcat插件：

<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/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>cn.itcast.ssm</groupId>

<artifactId>ssm\_parent</artifactId>

<version>1.0.0-SNAPSHOT</version>

</parent>

<artifactId>ssm\_web</artifactId>

<packaging>war</packaging>

<dependencies>

<dependency>

<groupId>cn.itcast.ssm</groupId>

<artifactId>ssm\_service</artifactId>

<version>1.0.0-SNAPSHOT</version>

</dependency>

</dependencies>

<build>

<plugins>

<!-- 配置Tomcat插件 -->

<plugin>

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

<artifactId>tomcat7-maven-plugin</artifactId>

<configuration>

<port>8080</port>

<path>/</path>

</configuration>

</plugin>

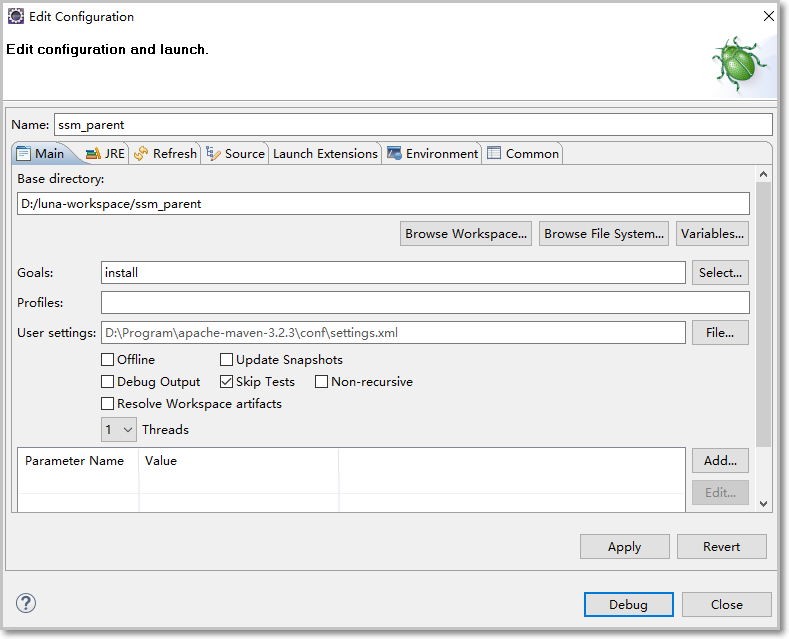
</plugins>

</build>

</project>

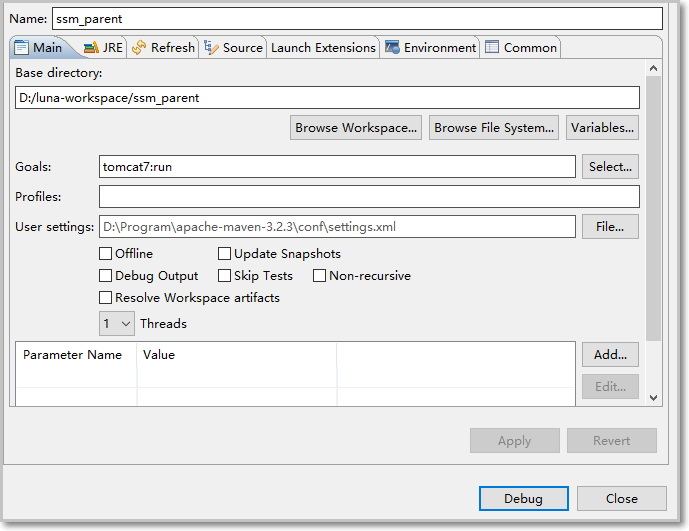
#### 安装到本地仓库

将项目安装到本地仓库后，各层之间才能从本地仓库找到各自的依赖：



#### 使用maven的tomcat插件启动

右键ssm\_parent，执行tomcat7:run即可。



如果发现报错如下：

[ERROR] Failed to execute goal on project ssm\_web: Could not resolve dependencies for project cn.itcast.maven:ssm\_web:war:1.0.0-SNAPSHOT: Could not find artifact cn.itcast.maven:ssm\_service:jar:1.0.0-SNAPSHOT -> [Help 1]

[ERROR]

[ERROR] To see the full stack trace of the errors, re-run Maven with the -e switch.

[ERROR] Re-run Ma

ven using the -X switch to enable full debug logging.

[ERROR]

[ERROR] For more information about the errors and possible solutions, please read the following articles:

[ERROR] [Help 1] http://cwiki.apache.org/confluence/display/MAVEN/DependencyResolutionException

报错原因：ssm\_web工程中在调用ssm\_service的时候提示找不到ssm\_service，原因是工程还没有安装到本地仓库或者在ssm\_web中没有依赖ssm\_service，因此找不到。

解决方案：检查ssm-controler使用添加了ssm\_service的依赖，并且将所有工程都install到本地仓库中。

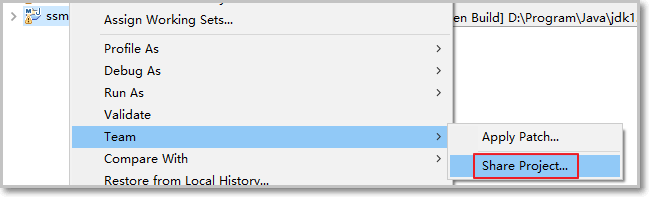
每一个都需要执行install吗？不需要，因为我们使用了多模块聚合（modules），所以只需要对ssm\_parent执行install即可。

将父工程安装到本地仓库后，会自动将子工程也安装到本地仓库中。

# SVN上传聚合工程

## 将maven工程发布到svn

### 步骤一：上传到svn

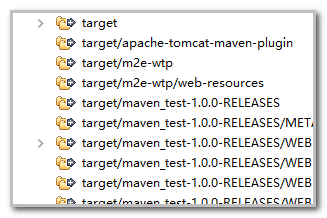


### 步骤二：忽略指定的资源

在将maven项目上传到svn服务器的时候，有几类文件是不需要上传的：

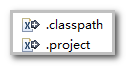
1. 所有含target的文件
2. 环境文件：.classpath、.settings、project

target文件:



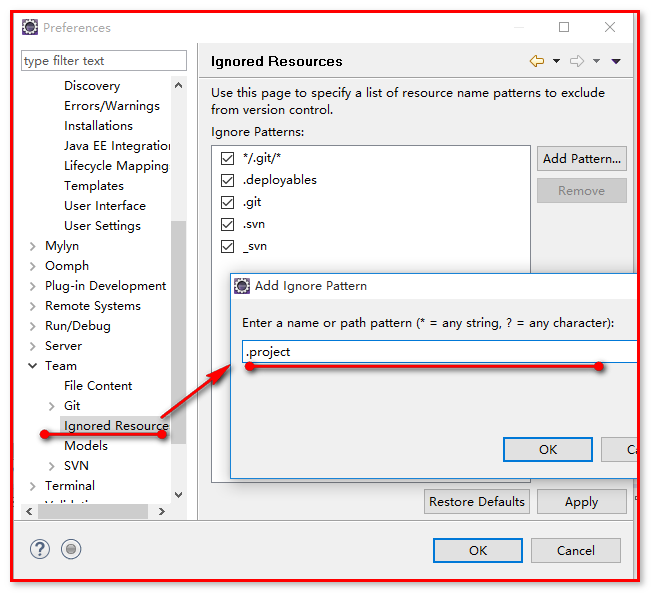
环境文件：





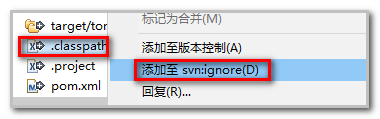
#### 全局指定（用的多）

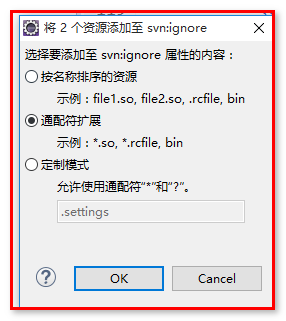
对所有的项目有效window-->属性



#### 单个指定

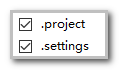
只对当前项目有效





* \* 匹配任何字符串，包括空串(没有字符)
* ? 匹配任何单字符
* [...] 匹配任何单在方括号[]内的单字符，在方括号内，一对字符被“-”分隔，匹配任何词汇表(lexically)上在他们中间的字符。例如[AGm-p]匹配任何单个的A,G,m,n,o或者p。

这里我们选择使用全局忽略指定资源：

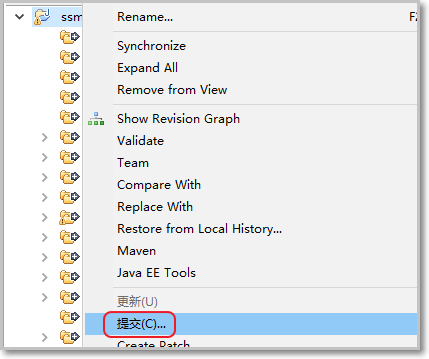






### 步骤三：提交

然后提交即可：

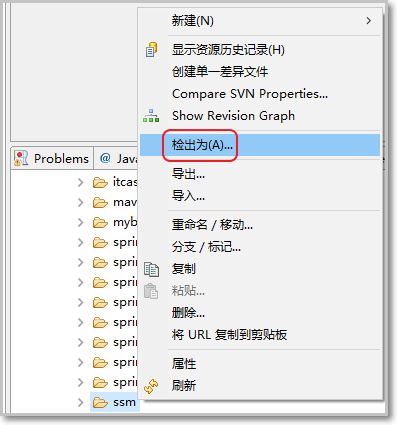


## 从svn检出maven工程

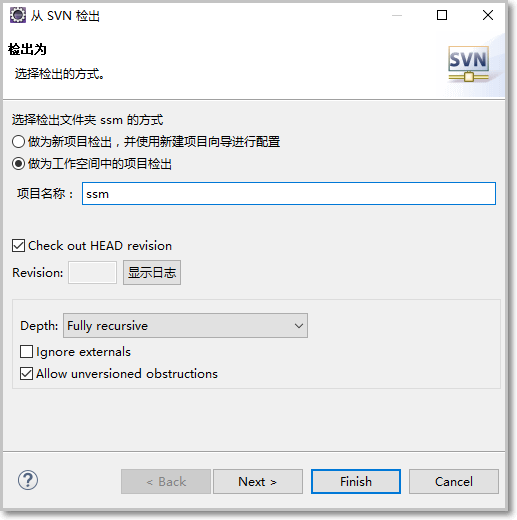
### 步骤一：检出工程

在检出之前，可以先将原项目删除。

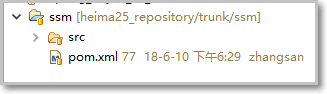
右键svn服务器中的maven项目，选择检出：



执行检出：

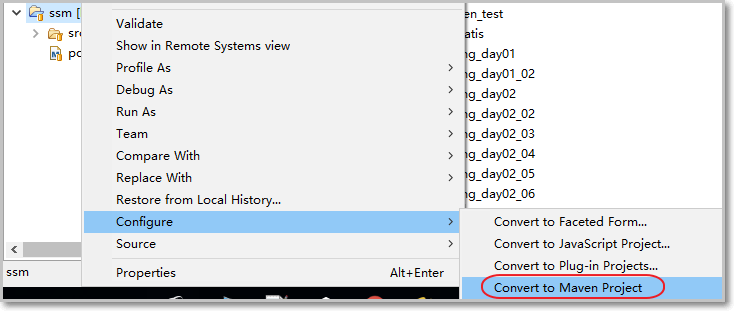


检出如下：检出的项目并不是一个maven工程的结构。



### 步骤二：转换为maven工程

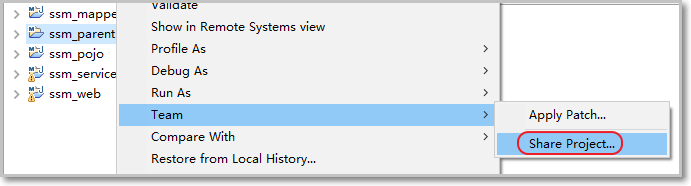
右键工程，进行转换：



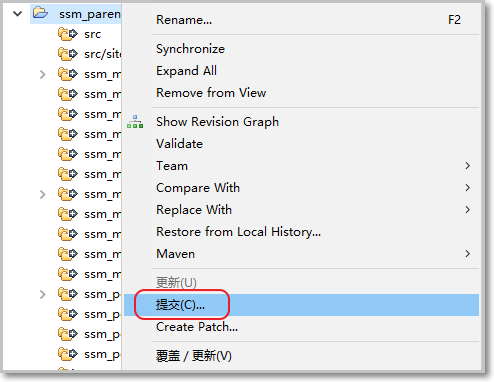
## 上传聚合工程

### 步骤一：提交父工程

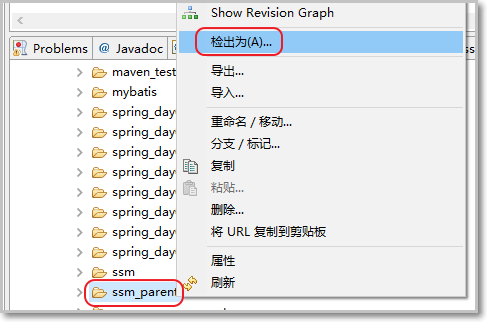
右键父工程，上传：



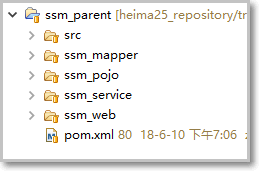
提交:



### 步骤二：检出聚合工程



检出效果:



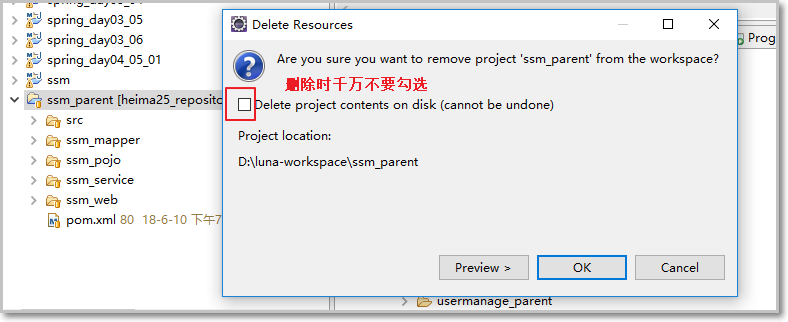
### 步骤三：重新导入聚合工程

问题：在检出之后，发现只有父工程？

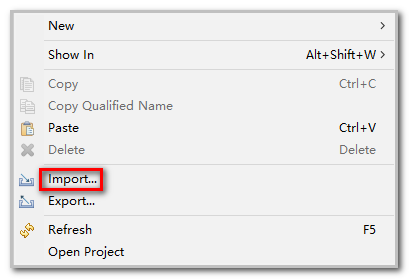
解决方案：在检出文件的时候其实已经将聚合工程中的所有项目到检出到工作空间中了。

查看工作空间如下：

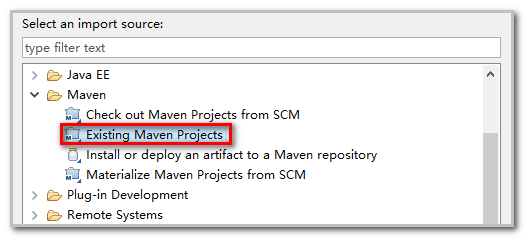
删除工程：注意千万不要将工作空间中的工程也选择删除。



删除之后，重新选择从工作空间中导入：



选择已存在的maven工程：



选择工作空间中的父工程ssm\_parent即可