SSM 整合&Maven 聚合工程

1.1. 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

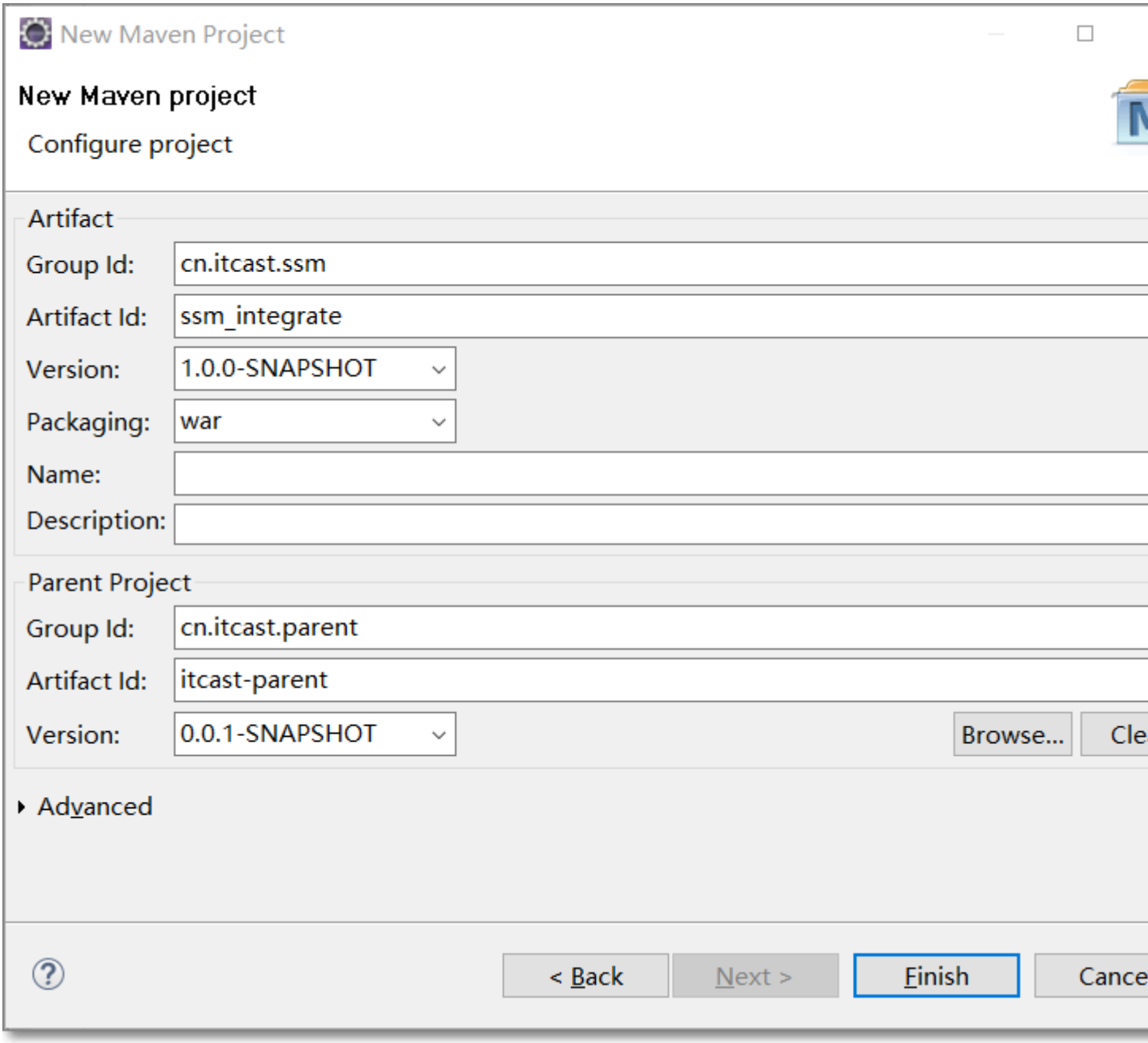
1.2. 使用技术

Spring + SpringMVC + Mybatis

数据库：mysql

2.创建工程---环境整理

2.1. 创建工程



2.2.引入依赖

参照 itcast-parent 工程的 pom.xml，在 pom.xml 中引入所需依赖

```
<dependencies>

      <!-- 单元测试 -->

      <dependency>
         <groupId>junit</groupId>
         <artifactId>junit</artifactId>
         <scope>test</scope>
      </dependency>

     <!—spring整合单元测试 -->

       <dependency>
         <groupId>org.springframework</groupId>
         <artifactId>spring-test</artifactId>
         <version>4.3.13.RELEASE</version>
      </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>
```

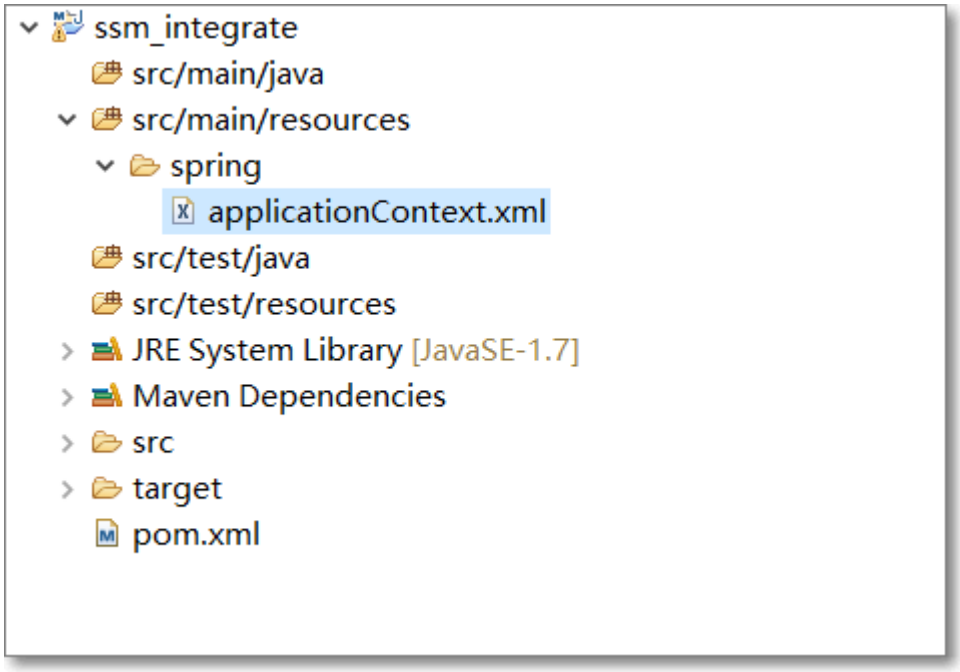
```
        <!-- MySql -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
        </dependency>

        <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>
```

3.配置 spring 环境

3.1.配置 applicationContext.xml

1、创建 spring 目录并在其目录下新建 applicationContext.xml 文件



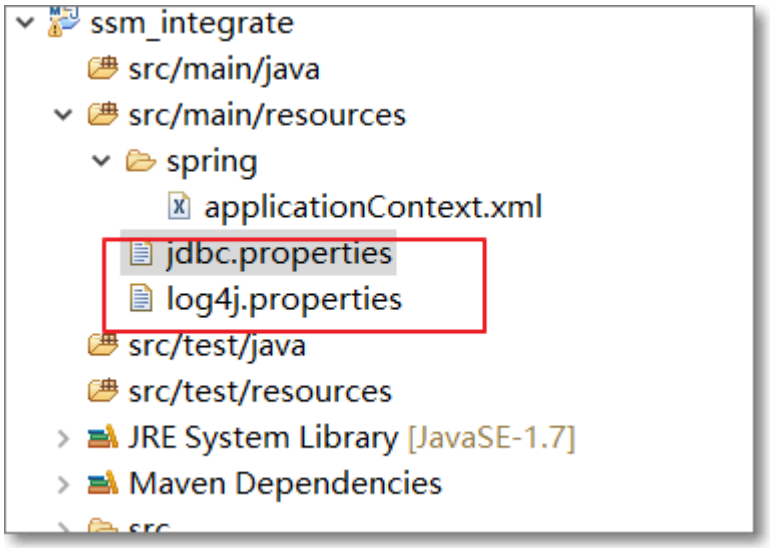
applicationContext.xml 的内容如下：

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:p="http://www.springframework.org/schema/p"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:tx="http://www.springframework.org/schema/tx"
    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-4.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-4.0.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-4.0.xsd
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx-4.0.xsd
    http://www.springframework.org/schema/util
    http://www.springframework.org/schema/util/spring-util-4.0.xsd">

    <!-- 引入外部属性配置文件 -->
    <context:property-placeholder location="classpath:jdbc.properties" />

    <!-- 配置数据源 -->
    <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource">
        <property name="driverClassName" value="${jdbc.driverClass}" />
        <property name="url" value="${jdbc.url}" />
        <property name="username" value="${jdbc.username}" />
        <property name="password" value="${jdbc.password}" />
    </bean>
```

如上黄色标识。于是，又需要引入 jdbc.properties 资源文件

3.2. 引入资源文件(jdbc.properties 以及 log4j.properties)

由于 applicationContext.xml 中数据源的连接信息是配置在 jdbc.properties 资源文件中的，所以这里需要引入该资源文件，参考之前的工程，可直接 copy 过来（顺便把 log4j.properties 文件一块儿 copy 过来）。如下：



jdbc.properties 内容如下：

```
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://127.0.0.1:3306/mybatis
jdbc.username=root
jdbc.password=root
```

4.整合 mybatis

在 spring 目录下新建 applicationContext-mybatis.xml，这样不同的框架或工具跟 spring 的基础配置分开配置，方便管理。

4.1. 配置 applicationContext-mybatis.xml

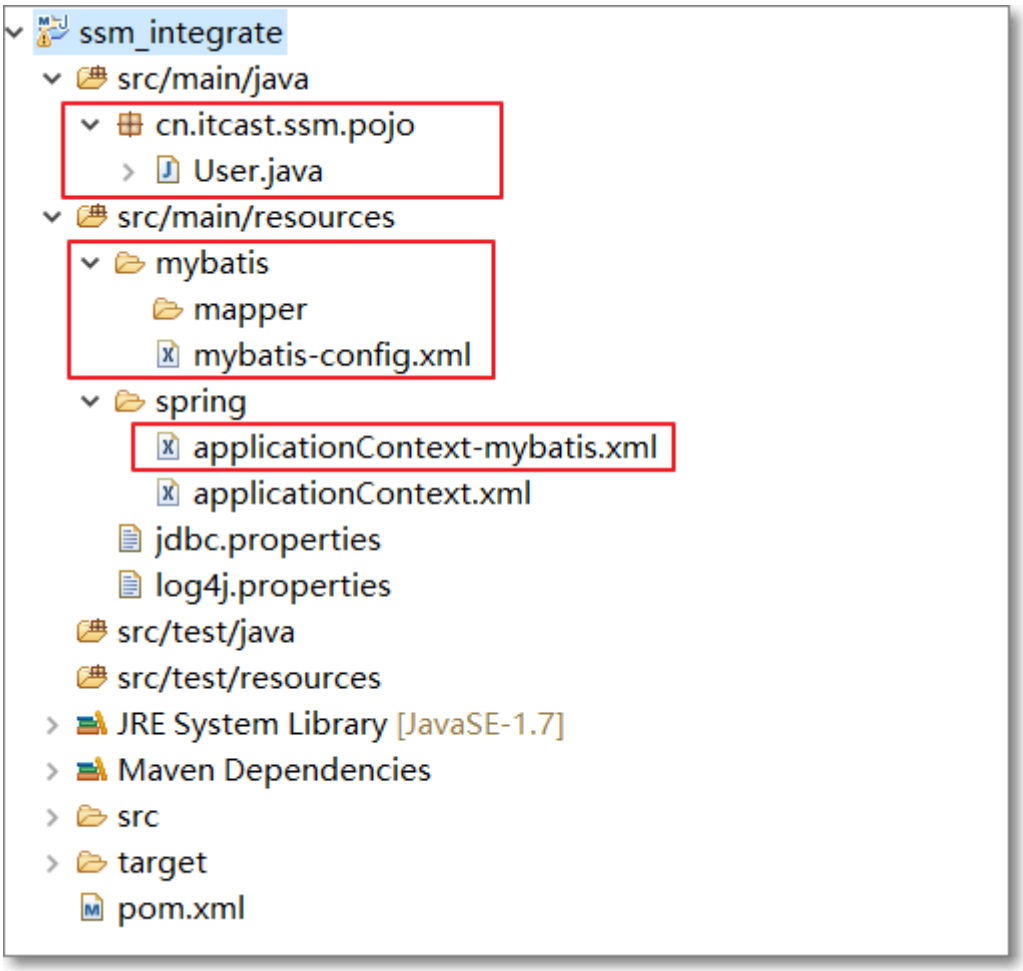
思路：spring 整合 mybatis，能整合 mybatis 的什么？

sqlSessionFactory、sqlSession、mapper 接口的初始化，过于复杂，能否交给 spring

mybatis 全局配置文件的读取，能否交给 spring

映射文件的引入，之前存在瑕疵，能否解决

在 spring 包下创建 applicationContext-mybatis.xml 配置文件，约束头信息参考 applicationContext.xml 配置；mybatis-config.xml 的内容参考之前的配置，并创建 cn.itcast.ssm.pojo 包目录将实体类 copy 到包中。



Mybatis-config.xml 内容：数据源已经在 spring 中配置,所以无需在 mybatis-config.xml 配置相关信息

```
<?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>
      <!-- 开启驼峰命名匹配 -->
      <setting name="mapUnderscoreToCamelCase" value="true" />
```

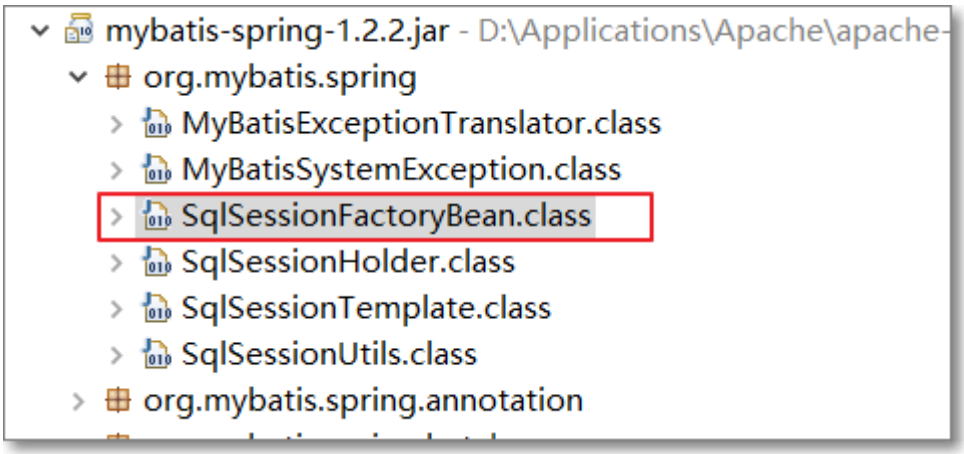
```
    </settings>


    <typeAliases>
        <!-- 别名 -->
        <package name="cn.itcast.ssm.pojo" />
    </typeAliases>

    <!--
    <mappers>
        <mapper resource="UserMapper.xml" />
    </mappers>
    -->

</configuration>
```

4.1.1. 构建 SqlSessionFactory

SqlSessionFactory 的构建过程，在 mybatis 中是比较麻烦的。当 mybatis 遇到 spring 之后，这个问题变的非常简单了。

在 mybatis-spring 的整合包下，存在一个 sqlSessionFactoryBean，它是用来在 spring 容器中生产 sqlSessionFactory 的工厂 Bean

sqlSessionFactoryBean 源码位置：



查看 sqlSessionFactoryBean 源码注释：

```
*/
public class SqlSessionFactoryBean implements FactoryBean<SqlSessionFactory>, I
  private sta
  private Res
  private Res
  private Dat
  private Tra
  private Pro
  private Sql
  private Sql
  private Str
```

org.mybatis.spring.SqlSessionFactoryBean

FactoryBean that creates an MyBatis SqlSessionFactory. This is the usual way to set up a shared M
context; the SqlSessionFactory can then be passed to MyBatis-based DAOs via dependency injection. E
JtaTransactionManager can be used for transaction demarcation in combination with a SqlSessior
span multiple databases or when container managed transactions (CMT) are being used.

用来生产sqlSessionFactor

Version:
Id
Author:
Putthibong Boonbong
Hunter Presnall
Eduardo Macarron
See Also:
setConfigLocation
setDataSource

mybatis全局配置以及数据源参数

结论：

应该在 applicationContext-mybatis.xml 中配置 SqlSessionFactoryBean，并且配置 dataSource 以及 configLocation 的属性

配置 applicationContext-mybatis.xml：

```
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:context="http://www.springframework.org/schema/context"
xmlns:p="http://www.springframework.org/schema/p"
   xmlns:aop="http://www.springframework.org/schema/aop"
xmlns:tx="http://www.springframework.org/schema/tx"
   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-4.0.xsd
   http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.0.xsd
   http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-4.0.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-4.0.xsd
   http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util-4.0.xsd">
```

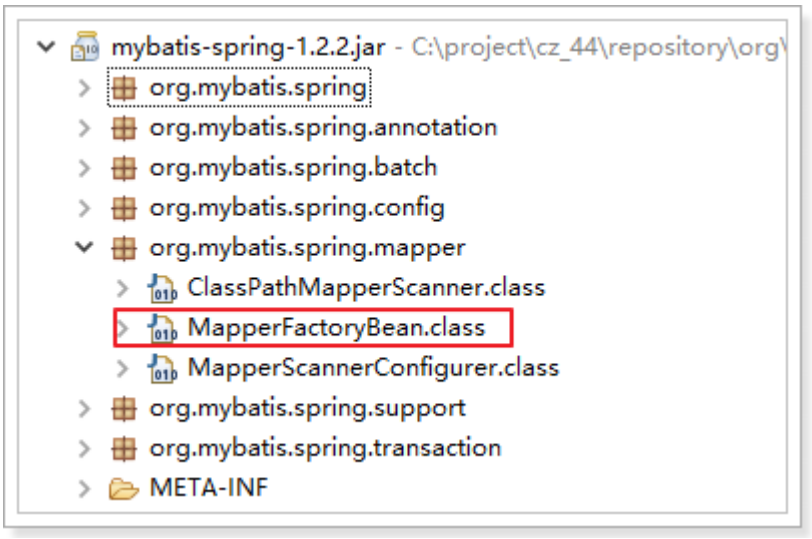
```
    <!-- spring构建sqlSessionFactory -->
    <bean id="sqlSessionFactory"
class="org.mybatis.spring.SqlSessionFactoryBean">
        <!-- 指定mybatis的数据源 -->
        <property name="dataSource" ref="dataSource"></property>
        <!-- 指定mybatis的全局配置文件 -->
        <property name="configLocation" value="classpath:mybatis/mybatis-
config.xml"></property>
    </bean>

</beans>
```

4.1.2. 配置 mapper 接口

既然 sqlSessionFactory 交给 spring 管理了，那么 mybatis 的 mapper 接口的动态代理实现能不能也交给 spring 进行管理呢?

1、参照源码注释

在 mybatis-spring 的整合包下，存在 MapperFactoryBean 这样一个工厂 bean，它可以帮咱们完成:



注释中提供了使用案例:

```
55 public class MapperFactoryBean<T> extends SqlSessionDaoSupport implements F
56
57   private Cla
58
59   private boo
60
61   /**
62    * Sets the
63    *
64    * @param m
65    */
66   public void
67     this.mapp
68   }
69
70   /**
71    * If addTo
72    * it must
73    * <p>
74    * If it is
```

org.mybatis.spring.mapper.MapperFactoryBean<T>

BeanFactory that enables injection of MyBatis mapper interfaces. It can be set up with a SqlSessio
SqlSessionTemplate.

Sample configuration:

```
<bean id="baseMapper" class="org.mybatis.spring.mapper.MapperFactoryBean" a
    <property name="sqlSessionFactory" ref="sqlSessionFactory" />
  </bean>

  <bean id="oneMapper" parent="baseMapper">
    <property name="mapperInterface" value="my.package.MyMapperInterface" /
  </bean>

  <bean id="anotherMapper" parent="baseMapper">
    <property name="mapperInterface" value="my.package.MyAnotherMapperInter
  </bean>
```

Note that this factory can only inject *interfaces*, not concrete classes.

2、 参照 mybatis 整合 spring 的官方文档：

数据映射器接口可以按照如下做法加入到 Spring 中:

```
<bean id="userMapper" class="org.mybatis.spring.mapper.MapperFactoryBean">
  <property name="mapperInterface" value="org.mybatis.spring.sample.mapper.UserMapper" />
  <property name="sqlSessionFactory" ref="sqlSessionFactory" />
</bean>
```

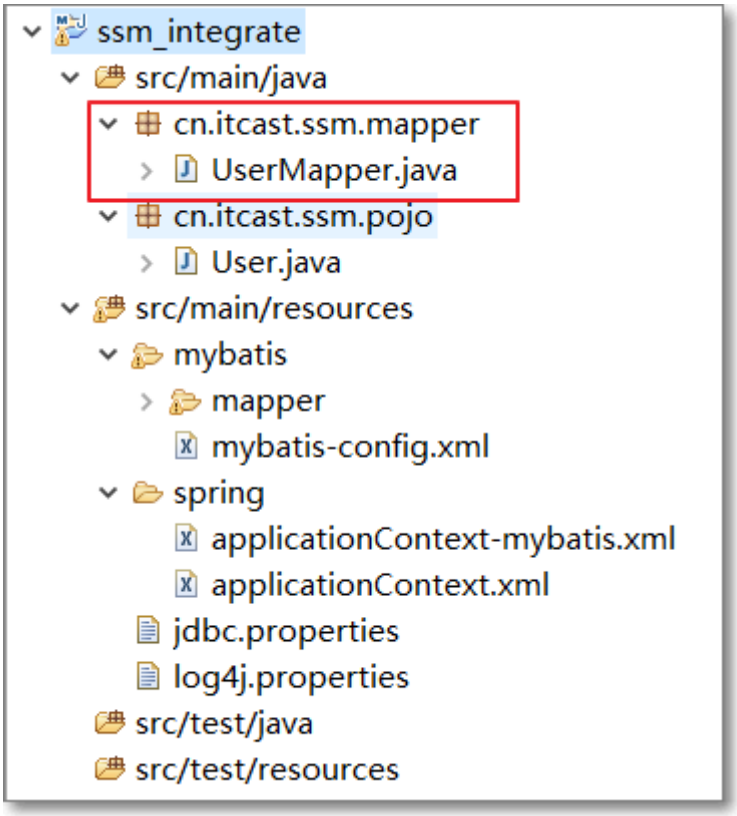
MapperFactoryBean 创建的代理类实现了 UserMapper 接口,并且注入到应用程序中。因为代理创建在运行时环境
须是一个接口,而不是一个具体的实现类。

在 applicationContext-mybatis.xml 中，将 Mapper 接口交给 spring 管理：

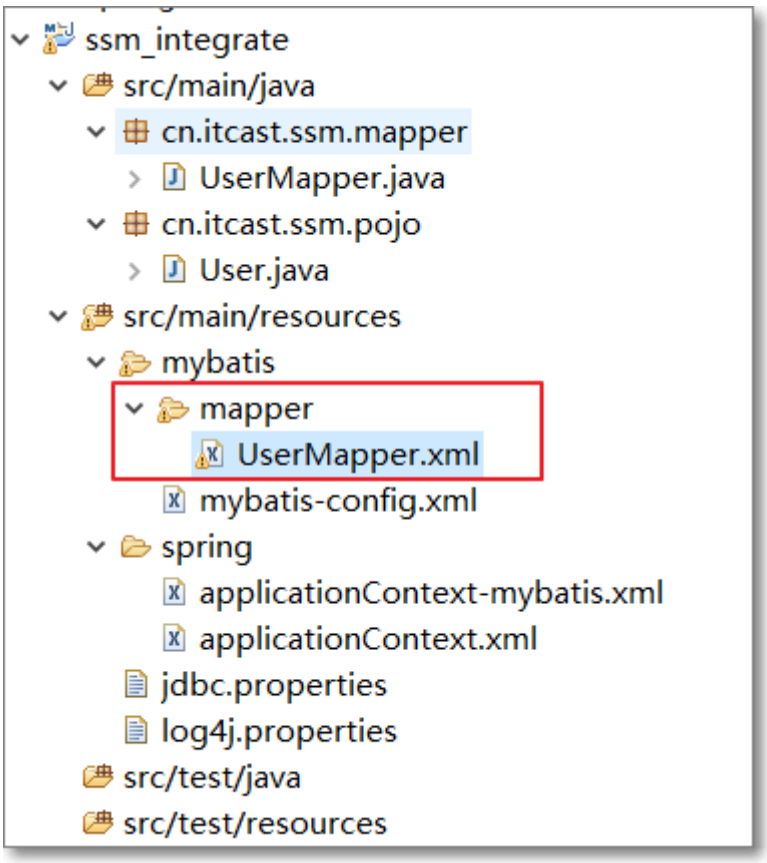
```
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:context="http://www.springframework.org/schema/context"
xmlns:p="http://www.springframework.org/schema/p"
   xmlns:aop="http://www.springframework.org/schema/aop"
xmlns:tx="http://www.springframework.org/schema/tx"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
   http://www.springframework.org/schema/context
```

```
http://www.springframework.org/schema/context/spring-context-4.0.xsd
    http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-4.0.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-4.0.xsd
    http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util-4.0.xsd">

    <!-- spring构建sqlSessionFactory -->
    <bean id="sqlSessionFactory"
class="org.mybatis.spring.SqlSessionFactoryBean">
        <!-- 指定mybatis的数据源 -->
        <property name="dataSource" ref="dataSource"></property>
        <!-- 指定mybatis的全局配置文件 -->
        <property name="configLocation" value="classpath:mybatis/mybatis-
config.xml"></property>
    </bean>

    <!-- spring实例化usermapper的动态实现 -->
    <bean id="userMapper"
class="org.mybatis.spring.mapper.MapperFactoryBean">
        <property name="mapperInterface"
value="cn.itcast.ssm.mapper.UserMapper" />
        <property name="sqlSessionFactory" ref="sqlSessionFactory" />
    </bean>

</beans>
```

创建 cn.itcast.ssm.mapper 目录及 UserMapper 接口：



创建对应的 UserMapper.xml 文件到 mybatis/mapper 目录下



将 UserMapper.xml 文件关联到 mybatis-config.xml 中:

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

4.2. Junit 测试整合

思路:

1、在 UserMapper 接口中定义一个方法（根据 id 查询用户信息）

2、在 UserMapper.xml 中定义根据 id 查询用户信息的 Statement

3、创建 UserMapper 接口的 junit test cast（即 UserMapperTest.java），通过 spring 整合 junit 测试获取 userMapper 对象

UserMapper.java:

```
public interface UserMapper {

   public User selectUserById(@Param("id")Long id);

}
```

UserMapper.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="cn.itcast.ssm.mapper.UserMapper">

   <select id="selectUserById" resultType="User">
      select * from tb_user where id = #{id}
   </select>
```

```
</mapper>
```

给 UserMapper 接口创建 junit 测试用例 UserMapperTest，内容：

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations =
{ "classpath:spring/applicationContext.xml",
      "classpath:spring/applicationContext-mybatis.xml" })
public class UserMapperTest {

   @Autowired
   private UserMapper userMapper;

   @Test
   public void testSelectUserById() {
      User user = userMapper.selectUserById(1L);
      System.out.println(user);
   }

}
```

控制台打印出用户信息，说明整合成功，日志输出：

```
2017-04-17 23:30:10,408 [main] [cn.itcast.usermanage.mapper.UserMapper.queryUserBy
2017-04-17 23:30:10,450 [main] [cn.itcast.usermanage.mapper.UserMapper.queryUserBy
2017-04-17 23:30:10,490 [main] [cn.itcast.usermanage.mapper.UserMapper.queryUserBy
2017-04-17 23:30:10,500 [main] [org.mybatis.spring.SqlSessionUtils]-[DEBUG] Closin
2017-04-17 23:30:10,500 [main] [org.springframework.jdbc.datasource.DataSourceUtil
User [id=1, userName=zhangsan, password=123456, name=张三, age=30, sex=1, birthday=
```

4.3. 优化整合程序

那些配置需要优化：

1、Mapper 接口的 spring 配置太过麻烦，每一个 Mapper 接口都要去配置

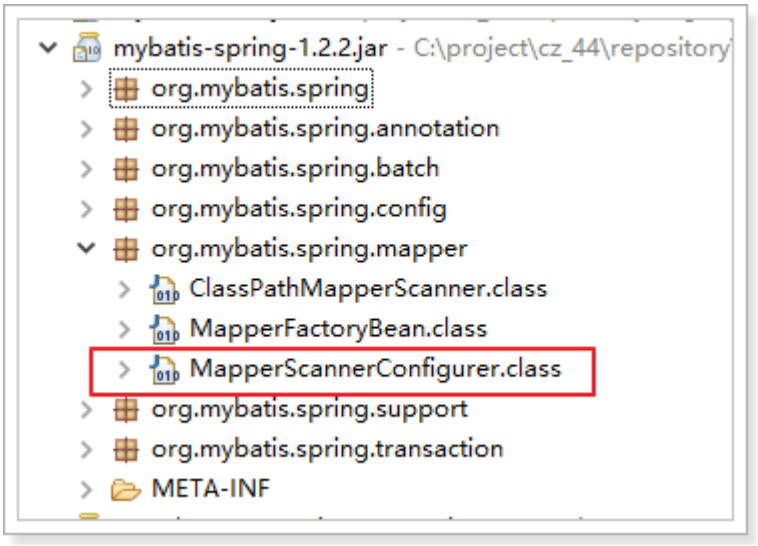
2、Mapper 映射文件，每次都要在 mybatis-config.xml 中引入

3、别名扫描，是 spring 比较擅长的，能否交给 spring 管理

4.3.1. 使用 Mapper 接口扫描

Mapper 接口的配置太麻烦，每次都要去配置，并且每次都要配置多行。

在 mybatis-spring 的整合包中，提供了 MapperScannerConfigurer 接口扫描类：



类中有以下属性

```
public class MapperScannerConfigurer implements BeanDefinitionRegistryPostProce

  private String basePackage;

  private boolean addToConfig = true;

  private SqlSessionFactory sqlSessionFactory;

  private SqlSessionTemplate sqlSessionTemplate;

  private String sqlSessionFactoryBeanName;

  private String sqlSessionTemplateBeanName;

  private Class<? extends Annotation> annotationClass;

  private Class<?> markerInterface;

  private ApplicationContext applicationContext;

  private String beanName;
```

需要配置 basePackage 以及 sqlSessionFactory 属性，但是 setSqlSessionFactory 方法已过期，推荐使用 setSQLSessionFactoryBeanName

```
/**
 * Specifies which {@code SqlSessionFactory} to use in the case that there is
 * more than one in the spring context. Usually this is only needed when you
 * have more than one datasource.
 * <p>
 * Use {@link #setSqlSessionFactoryBeanName(String)} instead.
 *
 * @param sqlSessionFactory
 */
@Deprecated
public void setSqlSessionFactory(SqlSessionFactory sqlSessionFactory) {
  this.sqlSessionFactory = sqlSessionFactory;
}
```

setSqlSessionFactory已过期
推荐setSqlSessionFactoryBeanName

查看 MapperScannerConfigurer 的注释：

```
MapperScannerConfigurer implements BeanDefinitionRegistryPostProcessor, Initia
```

The basePackage property can contain more than one package name, separated by either commas or semicolons.

多个包路

This class supports filtering the mappers created by either specifying a marker interface or an annotation. The annotatio
annotation to search for. The markerInterface property specifies a parent interface to search for. If both properties are
interfaces that match *either* criteria. By default, these two properties are null, so all interfaces in the given basePackage a

This configurer enables autowire for all the beans that it creates so that they are automatically autowired with the proper
SqlSessionTemplate. If there is more than one SqlSessionFactory in the application, however, autowiring cannot be
specify either an SqlSessionFactory or an SqlSessionTemplate to use via the *bean name* properties. Bean names are
because Spring does not initialize property placeholders until after this class is processed.

Passing in an actual object which may require placeholders (i.e. DB user password) will fail. Using bean names defers actu
startup process, after all placeholder substituation is completed. However, note that this configurer does support property
The basePackage and bean name properties all support ${property} style substitution.

Configuration sample:

使用方法：

```
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
      <property name="basePackage" value="org.mybatis.spring.sample.mapper" />
      <!-- optional unless there are multiple session factories defined -->
      <property name="sqlSessionFactoryBeanName" value="sqlSessionFactory" />
  </bean>
```

可选，除非

如果有多个 sqlSessionFactory 时，以逗号或者分号隔开，如果只有一个 sqlSessionFactory

时，可以省略 sqlSessionFactoryBeanName 的配置。

而我们只有一个 sqlSessionFactory，所以，只需要在 applicationContext-mybatis.xml 中配置：

```
<!-- mapper接口的包扫描 -->
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage" value="cn.itcast.ssm.mapper" />
</bean>
```

4.3.2. Mapper.xml 交给 spring

解决 mybatis 的 resource 配置方式，造成的麻烦（每次都要配置）

解决 mybatis 的 package 包扫描，造成的配置和 java 耦合。

```
<!-- spring构建sqlSessionFactory -->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <!-- 指定mybatis的数据源 -->
    <property name="dataSource" ref="dataSource"></property>
    <!-- 指定mybatis的全局配置文件 -->
    <property name="configLocation" value="classpath:mybatis/mybatis-config.xml"

    <!-- 扫描mapper下的所有xml文件 -->
    <property name="mapperLocations" value="classpath:mybatis/mapper/**/*.xml"><
</bean>
```

4.3.3. 别名扫描交给 spring

```
<!-- 整合mybatis,管理sqlSessionFactory -->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource"></property>
    <property name="configLocation" value="classpath:mybatis/mybatis-config.xml">
    <!-- 扫描mapper映射文件 -->
    <property name="mapperLocations" value="classpath:mybatis/mapper/**/*.xml"><

    <!-- 设置别名扫描 -->
    <property name="typeAliasesPackage" value="cn.itcast.ssm.pojo" />
</bean>
```

4.4. 整合的终极配置

4.4.1. applicationContext-mybatis.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:context="http://www.springframework.org/schema/context" xmlns:p="http://
   xmlns:aop="http://www.springframework.org/schema/aop" xmlns:tx="http://www.spr
   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-4.0.xsd
   http://www.springframework.org/schema/context http://www.springframework.org/s
4.0.xsd
   http://www.springframework.org/schema/aop http://www.springframework.org/schem
http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/
   http://www.springframework.org/schema/util http://www.springframework.org/sche

   <!-- 构建sqlSessionFactory -->
   <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
      <!-- 数据源，必须 -->
      <property name="dataSource" ref="dataSource" />
      <!-- mybatis的全局配置文件 -->
      <property name="configLocation" value="classpath:mybatis/mybatis-config.xml
      <!-- 引入mybatis映射文件 -->
      <property name="mapperLocations" value="classpath:mybatis/mappers/**/*.xml"
      <!-- 别名扫描 -->
      <property name="typeAliasesPackage" value="cn.itcast.ssm.pojo" />
   </bean>

   <!-- mapper接口的扫描 -->
   <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
      <property name="basePackage" value="cn.itcast.ssm.mapper" />
   </bean>
```

```
</beans>
```

4.4.2. 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，经典的数据库列名（多个单词时，以下划线连接）到经典java属性名（
        <setting name="mapUnderscoreToCamelCase" value="true" />
    </settings>

</configuration>
```

5.Spring 与 springmvc 的整合

Springmvc 与 spring 是同一个体系下的，他们在没有特殊需求的情况下是不用整合的

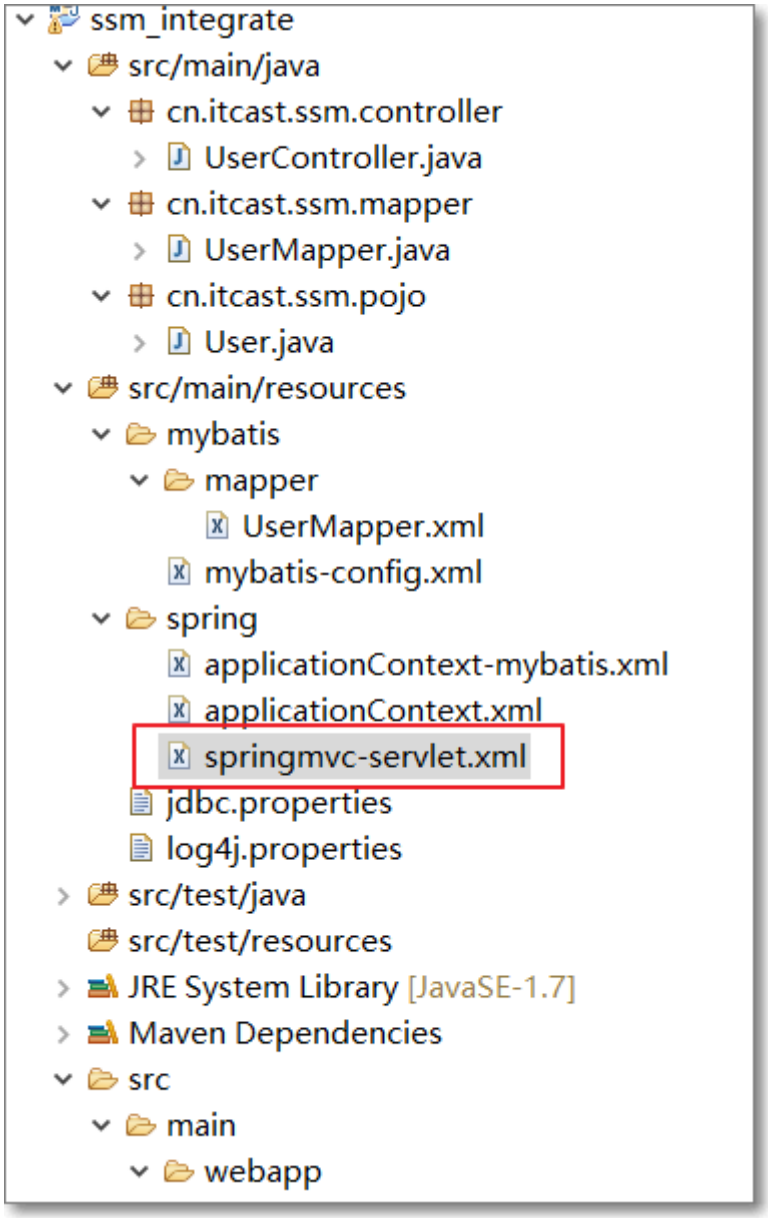
5.1.配置 web.xml

Web.xml 的内容如下：

```
<?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"
    id="MyWebApp" version="2.5">
    <display-name>ssm_integrate</display-name>


    <!--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>


    <!-- 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>

    <!-- 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>

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

</web-app>
```

5.2. 在 spring 目录下创建 springmvc-servlet.xml

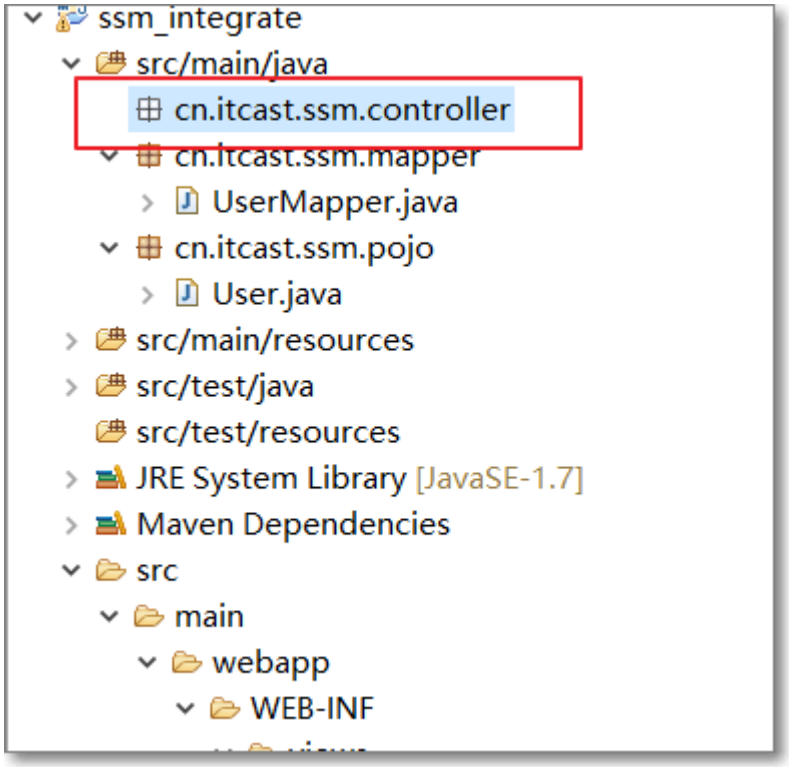


内容为:

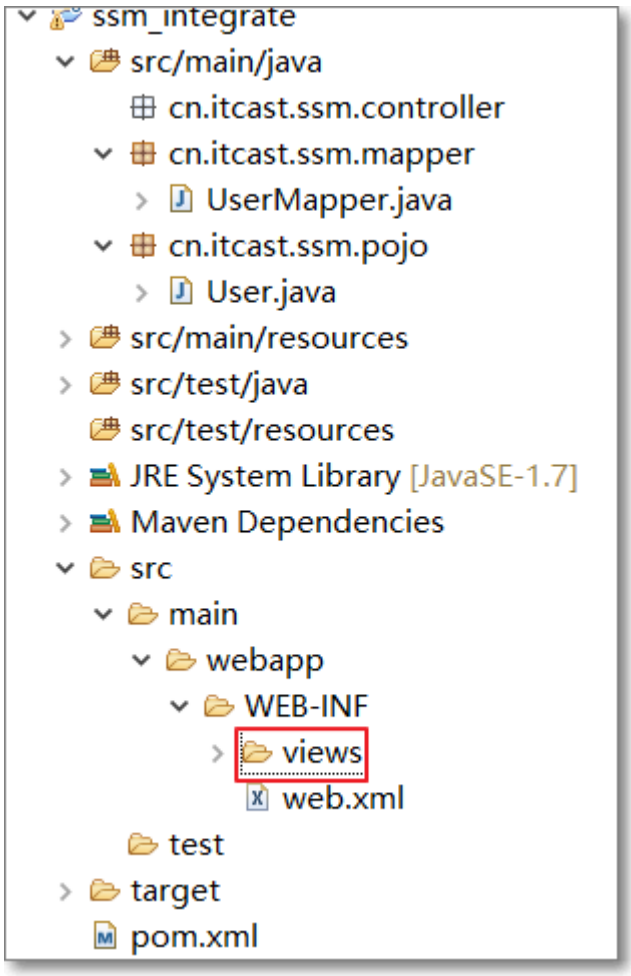
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:p="http://www.springframework.org/schema/p"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    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">

    <!-- 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 />

    <!-- 开启注解扫描 -->
    <context:component-scan base-
package="cn.itcast.ssm.controller"></context:component-scan>

    <!-- 配置视图解析器 -->
    <!-- Example: prefix="/WEB-INF/jsp/", suffix=".jsp", viewname="test"
-> "/WEB-INF/jsp/test.jsp"  -->
    <bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver"
>
        <property name="prefix" value="/WEB-INF/views/"></property>
        <property name="suffix" value=".jsp"></property>
    </bean>

</beans>
```

1、创建 springmvc-servlet.xml 配置文件中的注解扫描对应的包目录



2、创建视图解析器对应的视图目录



5.3. 创建 user.jsp

```
<table border="1">
        <tr>
            <td>ID</td>
            <td>用户名</td>
            <td>姓名</td>
            <td>年龄</td>
            <td>生日</td>
            <td>创建日期</td>
            <td>更新日期</td>
        </tr>
    </table>
```

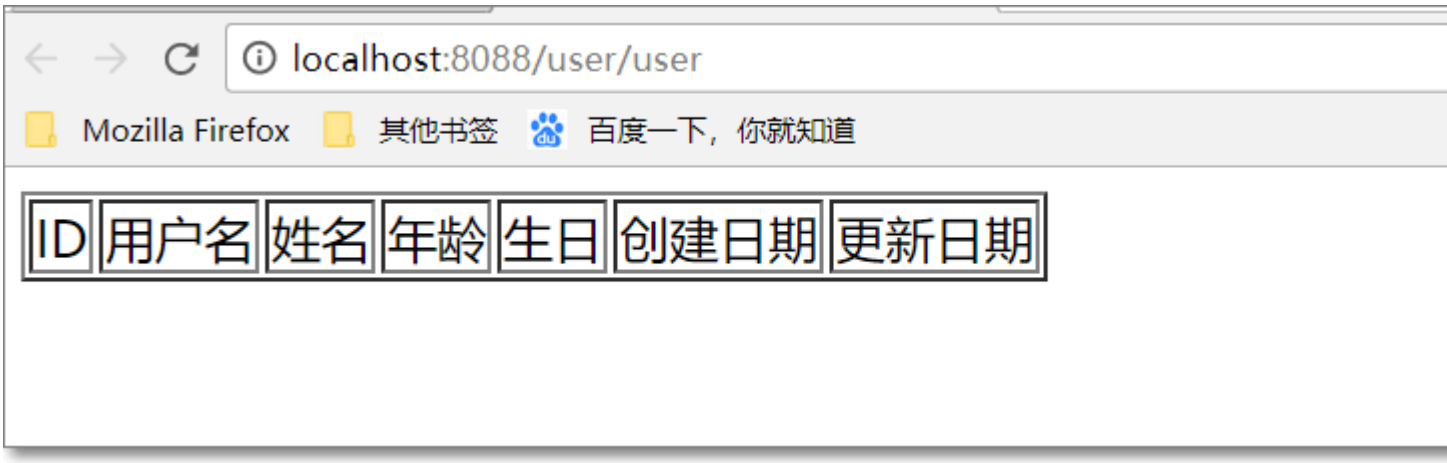
5.4. 跳转到 user.jsp

由于 user.jsp 在 WEB-INF 下，不能直接访问到该资源。必须通过 Controller 方法做跳转。

创建并编写 UserController 方法跳转到 user.jsp

```
@Controller
@RequestMapping("user")
public class UserController {


    @RequestMapping("user")
    public String toUser(){
        return "user";
    }
}
```

启动 tomcat 之后的访问效果：



6.根据 id 查询用户信息并在页面显示

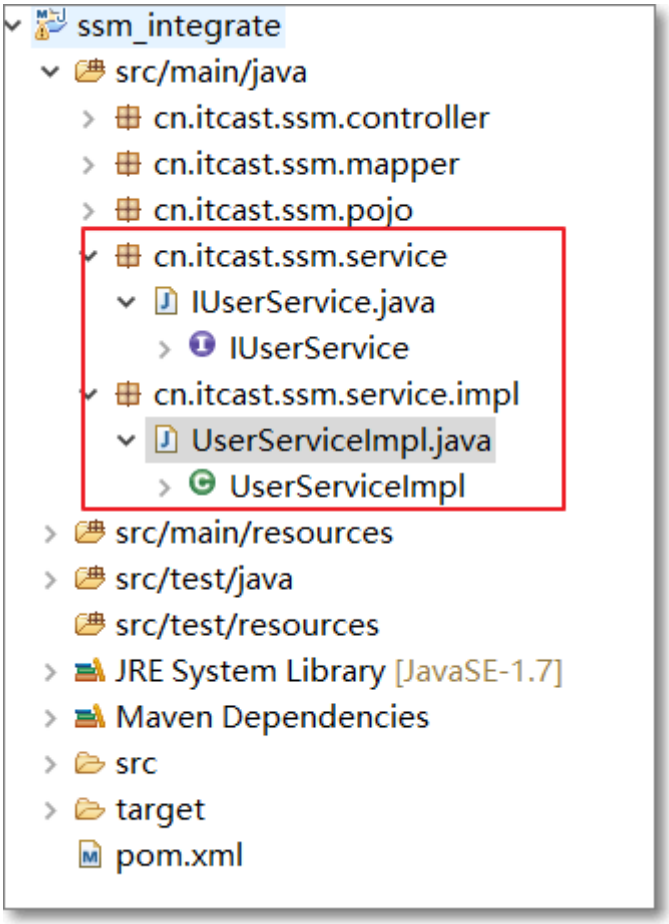
1.通过访问 url 传递用户 ID 访问对应的 Controller 的方法

2.Controller 通过调用 Service 的业务方法查询用户信息并返回到 Controller

3.通过 model 对象将用户信息放入 request,跳转 jsp

3. jsp 中通过 EL 表达式获取用户信息

6.1. 添加 service 层代码



IUserService 接口:

```
public interface IUserService {

   public User findUserById(Long id);
}
```

UserServiceImpl 接口实现类:

```
@Service("userService")
public class UserServiceImpl implements IUserService{

    @Autowired
    private UserMapper userMapper;

    @Override
    public User findUserById(Long id) {
        return userMapper.selectUserById(id);
    }

}
```

6.2. 配置注解扫描

```
<!-- 开启注解扫描,扫描service -->
<context:component-scan base-package="cn.itcast.ssm.service" />
```

6.3. 开启 spring 事物管理

```
<!-- 配置事物管理器 -->
<bean id="transactionManager"
    class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource" />
</bean>


<!-- 配置事物通知 -->
<tx:advice id="txAdvice">
    <tx:attributes>
        <tx:method name="save*" />
        <tx:method name="update*" />
        <tx:method name="delete*" />
        <tx:method name="find*" read-only="true" />
    </tx:attributes>
</tx:advice>


<!-- 配置事物aop -->
<aop:config>
    <aop:advisor advice-ref="txAdvice" pointcut="bean(*Service)"/>
</aop:config>
```

通过测试用例测试 UserService 方法

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations =
{ "classpath:spring/applicationContext.xml",
      "classpath:spring/applicationContext-mybatis.xml" })
public class UserServiceImplTest extends UserServiceImpl {

   @Autowired
   private IUserService userService;

   @Test
   public void testFindUserById() {
      System.out.println(userService.findUserById(1L));
   }

}
```

6.4. 编写 UserController 方法

```
@Controller
@RequestMapping("user")
public class UserController {

    @Autowired
    private IUserService userService;

    @RequestMapping("showuser")
    public String toUser(Model model, @RequestParam("id") Long id) {

        User user = userService.findUserById(id);

        model.addAttribute("user", user);

        return "user";
    }
}
```

在 jsp 页面通过 el 表达式获取用户信息:

```
<form>
    <table>
        <tr>
            <td>ID</td>
            <td>用户名</td>
            <td>姓名</td>
            <td>年龄</td>
            <td>生日</td>
            <td>创建日期</td>
            <td>更新日期</td>
        </tr>
        <tr>
            <td>${user.id }</td>
            <td>${user.userName}</td>
            <td>${user.name }</td>
            <td>${user.age }</td>
            <td>${user.birthday }</td>
            <td>${user.created }</td>
            <td>${user.updated }</td>
        </tr>
    </table>
</form>
```

localhost:8088/user/showuser?id=4

ID	用户名	姓名	年龄	生日	创建日期	更新日期
4	zhangwei	张伟	20	Thu Sep 01 00:00:00 CDT 1988	Fri Sep 19 16:56:04 CST 2014	Fri Sep 19 16:56:04 CST 2

日期格式通过 fmt 标签可转换成喜欢的格式(课后)

7.通过 ID 删除用户信息

UserController 方法:

```
@RequestMapping("deleteuser")
public String deleteuser(@RequestParam("id") Long id) {

        userService.deleteUserById(id);

    return "user";
}
```

UserService 方法:

```
@Override
public void deleteUserById(Long id) {
    userMapper.deleteUserById(id);
}
```

UserMapper 方法:

```
public void deleteUserById(@Param("id")Long id);
```

UserMapper.xml 语句

```
<delete id="deleteUserById" >
    delete from tb_user where id = #{id}
</delete>
```

:

URL: http://localhost:8088/user/deleteuser?id=10

8. Maven 项目的分解与聚合

目的:将 ssm 项目分解为多个模块,通过一个父工程统一配置管理多个子模块的通用配置文件和依赖

抽取子模块分别为:

ssm_parent:父工程:将通用 jar 包坐标统一配置在父工程中,聚合子模块

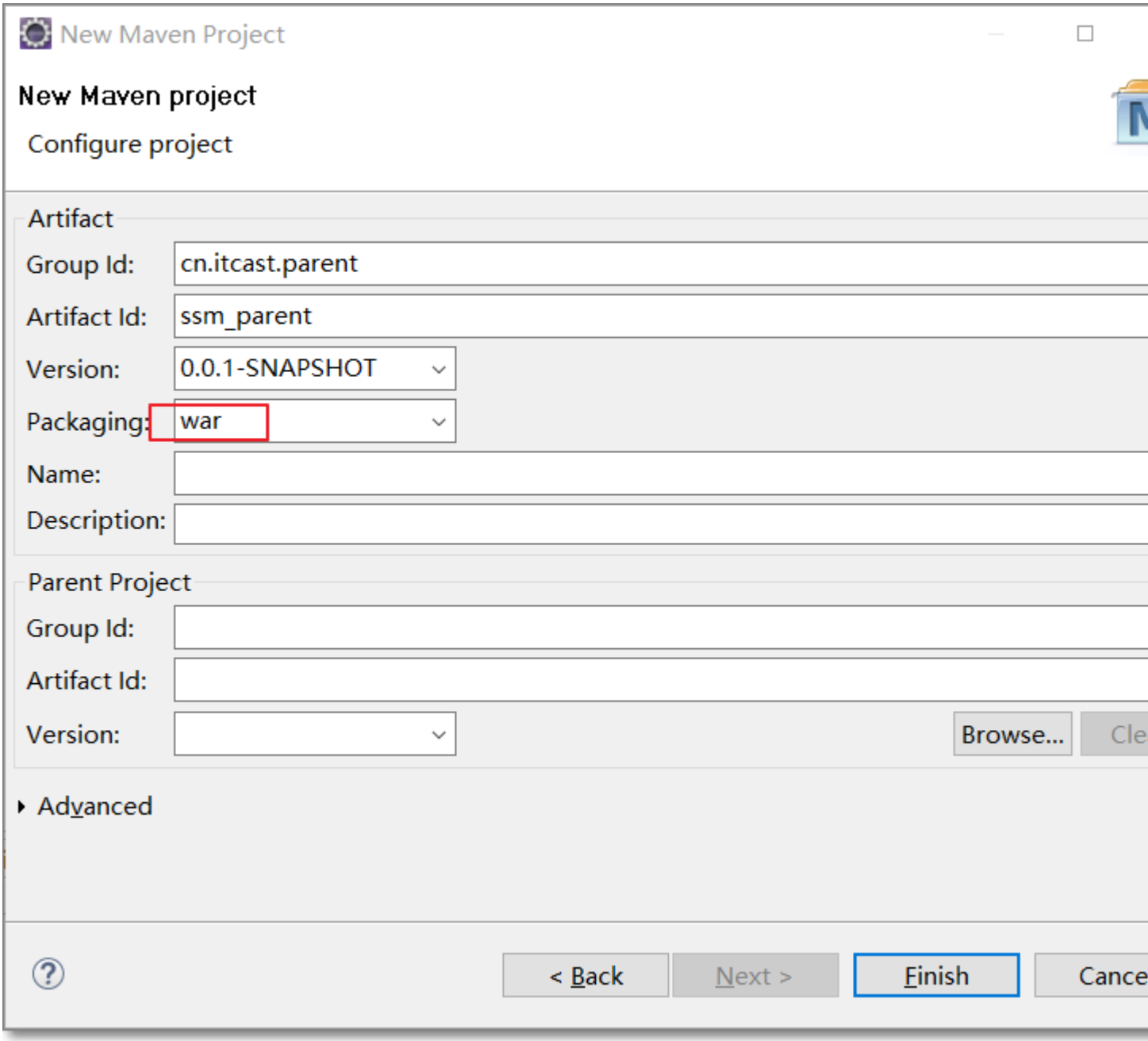
ssm_dao:管理持久层(数据访问)的相关代码和配置文件(jar)

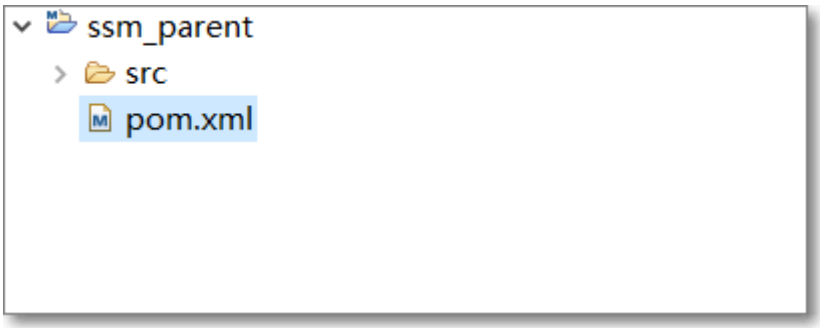
ssm_service: 管理业务层的相关代码和配置文件(jar)

ssm_web:管理表现层相关代码和配置文件(war)

3.1. 创建父工程(ssm_parent)

跳过骨架选择:





父工程中不需要编写代码,它的作用是管理所有子工程中的通用依赖版本,聚合子模块(子模块按需获取依赖)

将之前整合的 pom.xml 文件中的依赖拷贝到父工程 pom.xml 中

```
<!-- 集中定义依赖版本号 -->
  <properties>
      <junit.version>4.12</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>

<dependencyManagement>
      <dependencies>
          <!-- 单元测试 -->
          <dependency>
              <groupId>junit</groupId>
              <artifactId>junit</artifactId>
              <version>${junit.version}</version>
              <scope>test</scope>
          </dependency>

          <!-- Spring -->
          <dependency>
              <groupId>org.springframework</groupId>
              <artifactId>spring-context</artifactId>
              <version>${spring.version}</version>
          </dependency>
          <dependency>
              <groupId>org.springframework</groupId>
              <artifactId>spring-beans</artifactId>
              <version>${spring.version}</version>
          </dependency>
          <dependency>
              <groupId>org.springframework</groupId>
```

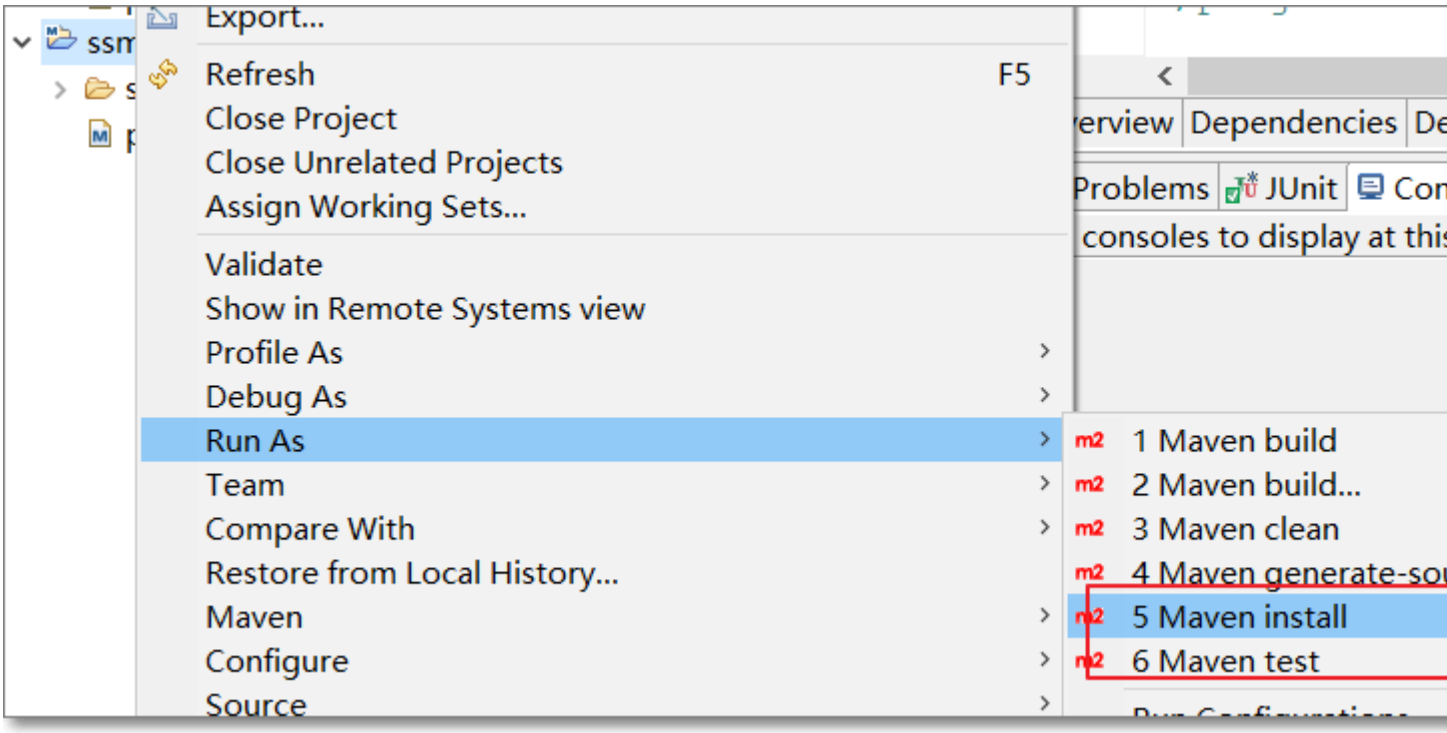
```
            <artifactId>spring-webmvc</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-jdbc</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-aspects</artifactId>
            <version>${spring.version}</version>
        </dependency>


        <!-- Mybatis -->
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis</artifactId>
            <version>${mybatis.version}</version>
        </dependency>
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis-spring</artifactId>
            <version>${mybatis.spring.version}</version>
        </dependency>
        <dependency>
            <groupId>com.github.miemiedev</groupId>
            <artifactId>mybatis-paginator</artifactId>
            <version>${mybatis.paginator.version}</version>
        </dependency>

        <!-- MySql -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>${mysql.version}</version>
        </dependency>

        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-log4j12</artifactId>
            <version>${slf4j.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>

        <!-- 连接池 -->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid</artifactId>
            <version>${druid.version}</version>
        </dependency>

        <!-- httpclient -->
        <dependency>
            <groupId>org.apache.httpcomponents</groupId>
            <artifactId>httpclient</artifactId>
            <version>${httpclient.version}</version>
        </dependency>

        <!-- JSP相关 -->
        <dependency>
            <groupId>jstl</groupId>
            <artifactId>jstl</artifactId>
            <version>${jstl.version}</version>
        </dependency>
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>servlet-api</artifactId>
            <version>${servlet-api.version}</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>jsp-api</artifactId>
            <version>${jsp-api.version}</version>
            <scope>provided</scope>
        </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>
</dependencyManagement>

<build>
    <finalName>${project.artifactId}</finalName>
    <plugins>
        <!-- 资源文件拷贝插件 -->
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-resources-plugin</artifactId>
            <version>2.7</version>
            <configuration>
                <encoding>UTF-8</encoding>
            </configuration>
        </plugin>
        <!-- java编译插件 -->
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.2</version>
            <configuration>
                <source>1.7</source>
                <target>1.7</target>
                <encoding>UTF-8</encoding>
            </configuration>
        </plugin>
    </plugins>
    <pluginManagement>
```

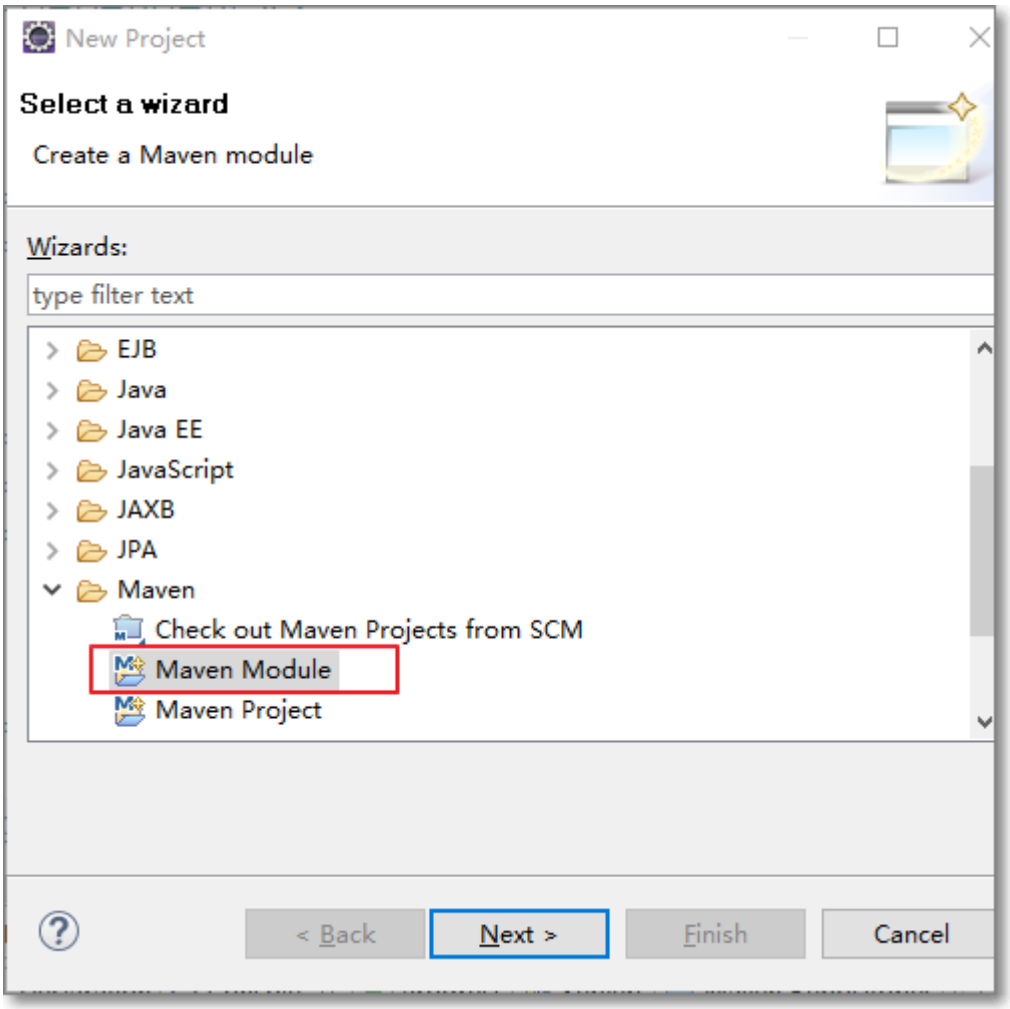
```
        <plugins>
            <!-- 配置Tomcat插件 -->
            <plugin>
                <groupId>org.apache.tomcat.maven</groupId>
                <artifactId>tomcat7-maven-plugin</artifactId>
                <version>2.2</version>
            </plugin>
        </plugins>
    </pluginManagement>
</build>
```

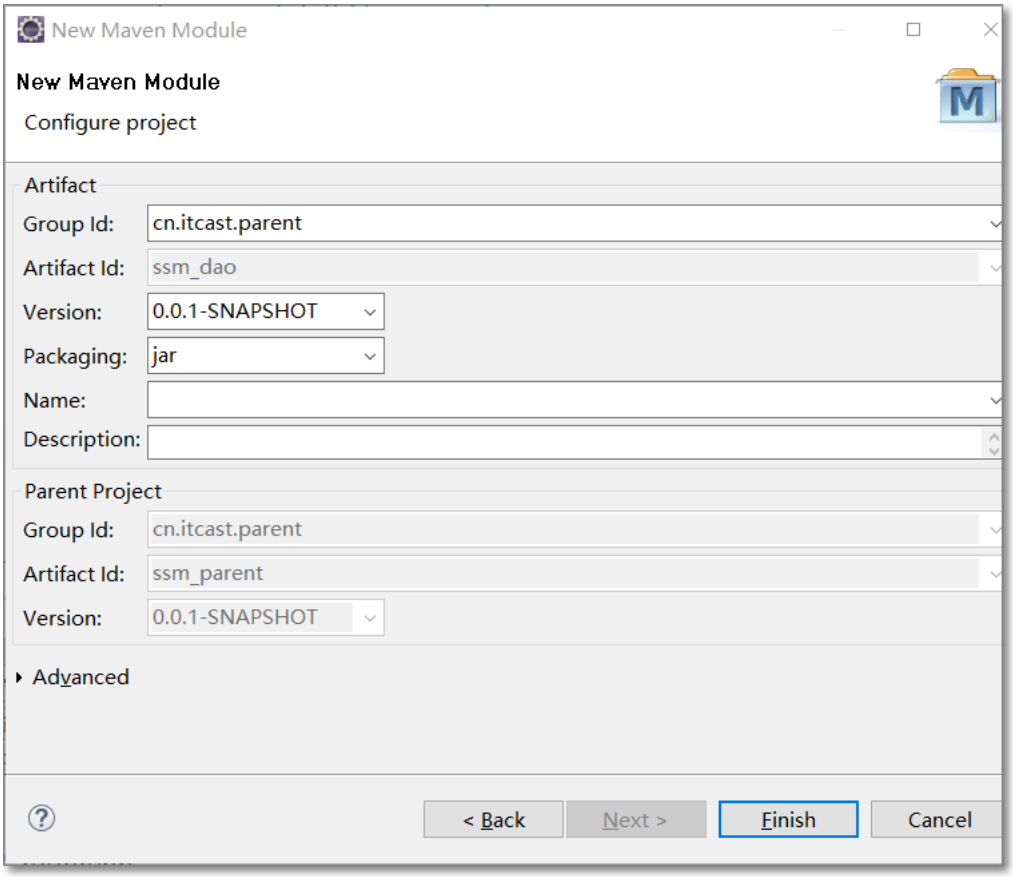
将父工程安装到本地仓库:



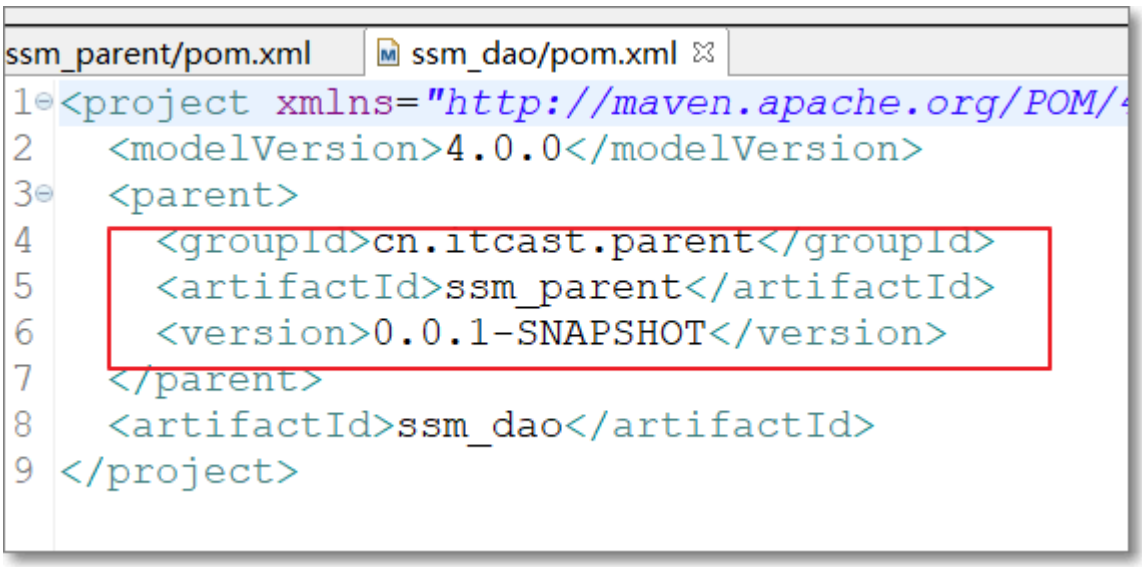
3.2. 创建子模块(ssm_dao)

选择 Maven Module:


New Project
Select a wizard
Create a Maven module
Wizards:
type filter text
EJB
Java
Java EE
JavaScript
JAXB
JPA
Maven
Check out Maven Projects from SCM
Maven Module
Maven Project
< Back
Next >
Finish
Cancel



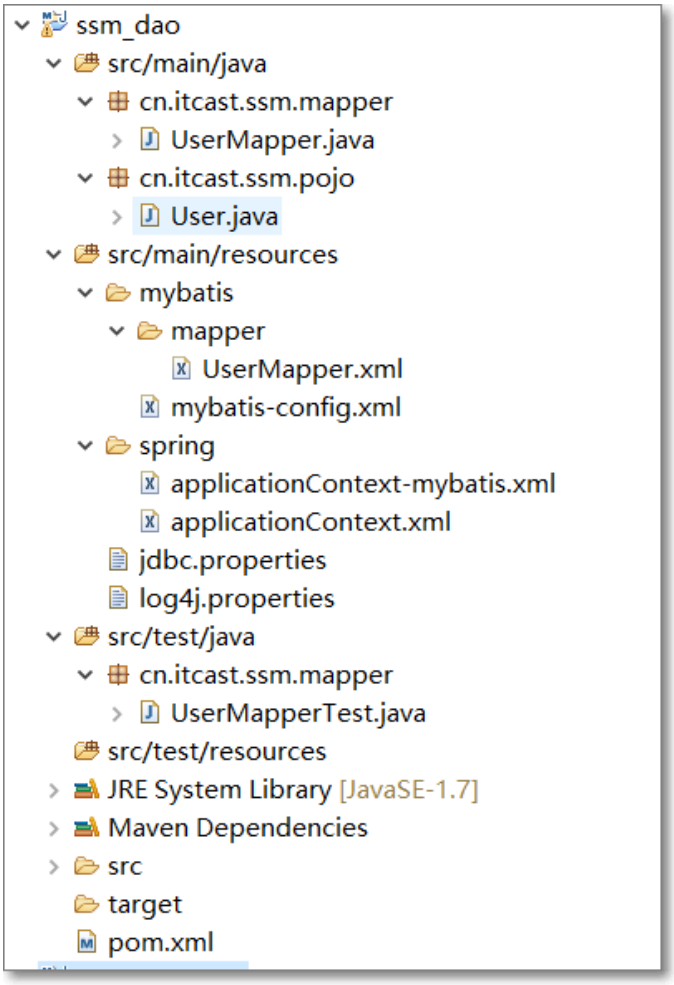
在子模块中自动关联了父工程:



在父工程中自动关联了子模块:

```
<modules>
    <module>ssm_dao</module>
</modules>
```

将 ssm_integrate 项目中的 dao 层相关代码和配置文件移植到子模块中:



引入 dao 层所需要的依赖:

```
<dependencies>
        <!-- spring相关 -->
        <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>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-test</artifactId>
        <version>4.3.13.RELEASE</version>
    </dependency>

    <!-- 单元测试 -->
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <scope>test</scope>
    </dependency>

    <!-- 连接池 -->
    <dependency>
        <groupId>com.alibaba</groupId>
        <artifactId>druid</artifactId>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
    </dependency>

    <!-- 日志 -->
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-log4j12</artifactId>
    </dependency>

    <!-- mybatis -->
    <dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis</artifactId>
    </dependency>
    <dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis-spring</artifactId>
    </dependency>
```

```
    </dependencies>
```

测试 mapper 方法:

```
2018-05-28 10:05:36,946 [main] [cn.itcast.ssm.mapper.UserMapper.selectUserById]-[DEBUG] ==>  Preparing: selec
2018-05-28 10:05:36,967 [main] [cn.itcast.ssm.mapper.UserMapper.selectUserById]-[DEBUG] ==> Parameters: 1(Lon
2018-05-28 10:05:36,986 [main] [cn.itcast.ssm.mapper.UserMapper.selectUserById]-[DEBUG] <==      Total: 1
2018-05-28 10:05:36,986 [main] [org.mybatis.spring.SqlSessionUtils]-[DEBUG] Closing non transactional SqlSess
2018-05-28 10:05:36,987 [main] [org.springframework.jdbc.datasource.DataSourceUtils]-[DEBUG] Returning JDBC C
User [id=1, userName=zhangsan, password=123456, name=张三, age=30, sex=1, birthday=Wed Aug 08 00:00:00 CST 1984
```

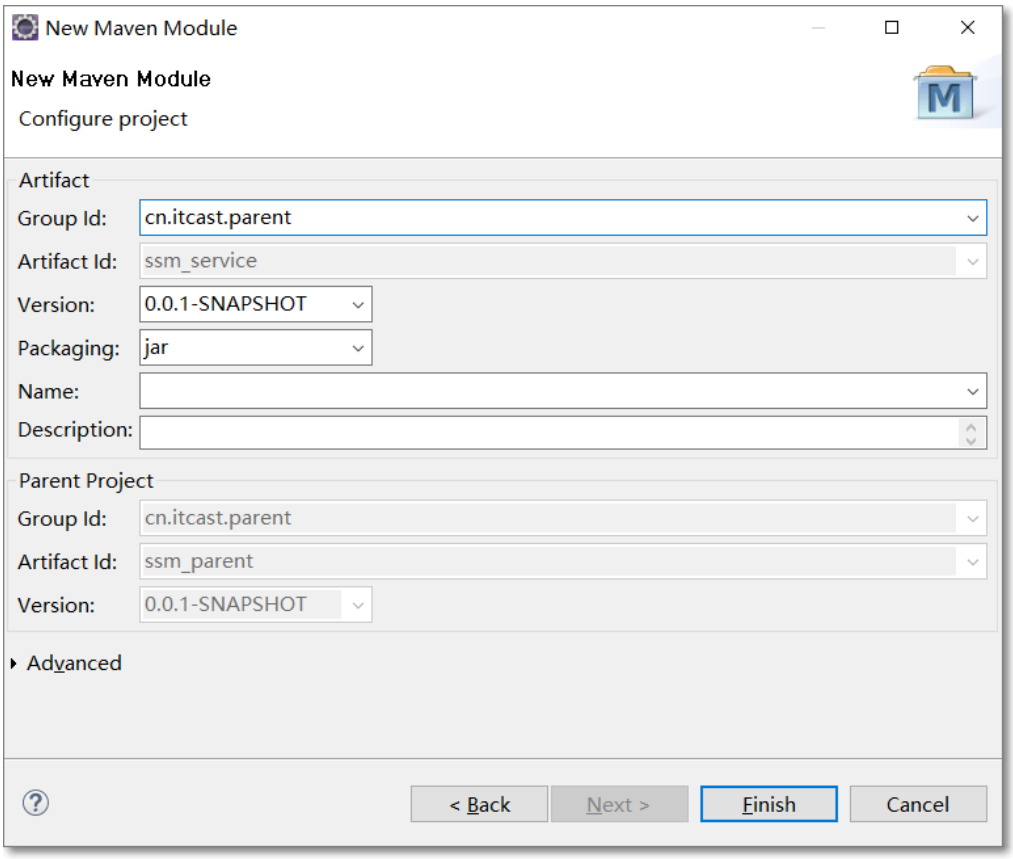
右击项目,将 dao 层打包安装到本地仓库:

Install 到本地仓库的目的是为了在其他项目中引用，本地仓库有了之后可以把 eclipse 中的项目关闭或者移出工作区，因为同一个工作区中项目也采用“就近原则”

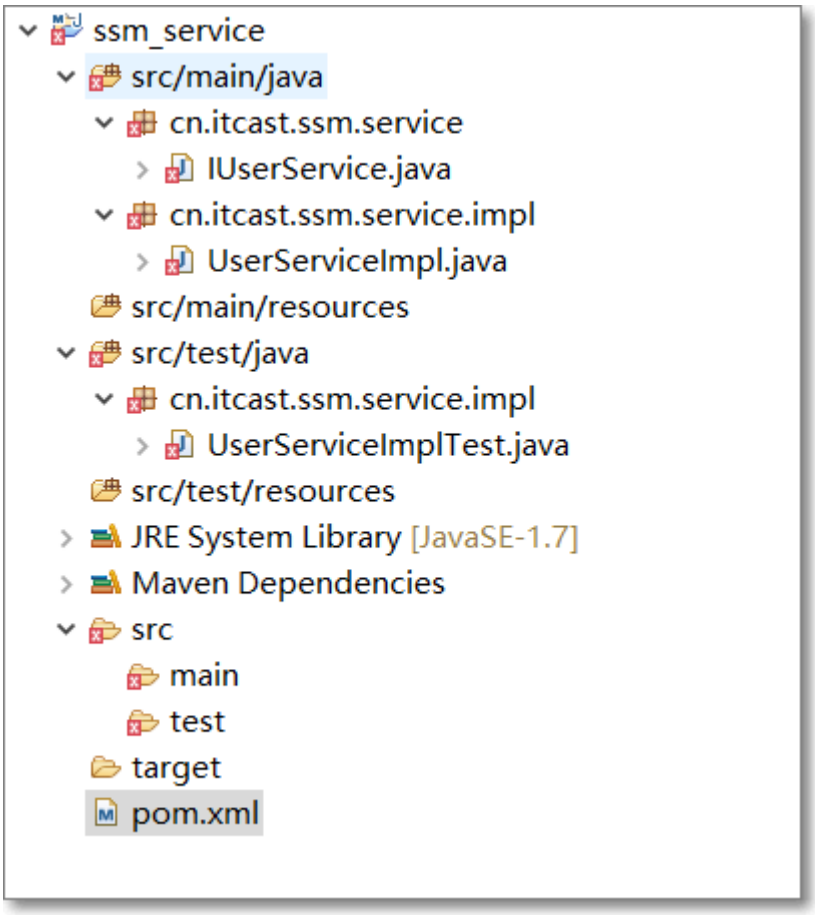
也就是说，工作区中有的话先依赖工作区中的项目，没有才会去本地仓库中查找。

3.3. 创建子模块(ssm_service)

创建 service 曾项目的步骤跟 dao 层相同



Service 层只需关注业务代码即可



但 Service 层需要依赖 dao 层的相关代码,所以在 service 模块需要添加 dao 层的依赖:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xm
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>cn.itcast.parent</groupId>
    <artifactId>ssm_parent</artifactId>
    <version>0.0.1-SNAPSHOT</version>
  </parent>
  <artifactId>ssm_service</artifactId>
  <dependencies>
    <dependency>
        <groupId>cn.itcast.parent</groupId>
        <artifactId>ssm_dao</artifactId>
        <version>0.0.1-SNAPSHOT</version>
    </dependency>
  </dependencies>
</project>
```

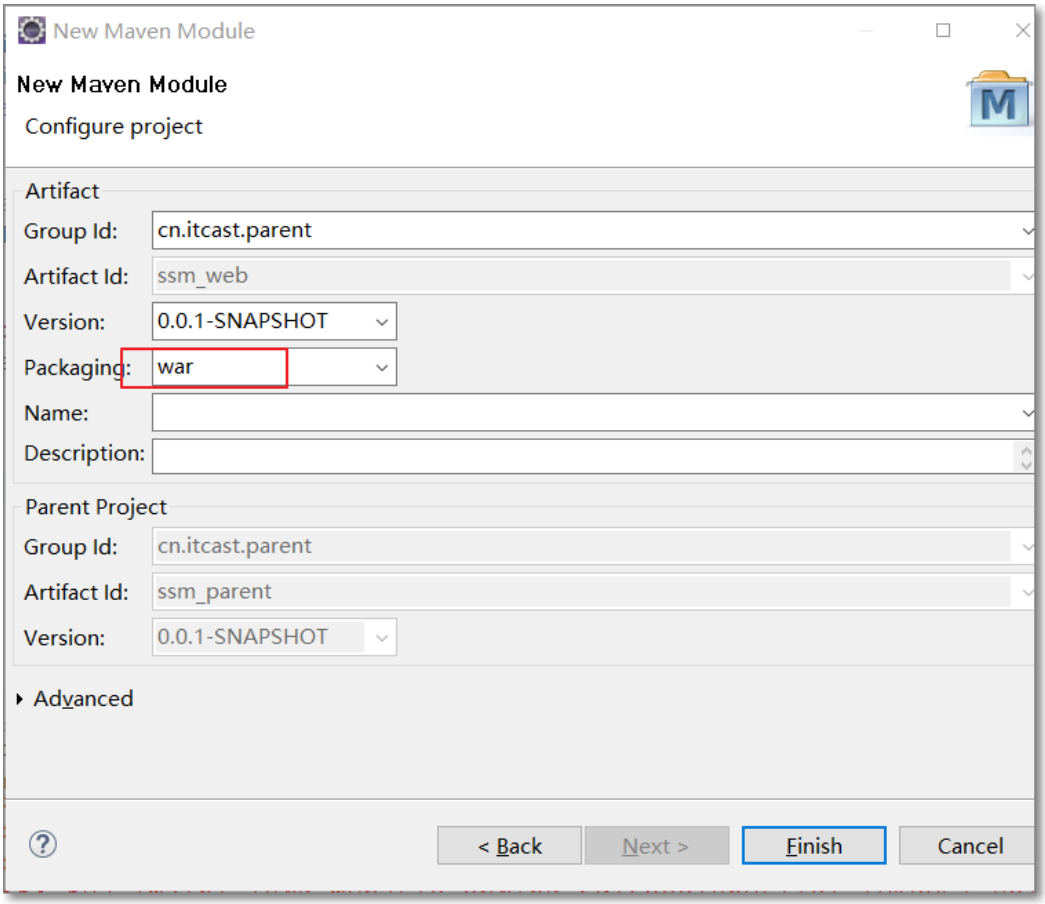
提示: 添加依赖之前需要将 dao 模块安装到本地仓库

测试功能:

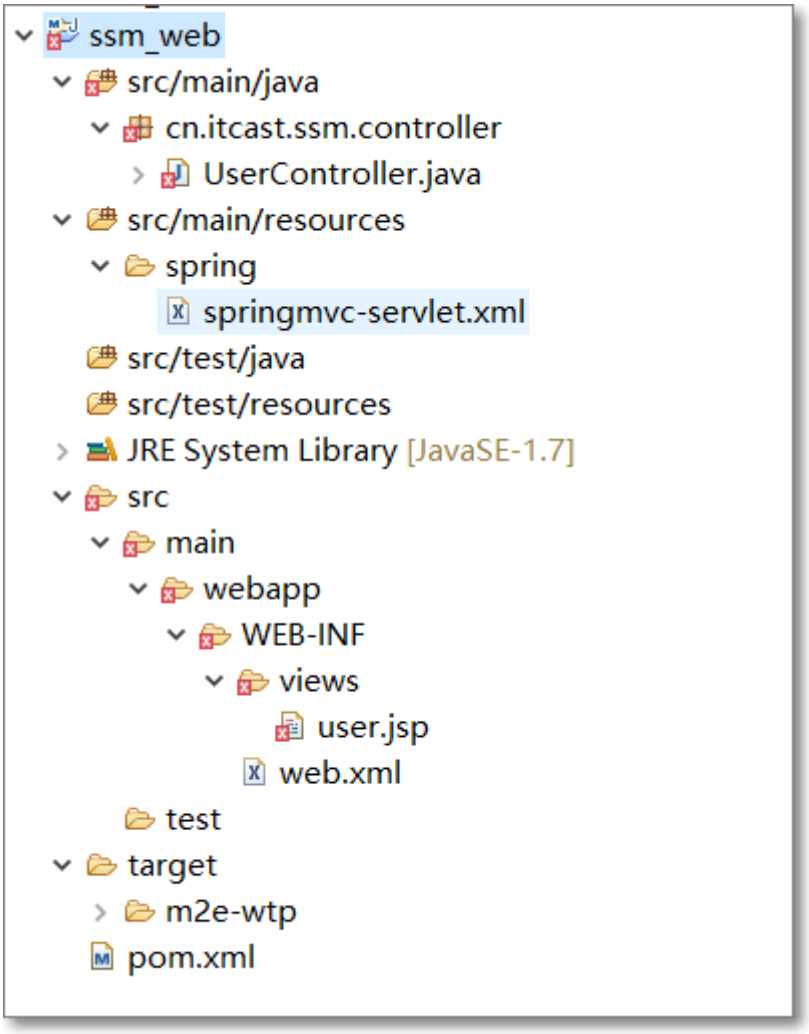
```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = { "classpath:spring/applicationContext.xml",
        "classpath:spring/applicationContext-mybatis.xml" })
public class UserServiceImplTest extends UserServiceImpl {

    @Autowired
    private IUserService userService;

    @Test
    public void testFindUserById() {
        System.out.println(userService.findUserById(1L));
    }

}
```

3.4. 创建子模块(ssm_web)

注意 web 层是个 web 项目,打包方式为 war



将 web 项目补全:



引入 service 层和 jsp 相关依赖:

```
<dependencies>
        <dependency>
            <groupId>cn.itcast.parent</groupId>
            <artifactId>ssm_service</artifactId>
            <version>0.0.1-SNAPSHOT</version>
        </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>
                <path>/</path>
                <port>8088</port>
            </configuration>
        </plugin>
    </plugins>
</build>
```

发布项目并测试相关功能:

启动报错:

```
五月28, 2018 10:47:28 上午org.apache.catalina.core.StandardContext loadOnStartup
严重: Servlet  threw load() exception
org.springframework.beans.factory.NoSuchBeanDefinitionException: No qualifying be
        at org.springframework.beans.factory.support.DefaultListableBeanFactory.r
        at org.springframework.beans.factory.support.DefaultListableBeanFactory.c
        at org.springframework.beans.factory.support.DefaultListableBeanFactory.r
        at org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPo
        at org.springframework.beans.factory.annotation.InjectionMetadata.inject(
        at org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPo
        at org.springframework.beans.factory.support.AbstractAutowireCapableBeanF
        at org.springframework.beans.factory.support.AbstractAutowireCapableBeanF
```

分析原因：web.xml 中的 spring 监听器只能加载到当前项目 applicationContext.xml 文件无法拿到 dao，service

的配置文件

配置 web.xml 的时候需要修改配置添*号,目的是让 spring 监听器读取到其他模块的配置文件

```
<!--Spring监听器-->
<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener
</listener>
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath*:spring/applicationContext*.xml</param-value>
</context-param>
```

测试访问: http://localhost:8088/user/showuser?id=1

ID	用户名	姓名	年龄	生日	创建日期	更新日期
1	zhangsan	张三	30	Wed Aug 08 00:00:00 CST 1984	Fri Sep 19 16:56:04 CST 2014	Sun Sep

5. 总结

1. 使用 maven 整合 ssm 工程
    1.1. 注意：整合思路
2. 拆分
    2.1. 思想：dao 工程：只做数据访问 service 只负责业务逻辑 web 只负责 Controller
3. 聚合
    3.1. 通过父工程聚合子模块

作业：案例：将 pojo 从 ssm_dao 中拆除成一个独立的模块