# 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

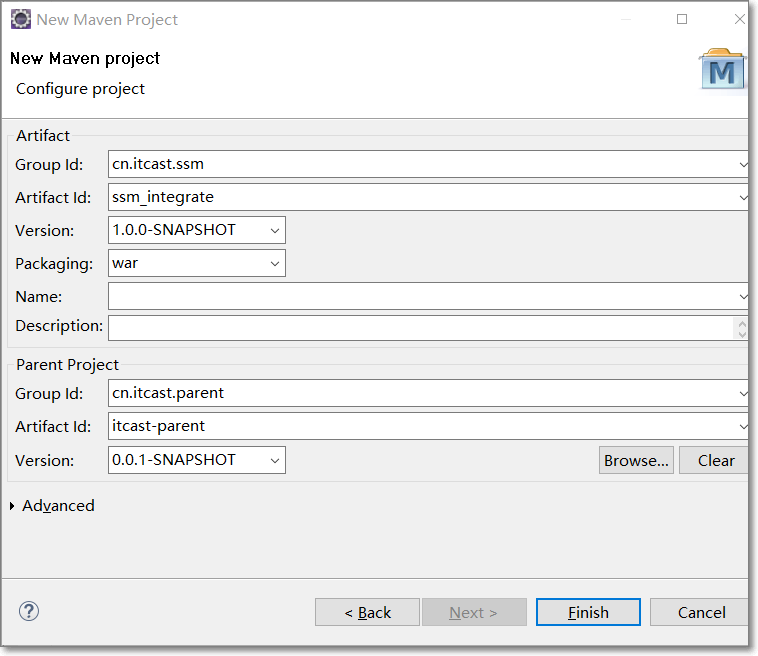
## 使用技术

Spring + SpringMVC + Mybatis

数据库：mysql

# 创建工程---环境整理

## 创建工程



## 引入依赖

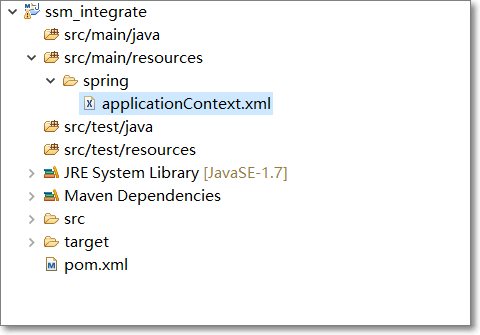
参照itcast-parent工程的pom.xml，在pom.xml中引入所需依赖

|  |
| --- |
| <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> |

# 配置spring环境

## 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}"* />

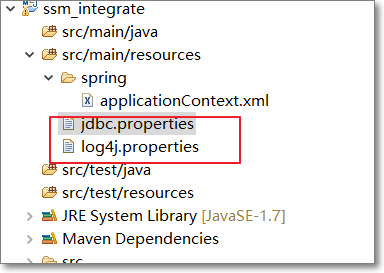
<property name=*"password"* value=*"${jdbc.password}"* />

</bean>

如上黄色标识。于是，又需要引入jdbc.properties资源文件

## 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

# 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。

## 配置applicationContext-mybatis.xml

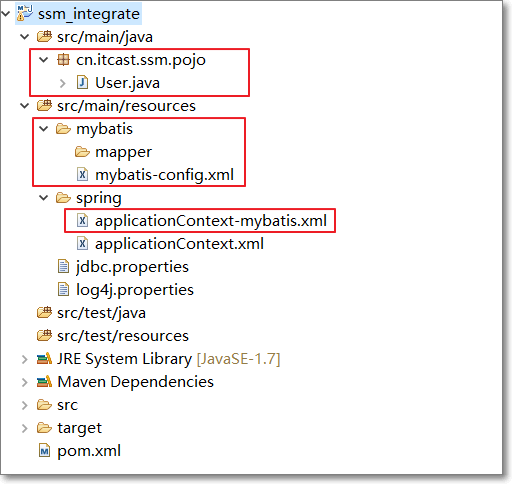
思路：spring整合mybatis，能整合mybatis的什么？

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>

-->

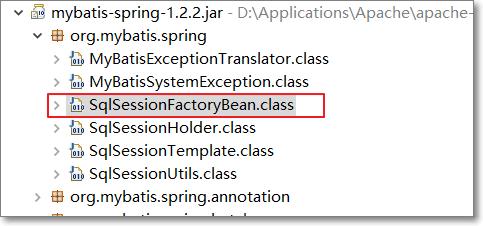
</configuration>

### 构建SqlSessionFactory

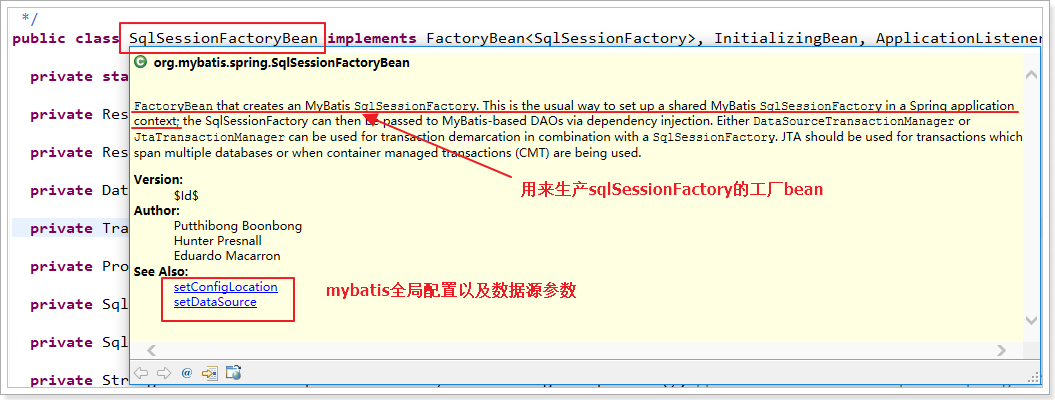
SqlSessionFactory的构建过程，在mybatis中是比较麻烦的。当mybatis遇到spring之后，这个问题变的非常简单了。

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：



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>

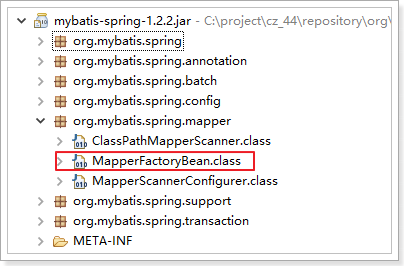
</beans>

### 配置mapper接口

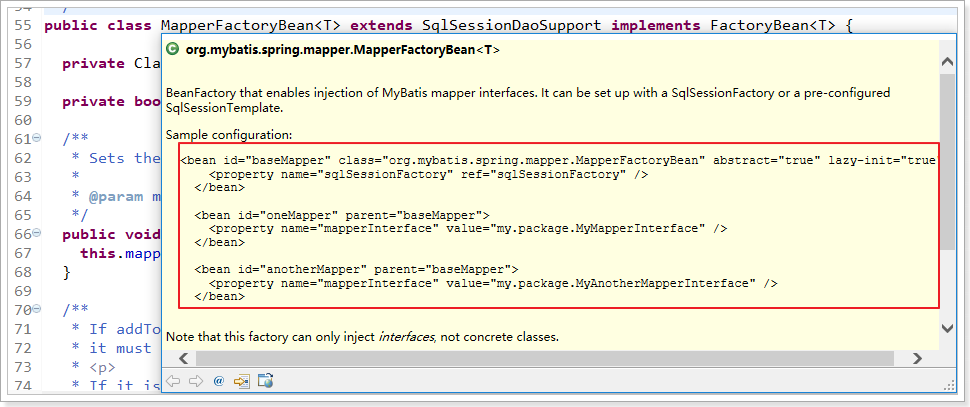
既然sqlSessionFactory交给spring管理了，那么mybatis的mapper接口的动态代理实现能不能也交给spring进行管理呢？

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：



2、 参照mybatis整合spring的官方文档：



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"*>

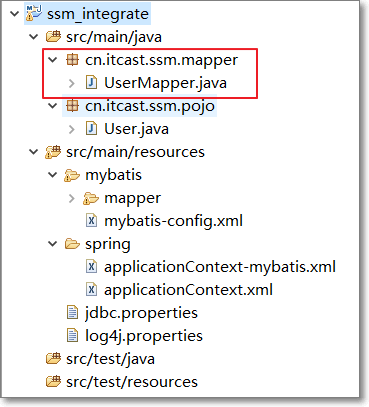
<property name=*"mapperInterface"* value=*"cn.itcast.ssm.mapper.UserMapper"* />

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

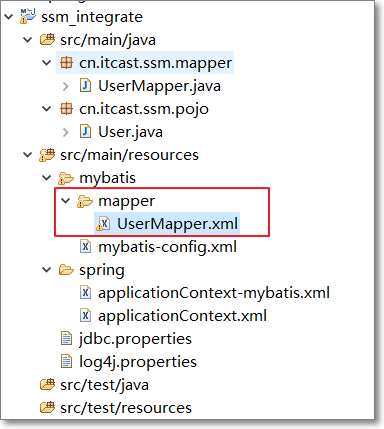
</bean>

</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:



## 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() {

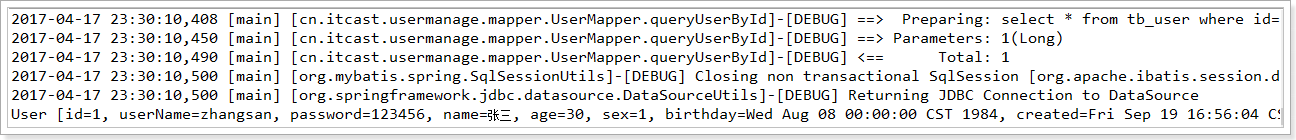
User user = userMapper.selectUserById(1L);

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

}

}

控制台打印出用户信息，说明整合成功，日志输出：



## 优化整合程序

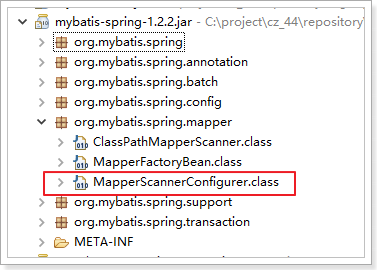
那些配置需要优化：

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

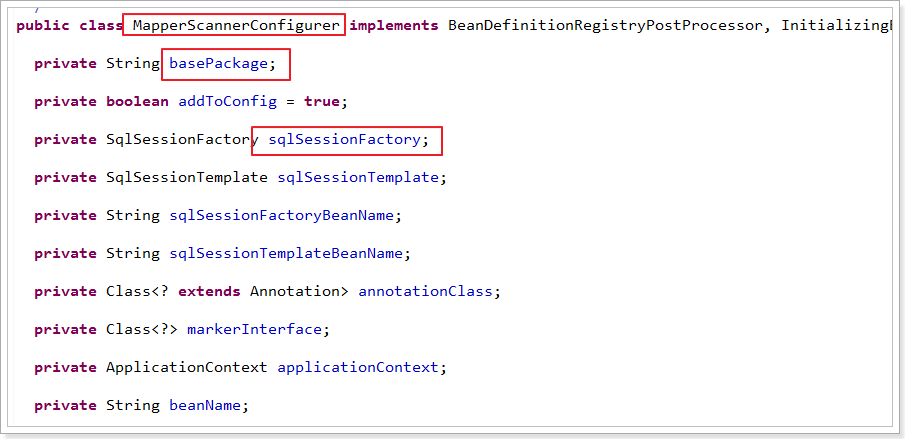
### 使用Mapper接口扫描

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。

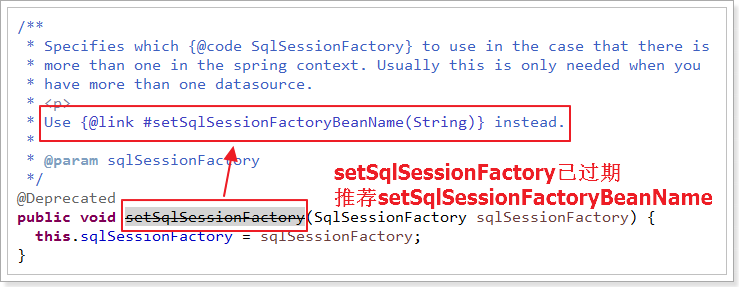
在mybatis-spring的整合包中，提供了MapperScannerConfigurer接口扫描类：



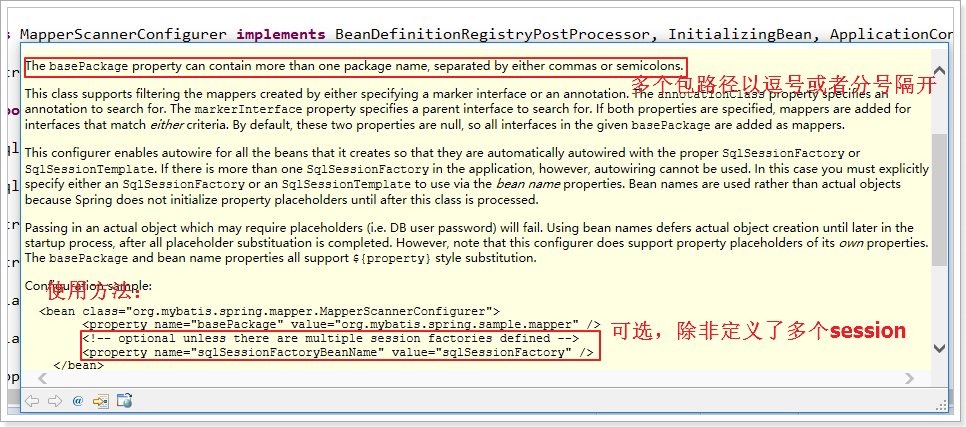
类中有以下属性



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



查看MapperScannerConfigurer的注释：



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"*>

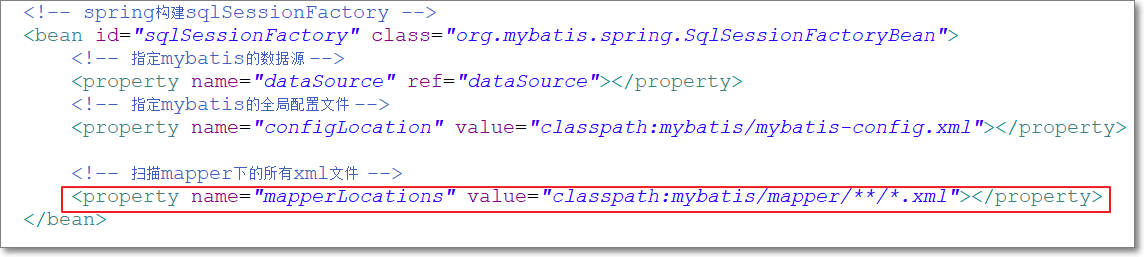
<property name=*"basePackage"* value=*"cn.itcast.ssm.mapper"* />

</bean>

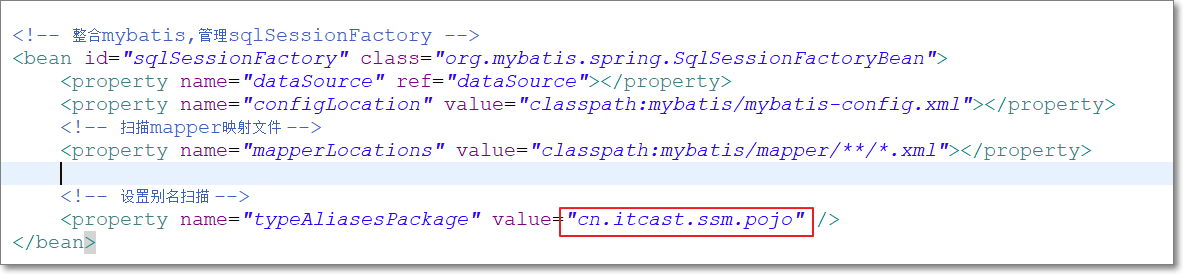
### Mapper.xml交给spring

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。



### 别名扫描交给spring



## 整合的终极配置

### 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"*>  <!-- 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"*></property>  <!-- 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> |

### 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（多个单词时，以驼峰命名） -->  <setting name=*"mapUnderscoreToCamelCase"* value=*"true"* />  </settings>  </configuration> |

# 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

## 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>

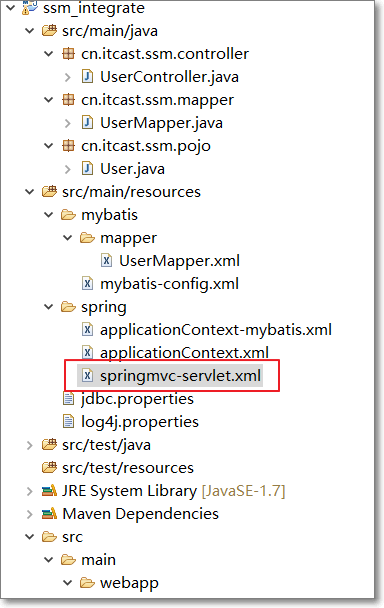
<welcome-file-list>

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

</welcome-file-list>

</web-app>

## 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"*>

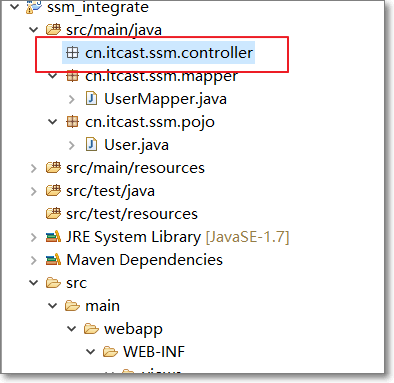
<property name=*"prefix"* value=*"/WEB-INF/views/"*></property>

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

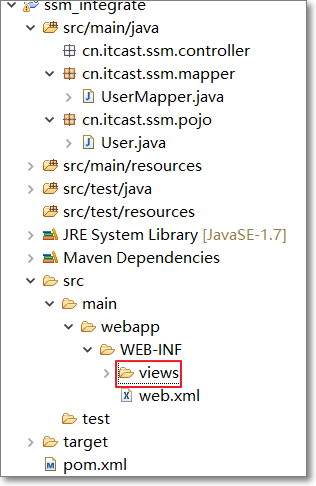
</bean>

</beans>

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



* 1. 创建视图解析器对应的视图目录



## 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>

## 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")

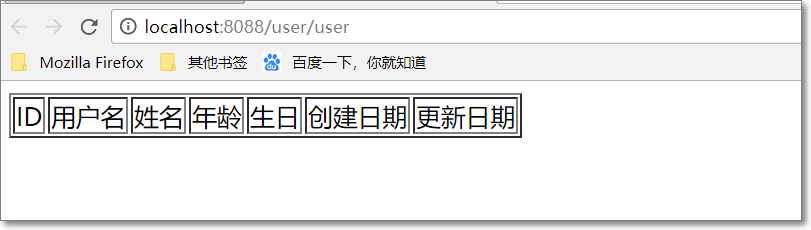
**public** String toUser(){

**return** "user";

}

}

启动tomcat之后的访问效果：



# 根据id查询用户信息并在页面显示

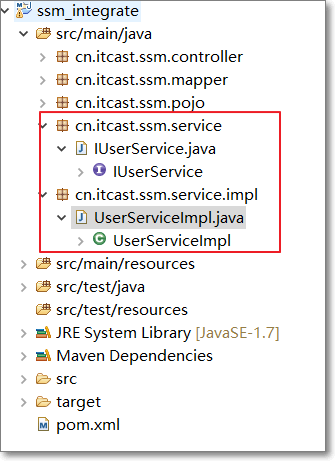
1.通过访问url传递用户ID 访问对应的Controller的方法

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

## 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

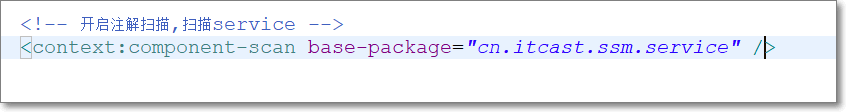
**public** User findUserById(Long id) {

**return** userMapper.selectUserById(id);

}

}

## 配置注解扫描



## 开启spring事物管理



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));

}

}

## 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);

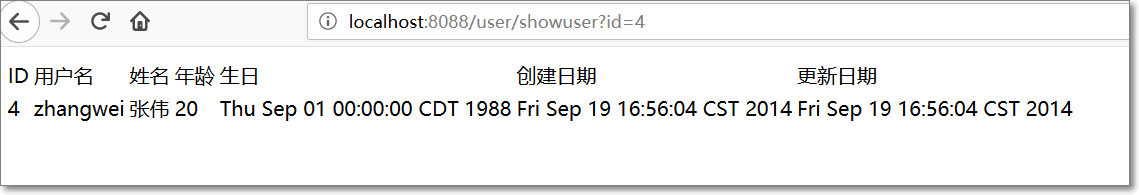
**return** "user";

}

}

在jsp页面通过el表达式获取用户信息:

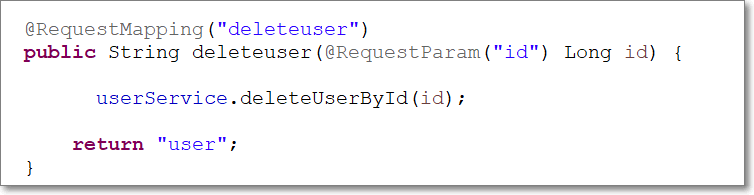




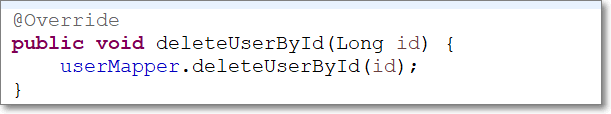
日期格式通过fmt标签可转换成喜欢的格式(课后)

# 通过ID删除用户信息

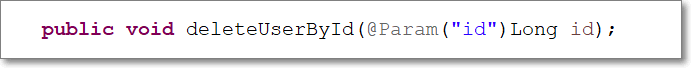
UserController方法:



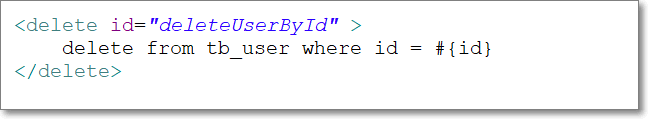
UserService方法:



UserMapper方法:



UserMapper.xml语句

:

URL: <http://localhost:8088/user/deleteuser?id=1>0

# 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**

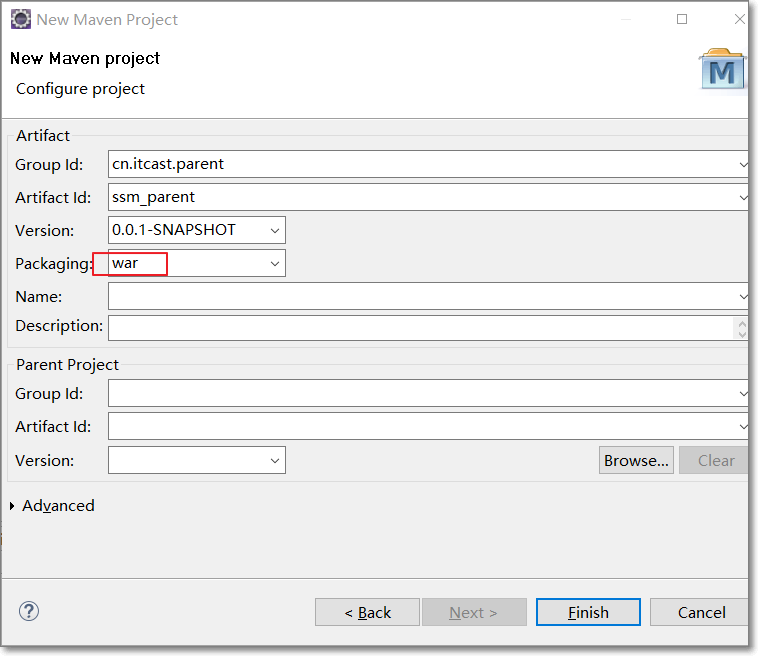
**ssm\_dao:管理持久层(数据访问)的相关代码和配置文件(jar)**

**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)**

## 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>

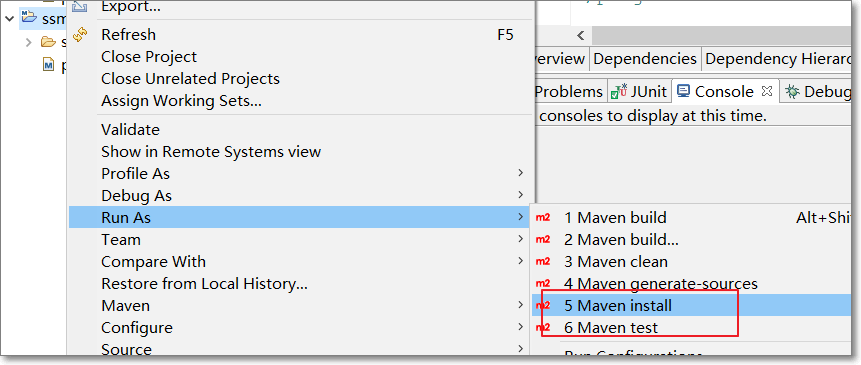
</plugin>

</plugins>

</pluginManagement>

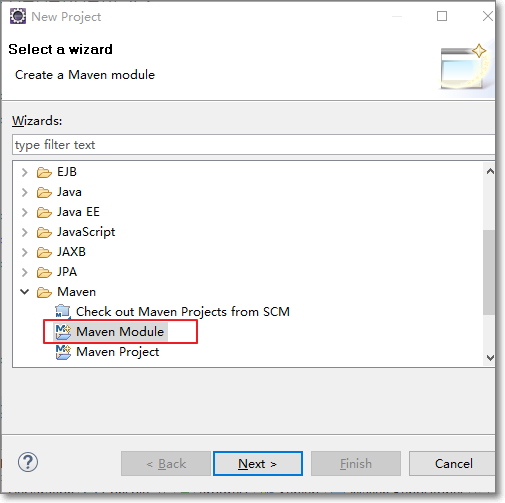
</build>

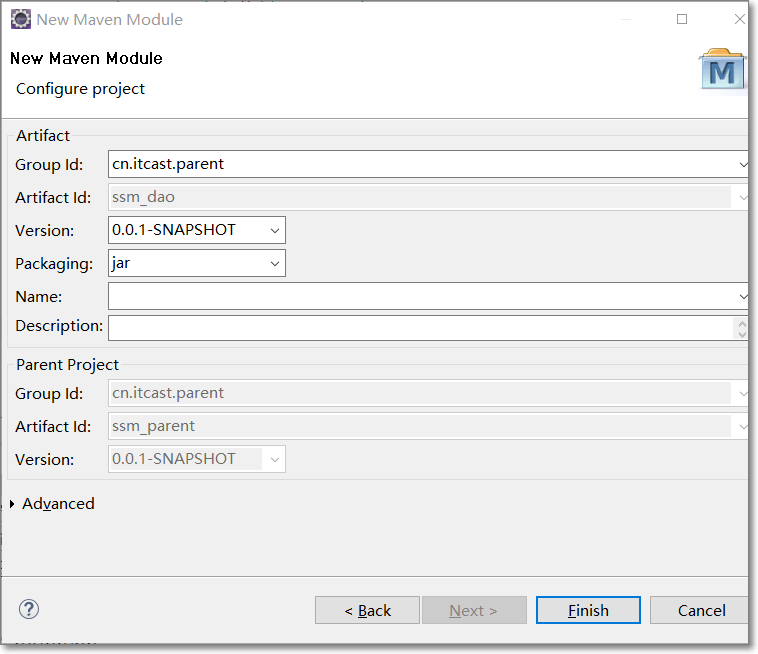
将父工程安装到本地仓库:



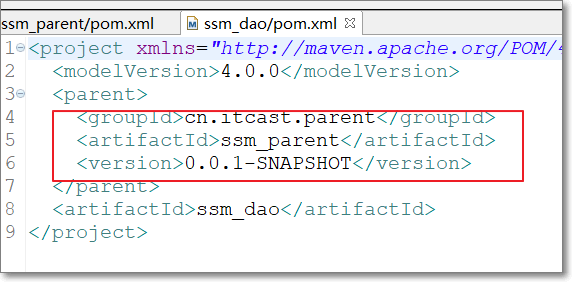
## 创建子模块(ssm\_dao)

选择Maven Module:





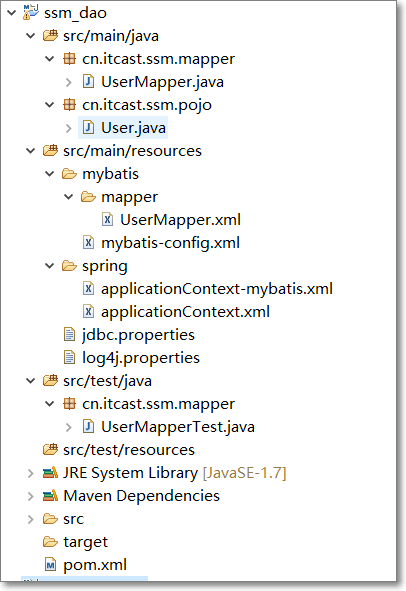
在子模块中自动关联了父工程：



在父工程中自动关联了子模块:



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>

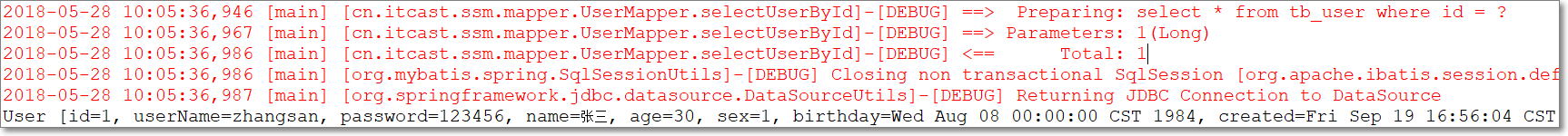
<groupId>org.mybatis</groupId>

<artifactId>mybatis-spring</artifactId>

</dependency>

</dependencies>

测试mapper方法:



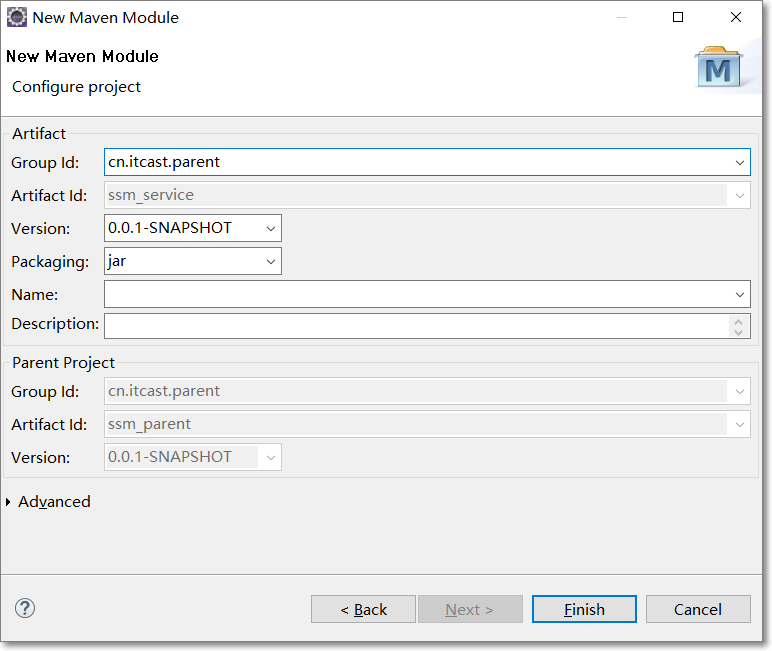
右击项目,将dao层打包安装到本地仓库:

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”

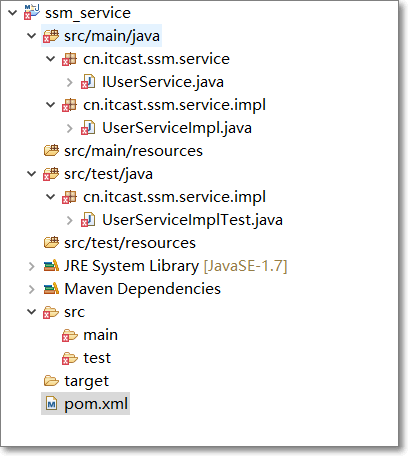
也就是说，工作区中有的话先依赖工作区中的项目，没有才会去本地仓库中查找。

## 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

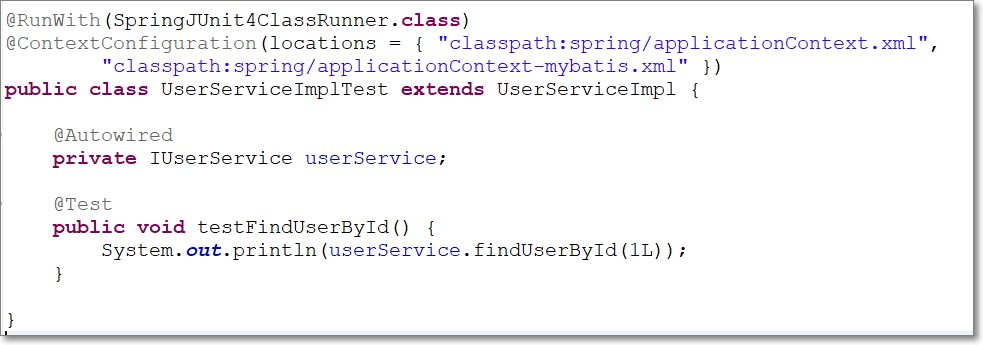


但Service层需要依赖dao层的相关代码,所以在service模块需要添加dao层的依赖:



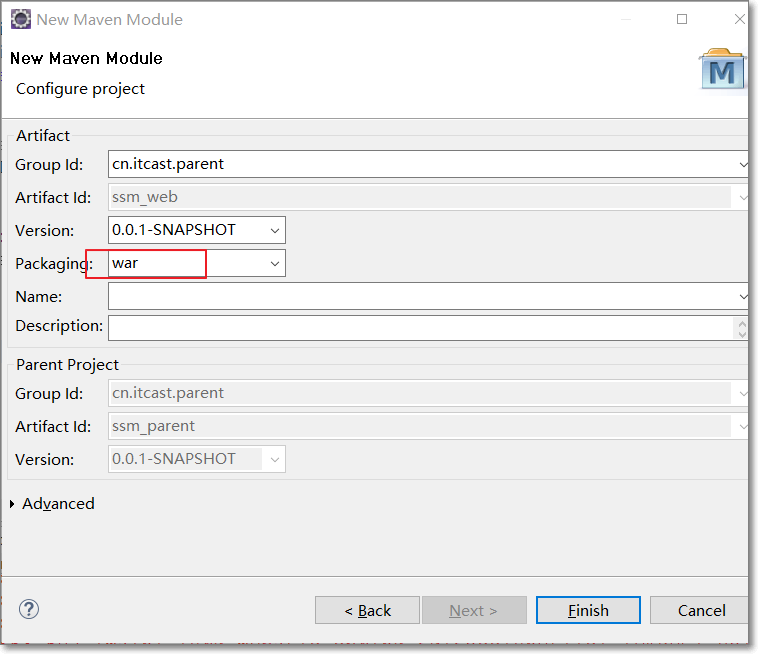
提示: 添加依赖之前需要将dao模块安装到本地仓库

测试功能:

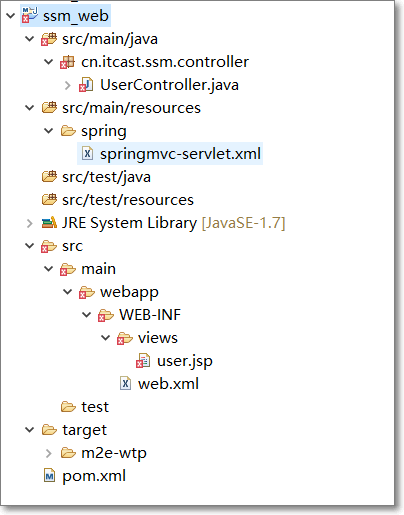


## 创建子模块(ssm\_web)

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>

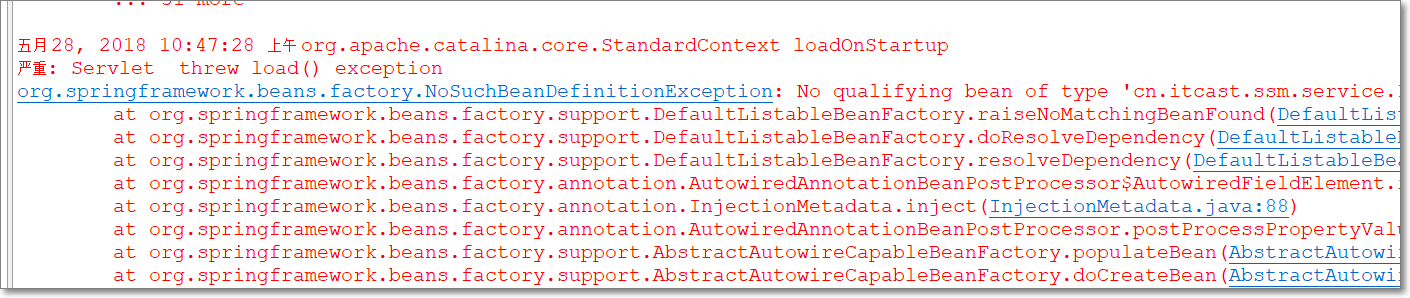
</plugin>

</plugins>

</build>

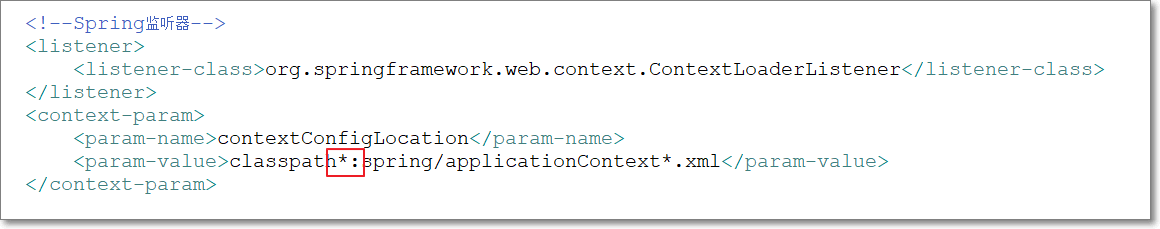
发布项目并测试相关功能:

启动报错:

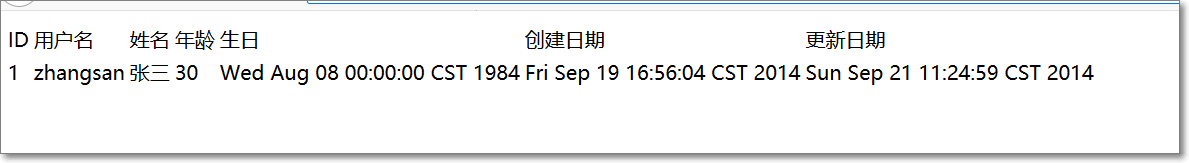


分析原因： web.xml中的spring监听器只能加载到当前项目applicationContext.xml文件无法拿到dao，service的配置文件

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



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



# 总结

1. 使用maven整合ssm工程
   1. 注意：整合思路
2. 拆分
   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
   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