SSM整合教程

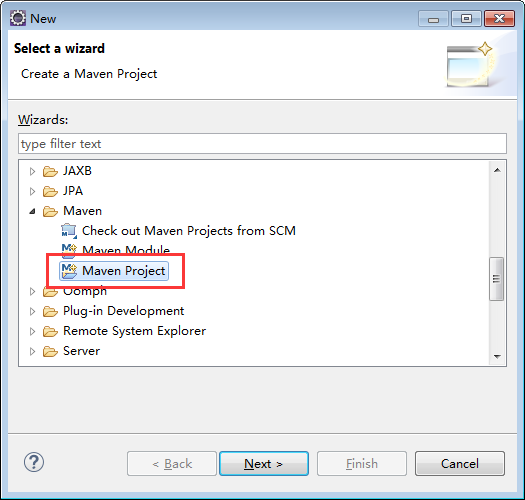
环境备注

IDE：Eclipse Neon (4.6)

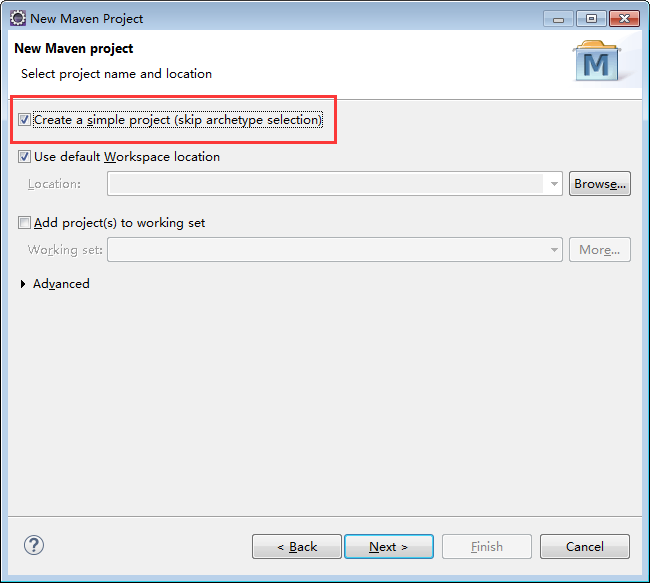
JDK版本：1.8

### 一、创建基本的Maven Web项目

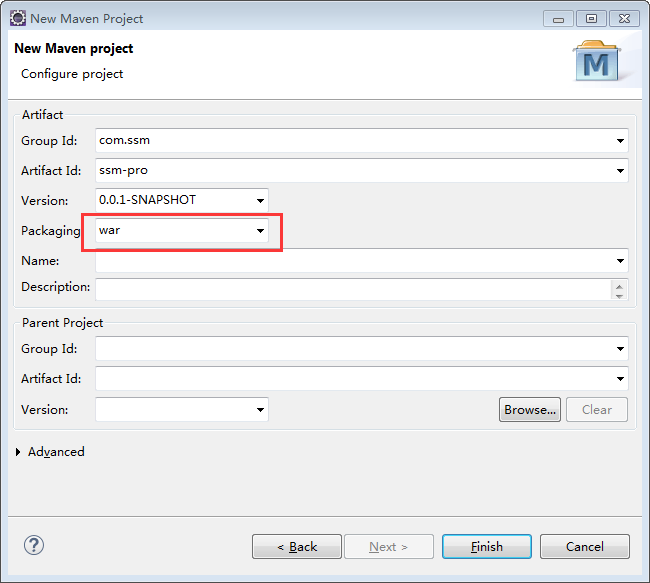
1. 在Eclipse的项目资源管理器Project Explorer右击选择【New】，单击【Other】，打开新建项目向导



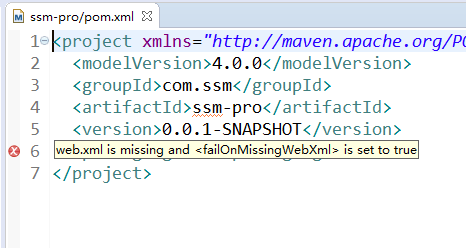
1. 选择【Maven】下的【Maven Project】，点击【Next】
2. 选中Create a simple project(skip archetype selection)复选框，点击【Next】



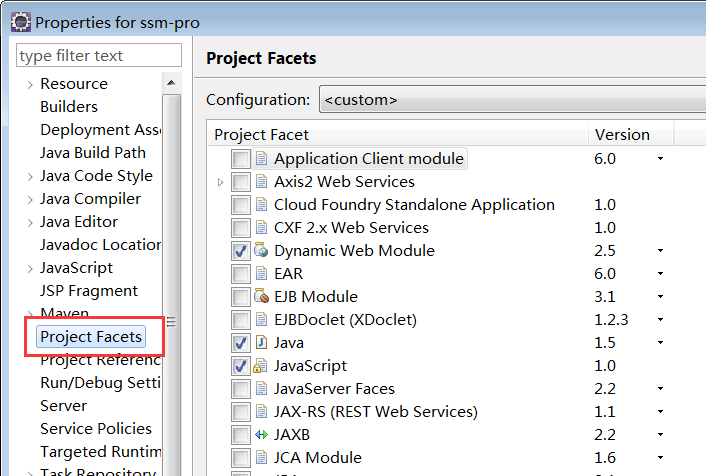
1. 输入Group Id = com.ssm，Artifact Id = ssm-pro，Version = 0.0.1-SNAPSHOT，Packaging = war，点击Finish。注意这里Packaging选择的是war，那么这个Maven项目就是Web项目了。



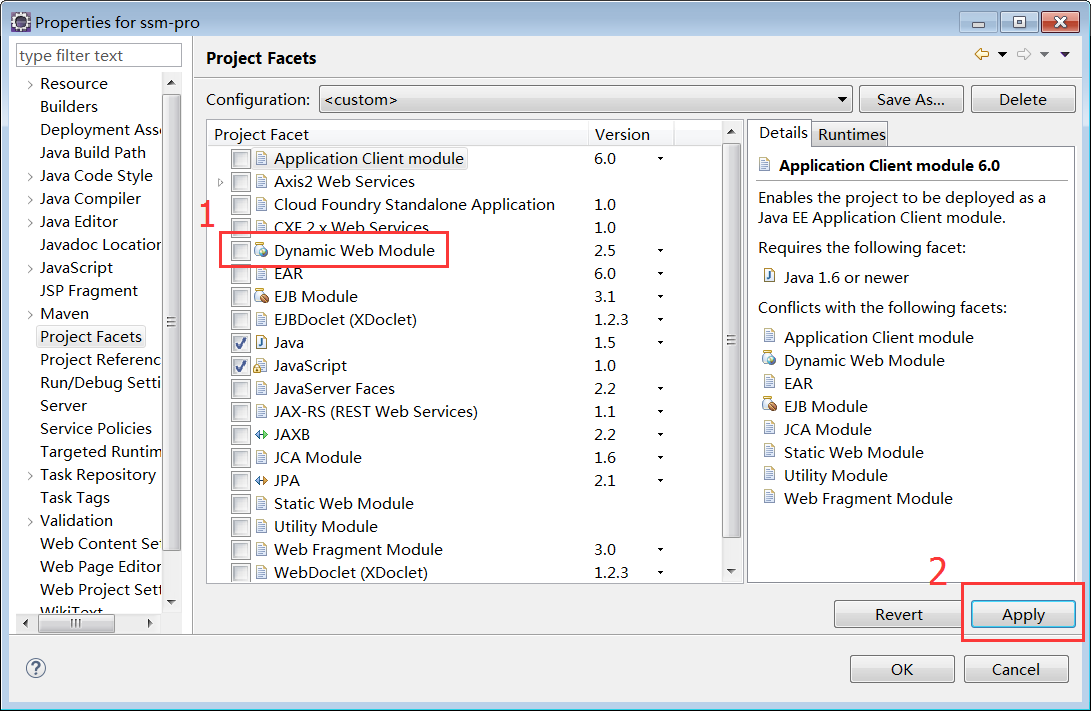
1. 等待项目创建完成后，打开pom文件夹，这里报了一个错误。错误信息是说缺少web.mxl文件，需要手动创建。



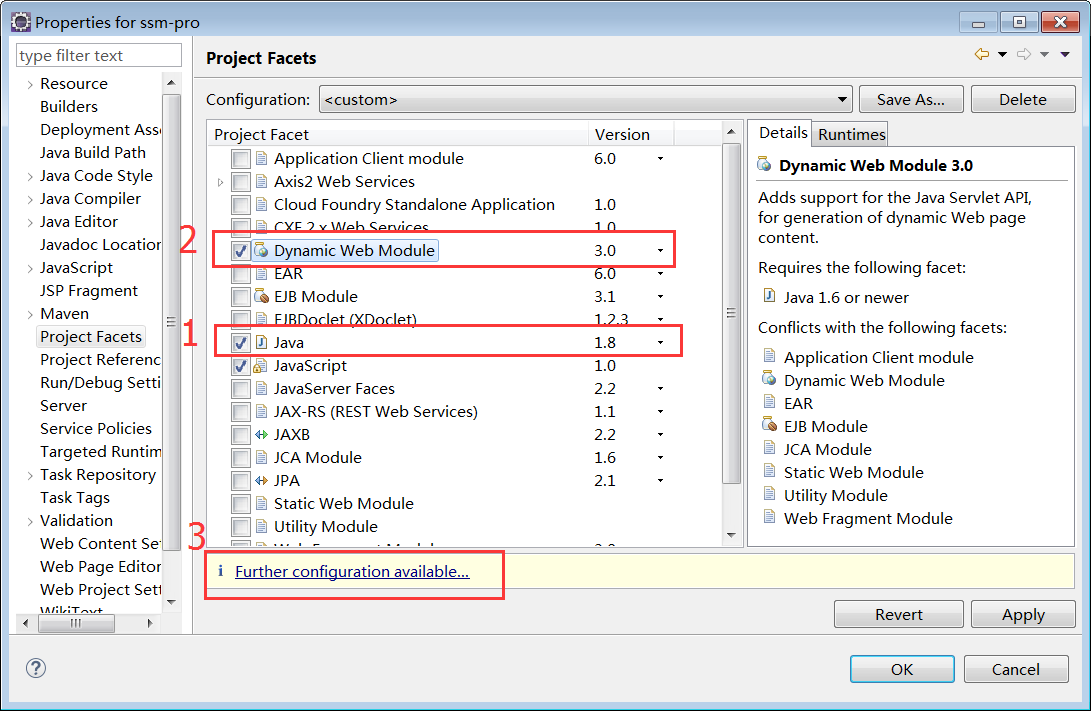
1. 右击项目，选择【Properties】设置，选中Project Facets



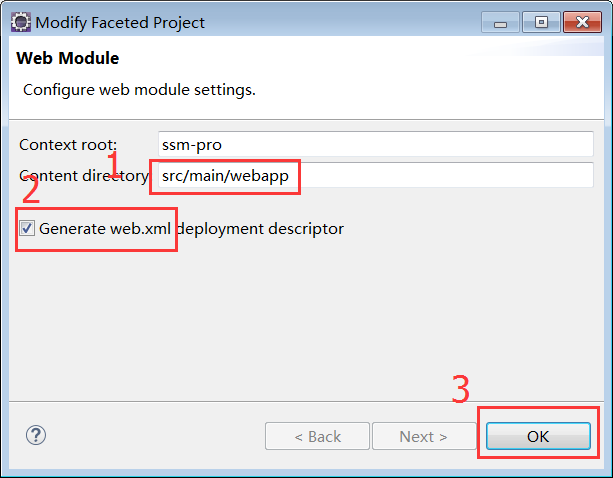
1. 去掉Dynamic Web Module前面的勾，点击【Apply】



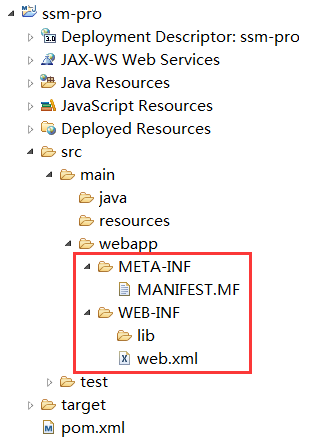
1. Java版本选择1.8，Dynamic Web Module版本选择3.0并勾选上，注意在下方会出现【Further configuration available...】选项，点击该选项



1. 设置Content directory=src/main/webapp，勾选下方Generate web.xml选项，点击【OK】回到Project Facets页面，然后点击【Apply】->【OK】。



1. 现在可看到项目多了如图所示的文件或文件夹，这正是web项目所需要的。



11. pom配置清单

设置源代码编码方式：

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

在dependencies元素中添加基础依赖，包含测试工具、日志处理、jdbc。

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.12</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>1.7.12</version>

</dependency>

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>1.2.17</version>

</dependency>

<!-- jdbc -->

<dependency>

<groupId>mysql</groupId>

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

<version>5.1.38</version>

</dependency>

添加web依赖，jsp-api和servlet-api在web容器中已经提供，为了避免jar包重复引起的错误，需要将jsp-api和servlet-api的scope设置为provided。

<!-- 支持Servlet -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>servlet-api</artifactId>

<version>2.5</version>

<scope>provided</scope>

</dependency>

<!-- 支持JSP -->

<dependency>

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

<artifactId>jsp-api</artifactId>

<version>2.1</version>

<scope>provided</scope>

</dependency>

<!-- 支持JSTL -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

<version>1.2</version>

</dependency>

指定编译源代码的jdk版本

<build>

<plugins>

<plugin>

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

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

12.添加日志文件log4j.properties，文件内容如下：

log4j.rootLogger=DEBUG, stdout

log4j.logger.com.github.pagehelper=DEBUG

log4j.logger.org.apache.ibatis=DEBUG

log4j.logger.com.github.pagehelper.mapper = TRACE

### Console output...

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

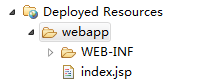
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern=%5p [%t] - %m%n

13. 更新下Maven项目。右击项目，选择【Maven】🡪【Update Project】，在弹出对话框后，直接点击OK。稍等片刻，项目没有报错，已经成功一半了！

14. 测试

在webapp下新建index.jsp文件



index.jsp内容如下：

<%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*

pageEncoding=*"UTF-8"*%>

<%

request.setAttribute("msg", "hello");

%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>

<title>Insert title here</title>

</head>

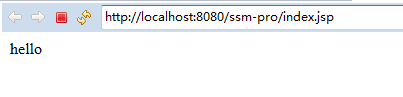
<body>

${msg }

</body>

</html>

将项目发布到Tomcat服务器，并访问<http://localhost:8080/ssm-pro/index.jsp>，结果如下表示项目配置成功：



### 二、整合Spring和Spring MVC

1. 添加Spring项目清单，用于管理Spring依赖。

在pom.xml文件中的dependencies后面添加如下配置：

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-framework-bom</artifactId>

<version>4.3.4.RELEASE</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

2. 添加Spring依赖。在pom.xml的dependencies中添加如下依赖：

<!-- Spring上下文 -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

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

</dependency>

<!-- Spring JDBC -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

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

</dependency>

<!-- Spring 事务 -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-tx</artifactId>

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

</dependency>

<!-- Spring 面向切编程 -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

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

</dependency>

<!-- spring-aop 依赖 -->

<dependency>

<groupId>org.aspectj</groupId>

<artifactId>aspectjweaver</artifactId>

<version>1.8.2</version>

</dependency>

3. 添加Spring MVC依赖。在pom.xml的dependencies中添加如下依赖：

<!-- Spring Web 核心 -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

</dependency>

4. 添加Spring XML配置文件

在src/main/resources中新增applicationContext.xml文件，内容如下：

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

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

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

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

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

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

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans.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*

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

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

<context:component-scan base-package=*"com.ssm.service.impl"*></context:component-scan>

</beans>

5. 添加Spring MVC的配置文件

在src/main/resources中新增spring-servlet.xml文件，内容如下：

<?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: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.xsd*

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

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

<!-- 启用注解驱动 -->

<mvc:annotation-driven/>

<!-- 静态资源映射规则 -->

<mvc:resources location=*"/static/"* mapping=*"/static/\*\*"*></mvc:resources>

<!-- 扫描controller包下的类 -->

<context:component-scan base-package=*"com.ssm.controller"*/>

<!-- 视图解析器 -->

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

<property name=*"viewClass"* value=*"org.springframework.web.servlet.view.JstlView"*></property>

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

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

</bean>

</beans>

1. 配置web.xml

对于Spring来说，需要增加如下配置：

<!-- needed for ContextLoaderListener -->

<context-param>

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

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

</context-param>

<!-- Bootstraps the root web application context before servlet initialization -->

<listener>

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

</listener>

针对Spring MVC，需要增加如下配置：

<!-- The front controller of this Spring Web application, responsible for handling all application requests -->

<servlet>

<servlet-name>springDispatcherServlet</servlet-name>

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

<init-param>

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

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

</init-param>

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

</servlet>

<!-- Map all requests to the DispatcherServlet for handling -->

<servlet-mapping>

<servlet-name>springDispatcherServlet</servlet-name>

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

</servlet-mapping>

配置乱码过滤器：

<filter>

<filter-name>SpringEncodingFilter</filter-name>

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

<init-param>

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

<param-value>UTF-8</param-value>

</init-param>

<init-param>

<param-name>forceEncoding</param-name>

<param-value>true</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>SpringEncodingFilter</filter-name>

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

</filter-mapping>

1. 测试

新建com.ssm.controller包，在该包下新建UserController类，内容如下：

@Controller

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

@RequestMapping("/user")

**public** String user(Model model){

model.addAttribute("user", "Tom");

**return** "user";

}

}

在webapp/WEB-INF目录下新建jsp目录，新建user.jsp文件，内容如下：

<%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*

pageEncoding=*"UTF-8"*%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>

<title>Insert title here</title>

</head>

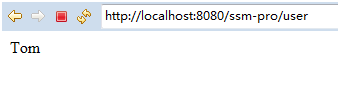
<body>

${user }

</body>

</html>

发布项目到Tomcat，然后访问<http://localhost:8080/ssm-pro/user>，浏览器显示如下内容表示配置成功：



### 三、Spring整合MyBatis

1. 在pom.xml中添加MyBatis依赖

<!-- MyBatis -->

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis</artifactId>

<version>3.3.0</version>

</dependency>

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis-spring</artifactId>

<version>1.3.0</version>

</dependency><!-- MyBatis -->

Mybatis-spring是spring整合mybatis需要的整合包

2. 配置SqlSessionFactoryBean，在applicationContext.xml文件中添加如下配置：

<bean id=*"dataSource"* class=*"org.apache.ibatis.datasource.pooled.PooledDataSource"*>

<property name=*"driver"* value=*"com.mysql.jdbc.Driver"*/>

<property name=*"url"* value=*"jdbc:mysql://localhost:3306/mybatis"*/>

<property name=*"username"* value=*"root"*/>

<property name=*"password"* value=*"root"*/>

</bean>

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

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

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

<property name=*"mapperLocations"*>

<array>

<value>classpath:mapper/\*.xml</value>

</array>

</property>

<property name=*"typeAliasesPackage"* value=*"com.ssm.model"*/>

</bean>

SqlSessionFactoryBean常用属性配置

* configLocation：MyBatis的配置XML路径，上面的配置mybatis-config.xml位于src/main/resources，配置文件内容如下：

<?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>

<setting name=*"logImpl"* value=*"LOG4J"*/>

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

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

<setting name=*"aggressiveLazyLoading"* value=*"false"*/>

</settings>

</configuration>

* dataSource：数据源
* mapperLocation：XML映射文件的路径，可以使用Ant风格的路径进行配置
* typeAliasesPackage：配置包中类的别名，配置后，包中的类在XML映射文件中可以省略包名部分，直接使用类名

3. 配置MapperScannerConfigurer

通过MapperScannerConfigurer类自动扫描所有Mapper接口，使用时可以直接注入接口。

在Spring配置文件applicationContext.xml中配置扫描类，代码如下：

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

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

</bean>

### 四、测试

1. 准备工作

新建一个测试表，使用用户表为例，新建users表并插入测试数据：

create table users(

id int,

name varchar(50),

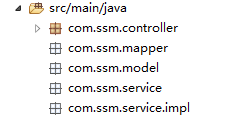
passwd varchar(50)

);

insert into users values(1, 'aaa', '123');

insert into users values(2, 'bbb', '456');

建好所需要的包，com.ssm.controller是之前创建过的：



在com.ssm.model包中创建User实体类，代码如下：

**public** **class** User {

**private** String id;

**private** String name;

**private** String passwd;

//getters、setters、toString方法

}

2. 开发Dao层

在com.ssm.mapper包下新建UserMapper接口，内容如下：

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

User selectById(**int** id);

}

在src/main/resources下新建mapper目录，在该目录下新建UserMapper.mxl文件，内容如下：

<?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=*"com.ssm.mapper.UserMapper"*>

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

select \* from users where id = #{id}

</select>

</mapper>

3. 开发Service层

在com.ssm.service中新建UserService接口，内容如下：

**public** **interface** UserService {

User selectById(**int** id);

}

在com.ssm.service.impl中新建UserServiceImpl，它实现UserService接口，内容如下：

@Service

**public** **class** UserServiceImpl **implements** UserService {

@Autowired

**private** UserMapper userMapper;

@Override

**public** User selectById(**int** id) {

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

}

}

4. 开发Controller层

修改UserController类，内容如下：

@Controller

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

@Autowired

**private** UserServiceImpl userServiceImpl;

@RequestMapping("/user")

**public** String user(Model model){

User user = userServiceImpl.selectById(1);

model.addAttribute("user", user);

**return** "user";

}

}

5. 开发View层

在webapp/WEB-INF/jsp目录中的user.jsp内容跟原先一样，保持不变。

user.jsp文件内容：

<%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*

pageEncoding=*"UTF-8"*%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>

<title>Insert title here</title>

</head>

<body>

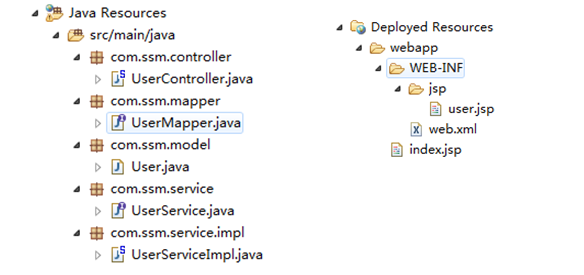
${user }

</body>

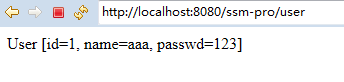
</html>

6. 发布应用

最终包结构和webapp目录结构



部署项目到Tomcat下，重启服务器。服务器启动后，然后在浏览器输入<http://localhost:8080/ssm-pro/user>并访问，结果如下表示此次整合成功：



参考资料：

《MyBatis从入门到精通\_\_刘增辉(著)\_ 电子工业出版社》

源码地址：

在我的工作空间下，名为ssm-pro的项目