文档网址：

<https://www.cnblogs.com/hcbin/p/5397977.html>

[**[JSP]Maven+SSM框架(Spring+SpringMVC+MyBatis) - Hello World**](https://www.cnblogs.com/hcbin/p/5397977.html)

来源：<http://blog.csdn.net/zhshulin/article/details/37956105?utm_source=tuicool&utm_medium=referral>（以下博文对原博文有改动和补充）

博文里红底白字的，为注意修改的地方，在这里先声明

**源码：**[Maven&SSM框架 - Hello World](http://files.cnblogs.com/files/hcbin/Maven%26SSM%E6%A1%86%E6%9E%B6-Hello_World.zip)

**开发环境：**

[Eclipse Java EE IDE for Web Developers. Version: Mars.2 Release (4.5.2)](http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/mars2)

[apache-tomcat-8.0.33](http://tomcat.apache.org/download-80.cgi)

[jdk1.8.0\_77](http://www.oracle.com/technetwork/java/javase/downloads/index-jsp-138363.html)

MySQL 5.0.11-dev（官网下载需要账号登录，故不提供，请自行百度）

**1、基本概念**

**1.1、Spring**

Spring是一个开源框架，Spring是于2003 年兴起的一个轻量级的Java 开发框架，由Rod Johnson 在其著作Expert One-On-One J2EE Development and Design中阐述的部分理念和原型衍生而来。它是为了解决企业应用开发的复杂性而创建的。Spring使用基本的JavaBean来完成以前只可能由EJB完成的事情。然而，Spring的用途不仅限于服务器端的开发。从简单性、可测试性和松耦合的角度而言，任何Java应用都可以从Spring中受益。 简单来说，Spring是一个轻量级的控制反转（IoC）和面向切面（AOP）的容器框架。

**1.2、SpringMVC**

Spring MVC属于SpringFrameWork的后续产品，已经融合在Spring Web Flow里面。Spring MVC 分离了控制器、模型对象、分派器以及处理程序对象的角色，这种分离让它们更容易进行定制。

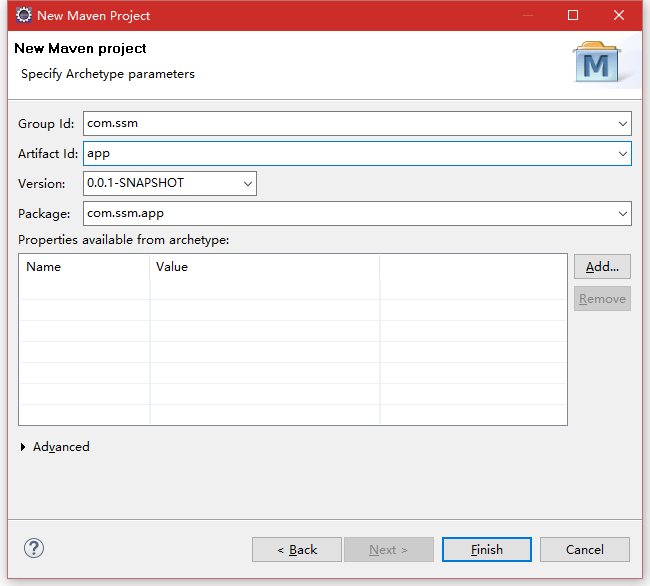
**1.3、MyBatis**

MyBatis 本是apache的一个开源项目iBatis, 2010年这个项目由apache software foundation 迁移到了google code，并且改名为MyBatis 。MyBatis是一个基于Java的持久层框架。iBATIS提供的持久层框架包括SQL Maps和Data Access Objects（DAO）MyBatis 消除了几乎所有的JDBC代码和参数的手工设置以及结果集的检索。MyBatis 使用简单的 XML或注解用于配置和原始映射，将接口和 Java 的POJOs（Plain Old Java Objects，普通的 Java对象）映射成数据库中的记录。

**2、开发环境搭建**

参看博文：[MyEclipse+Tomcat+MAVEN+SVN项目完整环境搭建](http://www.cnblogs.com/MyEclipse+Tomcat+MAVEN+SVN%E9%A1%B9%E7%9B%AE%E5%AE%8C%E6%95%B4%E7%8E%AF%E5%A2%83%E6%90%AD%E5%BB%BA)（作者：shu\_lin）

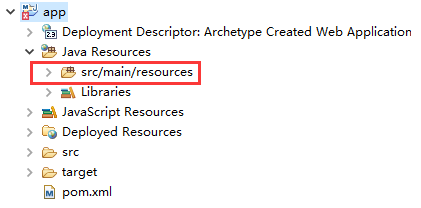
**3、Maven Web项目创建**



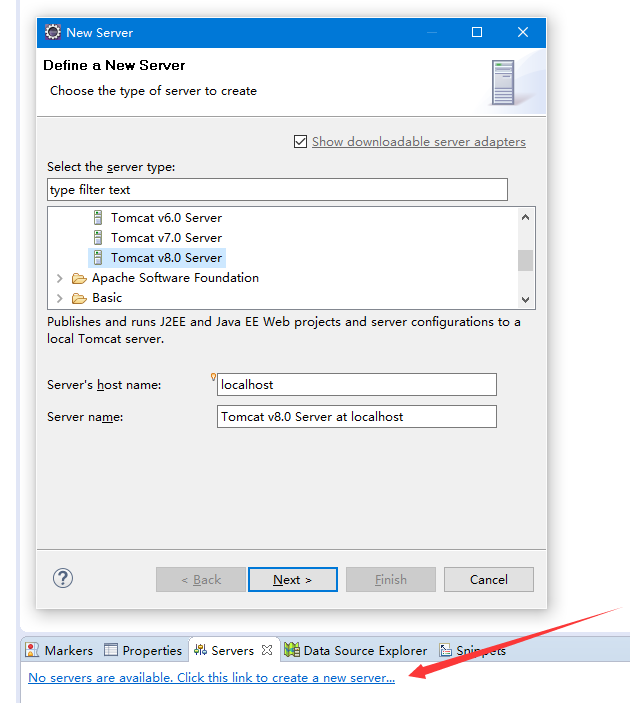
参看博文：[使用maven创建web项目](http://blog.csdn.net/zhshulin/article/details/37921705)（作者：shu\_lin）（博文最后一步可以不用设置，Maven->Update Project...会自动设置）

补充：（可跳过）

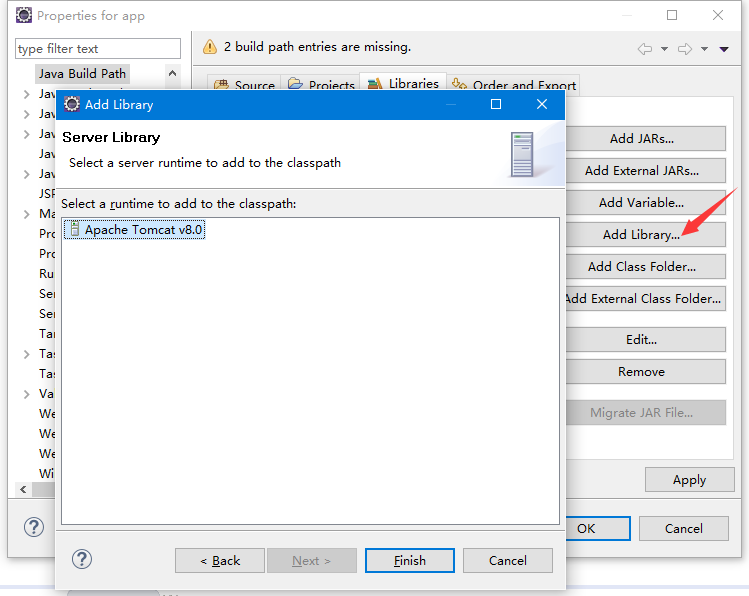
新建的Maven，Java Resources只有resource



可以添加Server Library自动生成



右击项目->Properties->Java Build Path->Add Library...->Server Library



**4、SSM整合**

下面主要介绍三大框架的整合，至于环境的搭建以及项目的创建，参看上面的博文。这次整合我分了2个配置文件，分别是spring-mybatis.xml，包含spring和mybatis的配置文件，还有个是spring-mvc的配置文件，此外有2个资源文件：jdbc.propertis和log4j.properties。

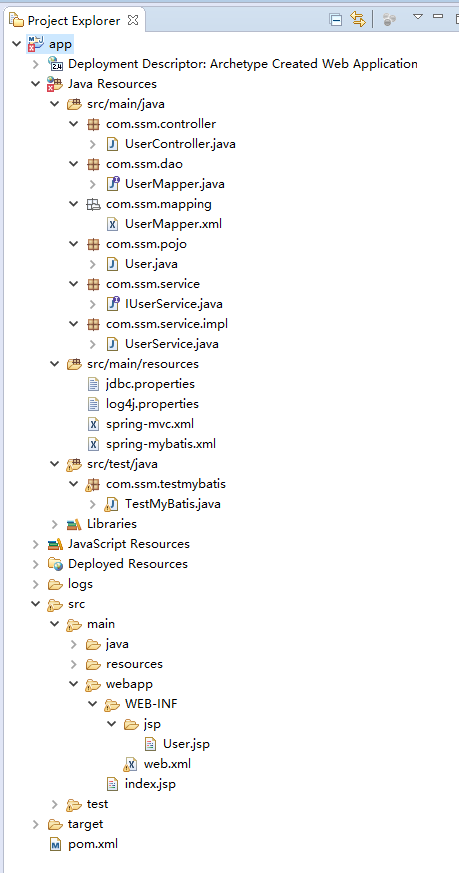
**使用框架都是较新的版本：**

Spring 4.0.2 RELEASE（PS：无需手动下载，Maven会自动下）

Spring MVC 4.0.2 RELEASE（PS：无需手动下载，Maven会自动下）

MyBatis 3.2.6（下载：<https://github.com/mybatis/mybatis-3/releases>）

**完整目录结构如下：**



**4.1、Maven引入需要的JAR包**

为了方便后面说的时候不需要引入JAR包，我这里直接给出所有需要的JAR包，这都是基本的JAR包，每个包的是干什么的都有注释，就不再多说了。

**pom.xml**

[复制代码](javascript:void(0);)

<properties>

<!-- spring版本号 -->

<spring.version>4.0.2.RELEASE</spring.version>

<!-- mybatis版本号 -->

<mybatis.version>3.2.6</mybatis.version>

<!-- log4j日志文件管理包版本 -->

<slf4j.version>1.7.7</slf4j.version>

<log4j.version>1.2.17</log4j.version>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.11</version>

<!-- 表示开发的时候引入，发布的时候不会加载此包 -->

<scope>test</scope>

</dependency>

<!-- spring核心包 -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

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

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

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

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-oxm</artifactId>

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

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-tx</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-webmvc</artifactId>

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

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

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

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context-support</artifactId>

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

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-test</artifactId>

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

</dependency>

<!-- mybatis核心包 -->

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis</artifactId>

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

</dependency>

<!-- mybatis/spring包 -->

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis-spring</artifactId>

<version>1.2.2</version>

</dependency>

<!-- 导入java ee jar 包 -->

<dependency>

<groupId>javax</groupId>

<artifactId>javaee-api</artifactId>

<version>7.0</version>

</dependency>

<!-- 导入Mysql数据库链接jar包 -->

<dependency>

<groupId>mysql</groupId>

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

<version>5.1.30</version>

</dependency>

<!-- 导入dbcp的jar包，用来在applicationContext.xml中配置数据库 -->

<dependency>

<groupId>commons-dbcp</groupId>

<artifactId>commons-dbcp</artifactId>

<version>1.2.2</version>

</dependency>

<!-- JSTL标签类 -->

<dependency>

<groupId>jstl</groupId>

<artifactId>jstl</artifactId>

<version>1.2</version>

</dependency>

<!-- 日志文件管理包 -->

<!-- log start -->

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>${log4j.version}</version>

</dependency>

<!-- 格式化对象，方便输出日志 -->

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>fastjson</artifactId>

<version>1.1.41</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>${slf4j.version}</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>${slf4j.version}</version>

</dependency>

<!-- log end -->

<!-- 映入JSON -->

<dependency>

<groupId>org.codehaus.jackson</groupId>

<artifactId>jackson-mapper-asl</artifactId>

<version>1.9.13</version>

</dependency>

<!-- 上传组件包 -->

<dependency>

<groupId>commons-fileupload</groupId>

<artifactId>commons-fileupload</artifactId>

<version>1.3.1</version>

</dependency>

<dependency>

<groupId>commons-io</groupId>

<artifactId>commons-io</artifactId>

<version>2.4</version>

</dependency>

<dependency>

<groupId>commons-codec</groupId>

<artifactId>commons-codec</artifactId>

<version>1.9</version>

</dependency>

</dependencies>

<build>

<finalName>app</finalName>

<plugins>

<plugin>

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

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

<version>3.5.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

<encoding>UTF-8</encoding>

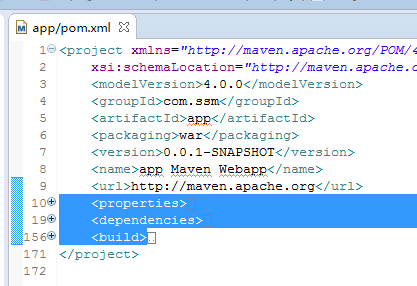
</configuration>

</plugin>

</plugins>

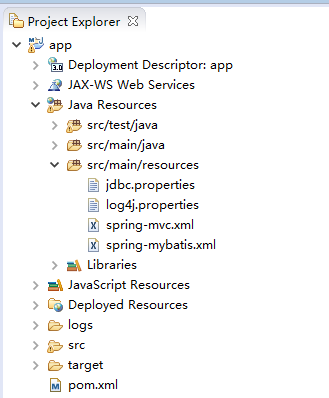
</build>

[复制代码](javascript:void(0);)



**4.2、Spring与MyBatis的整合**

所有需要的JAR包都引入以后，首先进行Spring与MyBatis的整合，然后再进行JUnit测试，先看一个项目结构图：



**4.2.1、建立JDBC属性文件**

**jdbc.properties**（文件编码修改为utf-8）

[复制代码](javascript:void(0);)

driver=com.mysql.jdbc.Driver

url=jdbc:mysql://127.0.0.1:3306/test?useUnicode=true&characterEncoding=UTF-8

username=root

password=123456

#定义初始连接数

initialSize=0

#定义最大连接数

maxActive=20

#定义最大空闲

maxIdle=20

#定义最小空闲

minIdle=1

#定义最长等待时间

maxWait=60000

[复制代码](javascript:void(0);)

**4.2.2、Log4j的配置（跳过此步也没影响）**

为了方便调试，一般都会使用日志来输出信息，Log4j是Apache的一个开放源代码项目，通过使用Log4j，我们可以控制日志信息输送的目的地是控制台、文件、GUI组件，甚至是套接口服务器、NT的事件记录器、UNIX Syslog守护进程等；我们也可以控制每一条日志的输出格式；通过定义每一条日志信息的级别，我们能够更加细致地控制日志的生成过程。  
Log4j的配置很简单，而且也是通用的，下面给出一个基本的配置，换到其他项目中也无需做多大的调整，如果想做调整或者想了解Log4j的各种配置，参看我转载的一篇博文，很详细：[Log4j配置详解](http://blog.csdn.net/zhshulin/article/details/37937365) （作者：shu\_lin）

**log4j.properties**（文件编码修改为utf-8）

[复制代码](javascript:void(0);)

#定义LOG输出级别

log4j.rootLogger=INFO,Console,File

#定义日志输出目的地为控制台

log4j.appender.Console=org.apache.log4j.ConsoleAppender

log4j.appender.Console.Target=System.out

#可以灵活地指定日志输出格式，下面一行是指定具体的格式

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

log4j.appender.Console.layout.ConversionPattern=**[**%c**]** - %m%n

#文件大小到达指定尺寸的时候产生一个新的文件

log4j.appender.File = org.apache.log4j.RollingFileAppender

#指定输出目录

log4j.appender.File.File = logs/springmvcMybist/ssm.log

#定义文件最大大小

log4j.appender.File.MaxFileSize = 10MB

#输出所以日志，如果换成DEBUG表示输出DEBUG以上级别日志

log4j.appender.File.Threshold = ALL

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

log4j.appender.File.layout.ConversionPattern =**[**%p**]** **[**%d{yyyy-MM-dd HH\:mm\:ss}**][**%c**]**%m%n

[复制代码](javascript:void(0);)

**4.2.3、建立spring-mybatis.xml配置文件**

这个文件就是用来完成spring和mybatis的整合的。这里面也没多少行配置，主要的就是自动扫描，自动注入，配置数据库。注释也很详细，大家看看就明白了。

**spring-mybatis.xml**

[复制代码](javascript:void(0);)

<?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-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/mvc

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

<!-- 自动扫描 -->

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

<!-- 引入配置文件 -->

<bean id="propertyConfigurer"

class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">

<property name="location" value="classpath:jdbc.properties" />

</bean>

<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"

destroy-method="close">

<property name="driverClassName" value="${driver}" />

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

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

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

<!-- 初始化连接大小 -->

<property name="initialSize" value="${initialSize}"></property>

<!-- 连接池最大数量 -->

<property name="maxActive" value="${maxActive}"></property>

<!-- 连接池最大空闲 -->

<property name="maxIdle" value="${maxIdle}"></property>

<!-- 连接池最小空闲 -->

<property name="minIdle" value="${minIdle}"></property>

<!-- 获取连接最大等待时间 -->

<property name="maxWait" value="${maxWait}"></property>

</bean>

<!-- spring和MyBatis完美整合，不需要mybatis的配置映射文件 -->

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

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

<!-- 自动扫描mapping.xml文件 -->

<property name="mapperLocations" value="classpath:com/ssm/mapping/\*.xml"></property>

</bean>

<!-- DAO接口所在包名，Spring会自动查找其下的类 -->

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

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

<property name="sqlSessionFactoryBeanName" value="sqlSessionFactory"></property>

</bean>

<!-- (事务管理)transaction manager, use JtaTransactionManager for global tx -->

<bean id="transactionManager"

class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

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

</bean>

</beans>

[复制代码](javascript:void(0);)

**4.2.4、JUnit测试**

经过以上步骤（log4j不配也没影响），我们已经完成了Spring和mybatis的整合，这样我们就可以编写一段测试代码来试试是否成功了。

**4.2.4.1、创建测试用表**

既然我们需要测试，那么我们就需要建立在数据库中建立一个测试表，这个表建的很简单，SQL语句为：

**MySQL**

[复制代码](javascript:void(0);)

#

# Structure for table "user\_t"

#

DROP TABLE IF EXISTS `user\_t`;

CREATE TABLE `user\_t` (

`id` int(**11**) NOT NULL AUTO\_INCREMENT,

`user\_name` varchar(**40**) NOT NULL,

`password` varchar(**255**) NOT NULL,

`age` int(**4**) NOT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=**2** DEFAULT CHARSET=utf8;

#

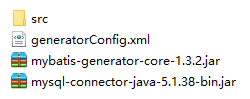
# Data for table "user\_t"

#

INSERT INTO `user\_t` VALUES (**1**,'测试','123456',**24**);

[复制代码](javascript:void(0);)

**4.2.4.2、利用MyBatis Generator自动创建代码**



随便找个位置建文件夹（路径最好都是英文），然后在里面建个src文件夹，下载下面那两个包，将generatorConfig.xml拷进去，最后打开cmd，进到文件夹，执行下面的命令，代码就会生成到src文件夹里

[mysql-connector-java.jar](http://dev.mysql.com/downloads/connector/j/)

[mybatis-generator-core.jar](http://grepcode.com/project/repo1.maven.org/maven2/org.mybatis.generator/mybatis-generator-core/)

**generatorConfig.xml**

[复制代码](javascript:void(0);)

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

<!DOCTYPE generatorConfiguration

PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"

"http://mybatis.org/dtd/mybatis-generator-config\_1\_0.dtd">

<generatorConfiguration>

<!-- 数据库驱动 -->

<classPathEntry location="mysql-connector-java-5.1.38-bin.jar" />

<context id="DB2Tables" targetRuntime="MyBatis3">

<commentGenerator>

<property name="suppressDate" value="true" />

<!-- 是否去除自动生成的注释 true：是 ： false:否 -->

<property name="suppressAllComments" value="true" />

</commentGenerator>

<!--数据库链接URL，用户名、密码 -->

<jdbcConnection driverClass="com.mysql.jdbc.Driver"

connectionURL="jdbc:mysql://127.0.0.1:3306/test" userId="root"

password="123456">

</jdbcConnection>

<javaTypeResolver>

<property name="forceBigDecimals" value="false" />

</javaTypeResolver>

<!-- 生成模型的包名和位置 -->

<javaModelGenerator targetPackage="com.ssm.pojo"

targetProject="src">

<property name="enableSubPackages" value="true" />

<property name="trimStrings" value="true" />

</javaModelGenerator>

<!-- 生成映射文件的包名和位置 -->

<sqlMapGenerator targetPackage="com.ssm.mapping"

targetProject="src">

<property name="enableSubPackages" value="true" />

</sqlMapGenerator>

<!-- 生成DAO的包名和位置 -->

<javaClientGenerator type="XMLMAPPER"

targetPackage="com.ssm.dao" targetProject="src">

<property name="enableSubPackages" value="true" />

</javaClientGenerator>

<!-- 要生成的表 tableName是数据库中的表名或视图名 domainObjectName是实体类名 -->

<table tableName="user\_t" domainObjectName="User"

enableCountByExample="false" enableUpdateByExample="false"

enableDeleteByExample="false" enableSelectByExample="false"

selectByExampleQueryId="false"></table>

</context>

</generatorConfiguration>

[复制代码](javascript:void(0);)

**cmd命令**

java -jar mybatis-generator-core-1.3.2.jar -configfile generatorConfig.xml -overwrite

可做成.bat文件备用，放在目录运行即可

**cmd命令**

set current\_path="%cd%"

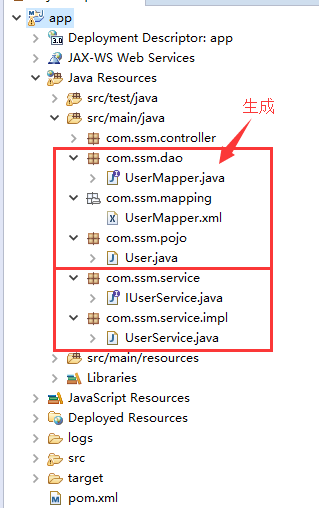
java -jar mybatis-generator-core-1.3.2.jar -configfile generatorConfig.xml -overwrite

pause

exit

参考博文：[SSM框架——使用MyBatis Generator自动创建代码](http://blog.csdn.net/zhshulin/article/details/23912615)（作者：shu\_lin）

这个可根据表自动创建实体类、MyBatis映射文件以及DAO接口，完成后将文件复制到工程中。如图：



IUserService.java

[复制代码](javascript:void(0);)

package com.ssm.service;

import com.ssm.pojo.User;

public interface IUserService {

public User getUserById(int userId);

}

[复制代码](javascript:void(0);)

UserService.java

[复制代码](javascript:void(0);)

package com.ssm.service.impl;

import javax.annotation.Resource;

import org.springframework.stereotype.Service;

import com.ssm.dao.UserMapper;

import com.ssm.pojo.User;

import com.ssm.service.IUserService;

@Service("userService")

public class UserService implements IUserService {

@Resource

private UserMapper userDao;

public User getUserById(int userId) {

return this.userDao.selectByPrimaryKey(userId);

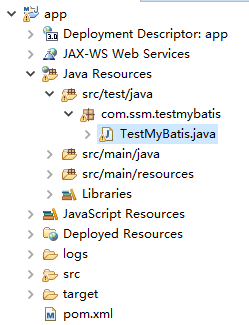
}

}

[复制代码](javascript:void(0);)

**4.2.4.4、建立测试类**

测试类在src/test/java中建立，下面测试类中注释掉的部分是不使用Spring时，一般情况下的一种测试方法；如果使用了Spring那么就可以使用注解的方式来引入配置文件和类，然后再将service接口对象注入，就可以进行测试了。如果测试成功，表示Spring和Mybatis已经整合成功了。输出信息使用的是Log4j打印到控制台。



**TestMyBatis.java**

[复制代码](javascript:void(0);)

package com.ssm.testmybatis;

import javax.annotation.Resource;

import org.apache.log4j.Logger;

import org.junit.Before;

import org.junit.Test;

import org.junit.runner.RunWith;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import org.springframework.test.context.ContextConfiguration;

import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import com.alibaba.fastjson.JSON;

import com.ssm.pojo.User;

import com.ssm.service.IUserService;

@RunWith(SpringJUnit4ClassRunner.class) // 表示继承了SpringJUnit4ClassRunner类

@ContextConfiguration(locations = { "classpath:spring-mybatis.xml" })

public class TestMyBatis {

private static Logger logger = Logger.getLogger(TestMyBatis.class);

// private ApplicationContext ac = null;

@Resource

private IUserService userService = null;

// @Before

// public void before() {

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

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

// }

@Test

public void test() {

User user = userService.getUserById(1);

// System.out.println(user.getUserName());

// logger.info("值："+user.getUserName());

logger.info(JSON.toJSONString(user));

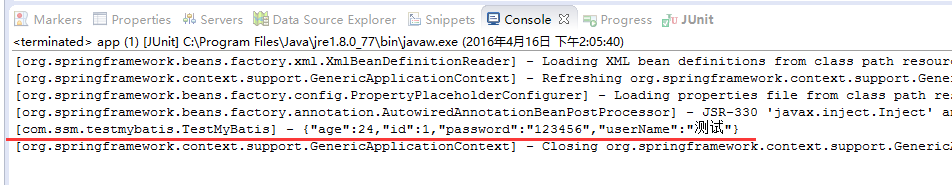
}

}

[复制代码](javascript:void(0);)

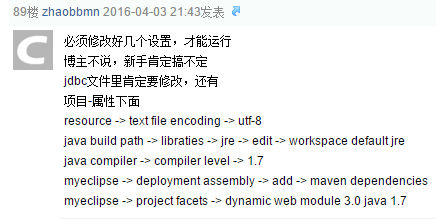
**测试：**

先clean项目（可跳过），再Maven->Update Project...（重点），就是为了更新项目设置，（右击项目->Run As->JUnit Test）



至此，完成Spring和mybatis这两大框架的整合，下面在继续进行SpringMVC的整合。

如测试失败请参考网友方法，一般clean项目和右击项目->Maven->Update Project...更新一下项目设置都可以解决：



**4.3、整合SpringMVC**

上面已经完成了2大框架的整合，SpringMVC的配置文件单独放，然后在web.xml中配置整合。

**4.3.1、配置spring-mvc.xml**

配置里面的注释也很详细，在此就不说了，主要是自动扫描控制器，视图模式，注解的启动这三个。

**spring-mvc.xml**

[复制代码](javascript:void(0);)

<?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-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/mvc

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

<!--避免IE执行AJAX时，返回JSON出现下载文件 -->

<bean id="mappingJacksonHttpMessageConverter"

class="org.springframework.http.converter.json.MappingJacksonHttpMessageConverter">

<property name="supportedMediaTypes">

<list>

<value>text/html;charset=UTF-8</value>

</list>

</property>

</bean>

<!-- 启动SpringMVC的注解功能，完成请求和注解POJO的映射 -->

<bean

class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter">

<property name="messageConverters">

<list>

<ref bean="mappingJacksonHttpMessageConverter" /> <!-- JSON转换器 -->

</list>

</property>

</bean>

<!-- 对静态资源的访问 -->

<!-- <mvc:resources mapping="/js/\*\*" location="/js/" /> <mvc:resources mapping="/css/\*\*"

location="/css/" /> -->

<!-- 自动扫描该包，使SpringMVC认为包下用了@controller注解的类是控制器 -->

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

<!-- 添加注解驱动 -->

<mvc:annotation-driven enable-matrix-variables="true" />

<!-- 允许对静态资源文件的访问 -->

<mvc:default-servlet-handler />

<!-- 定义跳转的文件的前后缀 ，视图模式配置 -->

<bean

class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<!-- 这里的配置我的理解是自动给后面action的方法return的字符串加上前缀和后缀，变成一个 可用的url地址 -->

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

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

</bean>

<!-- 配置文件上传，如果没有使用文件上传可以不用配置，当然如果不配，那么配置文件中也不必引入上传组件包 -->

<bean id="multipartResolver"

class="org.springframework.web.multipart.commons.CommonsMultipartResolver">

<!-- 默认编码 -->

<property name="defaultEncoding" value="UTF-8" />

<!-- 文件大小最大值 -->

<property name="maxUploadSize" value="10485760000" />

<!-- 内存中的最大值 -->

<property name="maxInMemorySize" value="40960" />

</bean>

</beans>

[复制代码](javascript:void(0);)

**4.3.2、配置web.xml文件**

这里面对spring-mybatis.xml的引入以及配置的spring-mvc的Servlet就是为了完成SSM整合，之前2框架整合不需要在此处进行任何配置。配置一样有详细注释，不多解释了。

**web.xml**

[复制代码](javascript:void(0);)

<?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\_3\_0.xsd"

version="3.0">

<display-name>Archetype Created Web Application</display-name>

<!-- Spring和mybatis的配置文件 -->

<context-param>

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

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

</context-param>

<!-- 编码过滤器 -->

<filter>

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

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

<async-supported>true</async-supported>

<init-param>

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

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

</init-param>

</filter>

<filter-mapping>

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

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

</filter-mapping>

<!-- Spring监听器 -->

<listener>

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

</listener>

<!-- 防止Spring内存溢出监听器 -->

<listener>

<listener-class>org.springframework.web.util.IntrospectorCleanupListener</listener-class>

</listener>

<!-- Spring MVC servlet -->

<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-mvc.xml</param-value>

</init-param>

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

<async-supported>true</async-supported>

</servlet>

<servlet-mapping>

<servlet-name>SpringMVC</servlet-name>

<!-- 此处可以可以配置成\*.do，对应struts的后缀习惯 -->

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

</servlet-mapping>

<welcome-file-list>

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

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

</welcome-file-list>

</web-app>

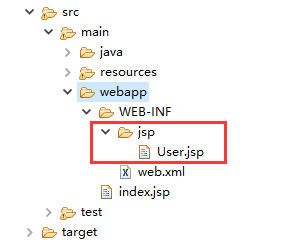
[复制代码](javascript:void(0);)

**4.3.3、测试**

至此已经完成了SSM三大框架的整合了，接下来测试一下，如果成功了，那么恭喜你，如果失败了，继续调试吧，作为程序员就是不停的与BUG做斗争！

**4.3.3.1、新建jsp页面**

User.jsp   此页面仅输出一下用户名，完成一个完整的简单流程。



**User.jsp**

[复制代码](javascript:void(0);)

<%@ 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>测试</title>

</head>

<body>${user.userName}

</body>

</html>

[复制代码](javascript:void(0);)

**4.3.3.2、建立UserController类**

**UserController.java**

[复制代码](javascript:void(0);)

package com.ssm.controller;

import javax.annotation.Resource;

import javax.servlet.http.HttpServletRequest;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.RequestMapping;

import com.ssm.pojo.User;

import com.ssm.service.IUserService;

@Controller

@RequestMapping("/user")

public class UserController {

@Resource

private IUserService userService;

@RequestMapping("/showUser")

public String toIndex(HttpServletRequest request, Model model) {

int userId = Integer.parseInt(request.getParameter("id"));

User user = this.userService.getUserById(userId);

model.addAttribute("user", user);

return "User";

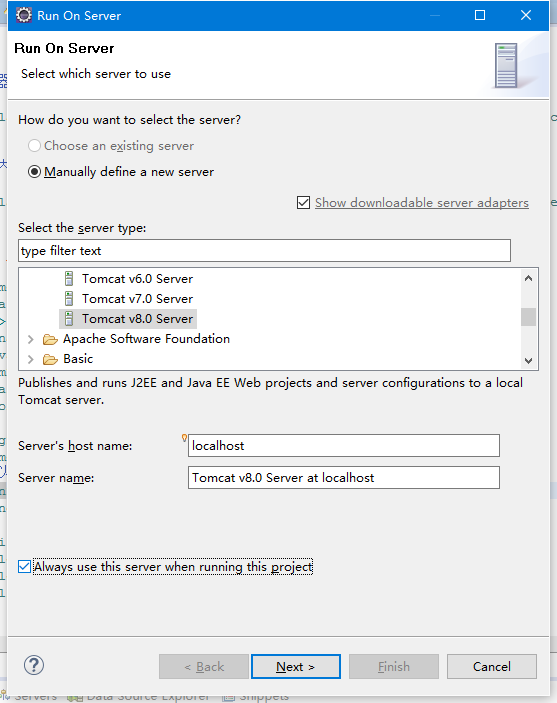
}

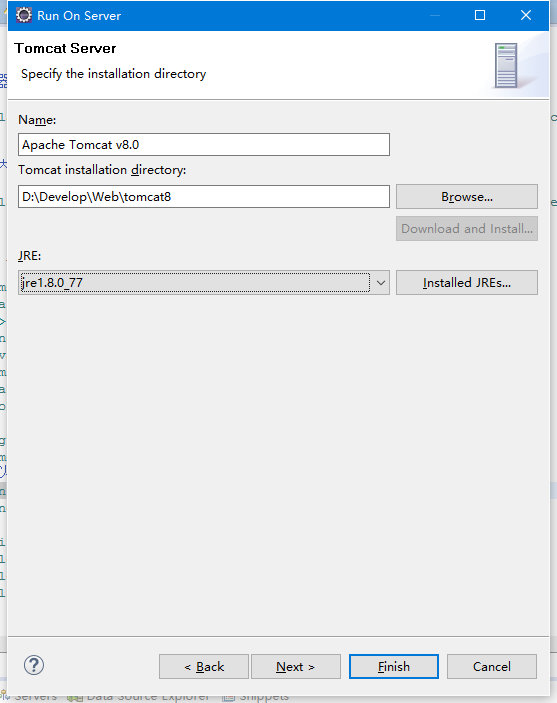
}

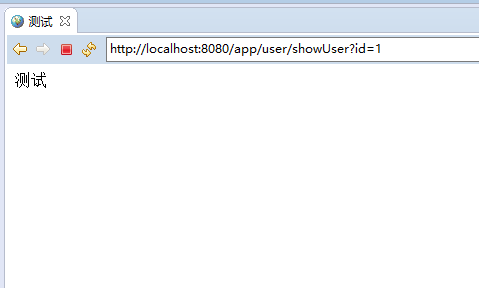
[复制代码](javascript:void(0);)

**4.3.3.3、部署项目**

再次测试（右击项目->Run As->Run On Serser），输入地址：<http://localhost:8080/app/user/showUser?id=1>







如果成功请不要开心得太早，Maven->Update Project...或将项目拷到其他机上后会立马出现404！shu\_lin与其他博主都没有为此作出任何后续说明，为此写出此博文作出补充。

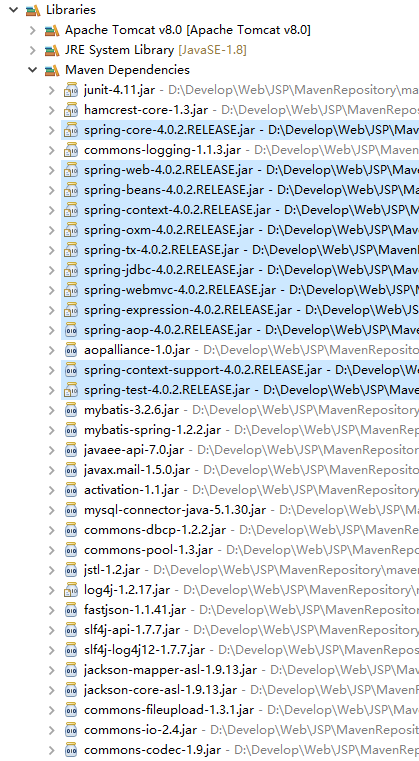
**补充：**

无论是shu\_lin的[SSM框架——详细整合教程（Spring+SpringMVC+MyBatis）](http://blog.csdn.net/zhshulin/article/details/37956105?utm_source=tuicool&utm_medium=referral)还是其他人的博文，都没有写明Maven->Update Project...后会导致项目不能运行，会出现关于Spring包的ClassNotFoundException错误这个问题，

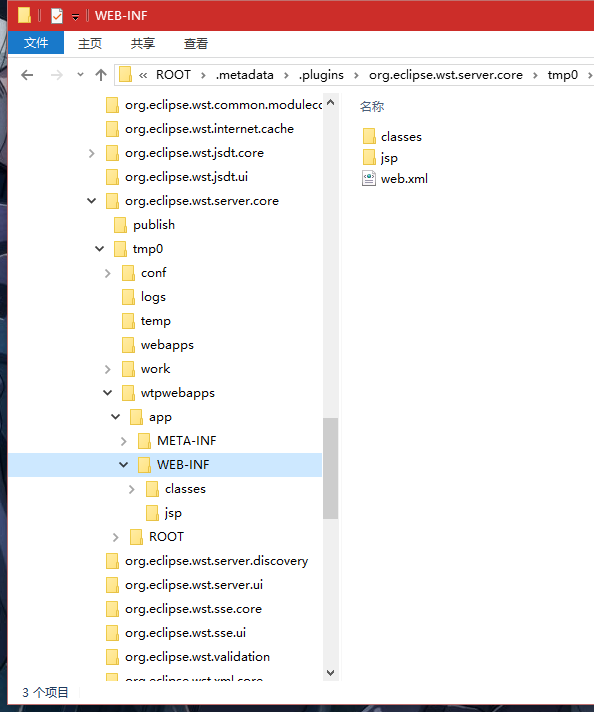
（对于我这个新手（Java零基础新手，Eclipse也不熟悉）头大了。。。花了近一个月来学这个Hello World。就是卡在这个问题上。）

**疑问开始，找Maven**

带着这个疑问？就第一时间找到Maven，可是奇怪了，都包含了？！为什么还是找不到类？



从百度里找了多个答案后，终于找到问题所在，就是编译时没有复制依赖包到WEB-INF/lib里，编译时也没有生成lib文件夹



**临时方法：**（PS：感觉方法不太对，放弃了）

百度里有个方法就是教你修改.classpath，手动添加/WEB-INF/lib

<classpathentry kind="con" path="org.eclipse.m2e.MAVEN2\_CLASSPATH\_CONTAINER">

<attributes>

<attribute name="maven.pomderived" value="true"/>

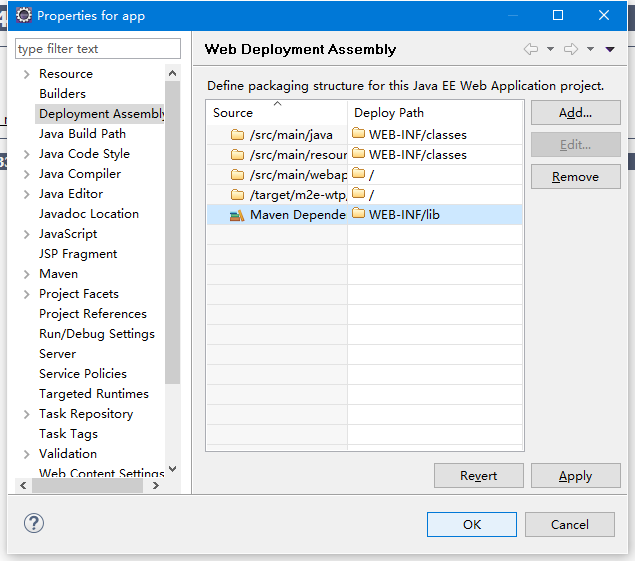
<attribute name="org.eclipse.jst.component.dependency" value="/WEB-INF/lib"/>

</attributes>

</classpathentry>

**临时方法：**

右击项目->Properties->Deployment Assembly->Add->Java Build Path Entries->Maven Dependencies->Finish完成添加，再点击OK关闭



再次测试（右击项目->Run As->Run On Serser），输入地址：<http://localhost:8080/app/user/showUser?id=1>，运行OK！

**完美解决方法：**

可是毕竟是临时方法，Maven->Update Project...后还是会导致项目不能运行，因为配置被Maven还原了！！！不行，这还是没有解决，还需要再找，几番周折终于找到了！

打开cmd，定位到项目目录下，执行下面命令

**cmd命令**（根据Java Build Path的列表得知，只要不改动包，就不需要再次运行此命令）

mvn eclipse:eclipse -Dwtpversion=2.0

可做成.bat文件备用，放在项目根目录运行即可

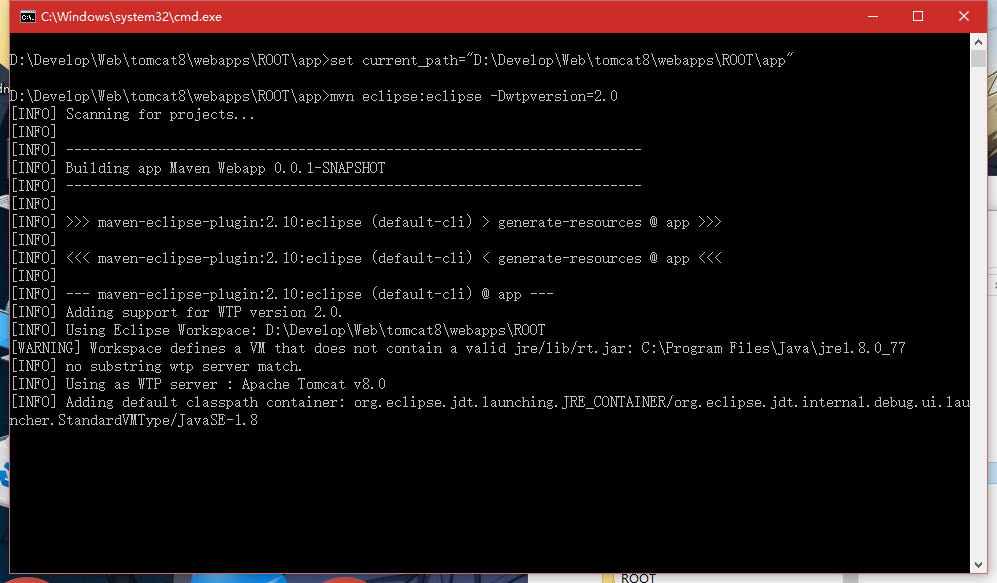
**cmd命令**

set current\_path="%cd%"

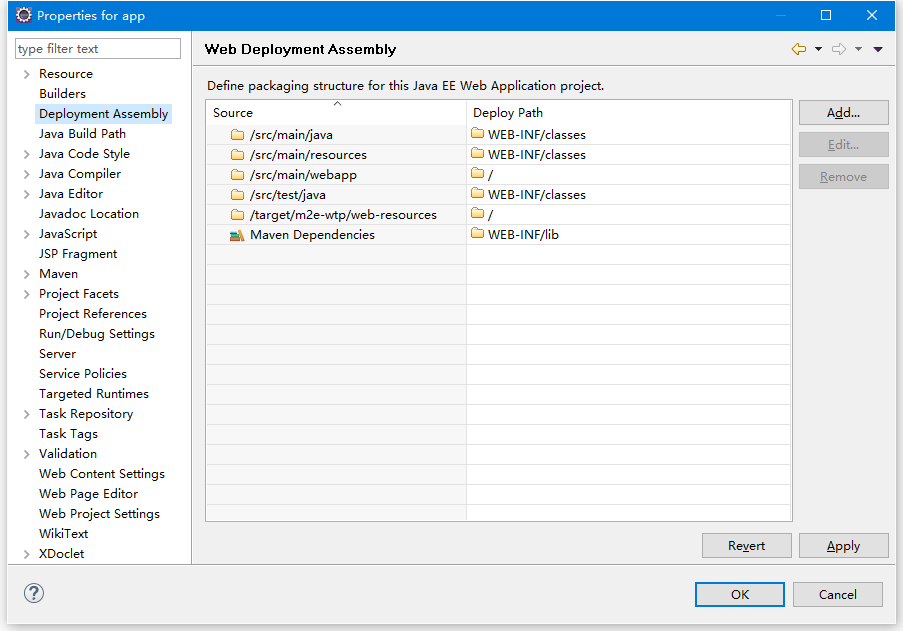
mvn eclipse:eclipse -Dwtpversion=2.0

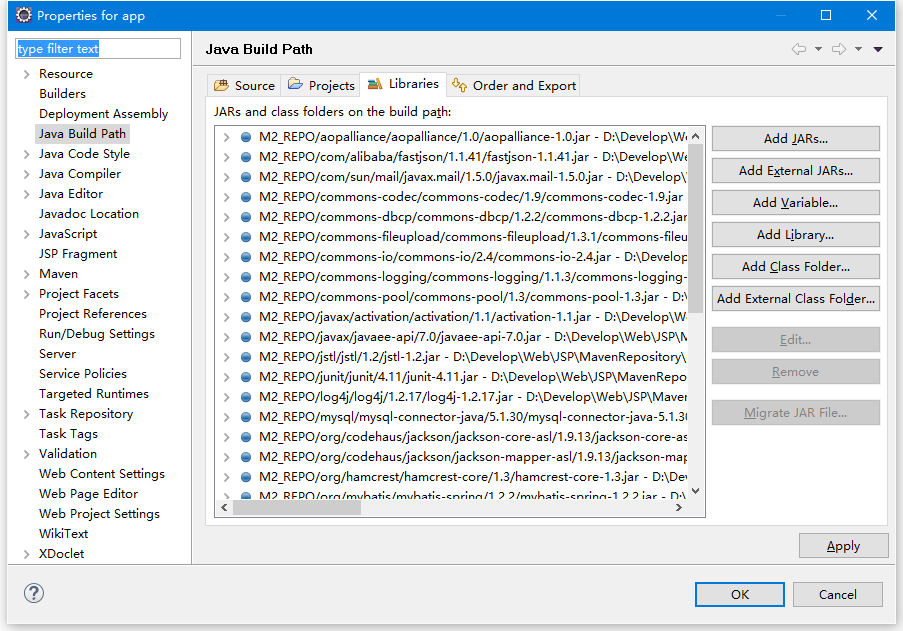
pause

exit

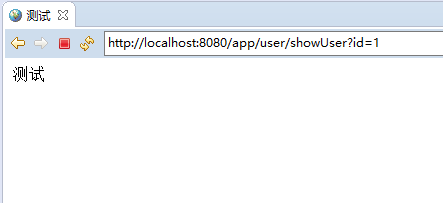


执行命令以后，clean项目和Maven->Update Project...后，Deployment Assembly和Java Build Path都有变化

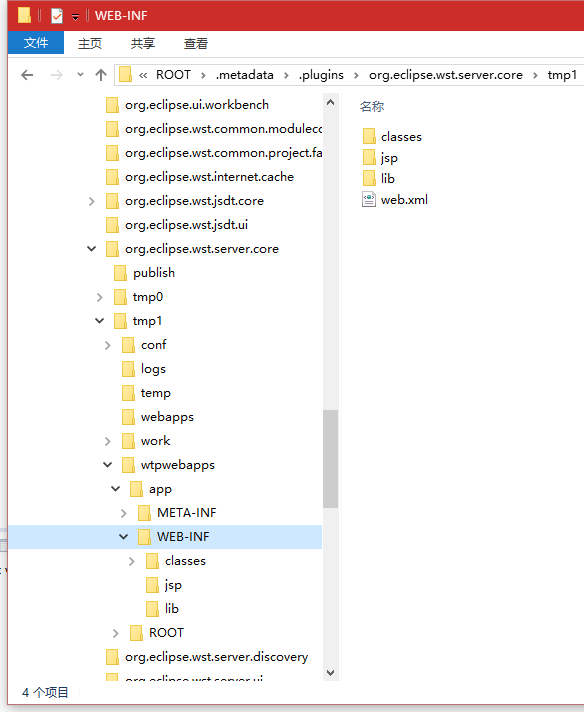




再次测试（右击项目->Run As->Run On Serser），输入地址：<http://localhost:8080/app/user/showUser?id=1>，运行OK！完美！



再看看WEB-INF里，终于自动生成lib文件夹



个性签名：做要做好，做到不三不四不如不做。