# 建立单工程步骤

## 第一种 Maven工程创建

### 第一步：maven环境配置

系统maven环境配置(可省略，cmd若要使用maven需配置)

下载apache的maven：http://maven.apache.org/download.cgi 最新版3.6需要JDk1.7环境

配置环境变量使用bin中命令：tomcat运行工程需要用到

MAVEN\_HOME：D:\apache-maven-3.6\apache-maven

Path：% MAVEN\_HOME%/bin;

### 第二步：配置settings.xml

settings.xml：maven目录下的\apache-maven\conf\settings.xml

1指定本地仓库

<!-- localRepository

| The path to the local repository maven will use to store artifacts.

| Default: ${user.home}/.m2/repository

<localRepository>/path/to/local/repo</localRepository>

-->

<localRepository>D:\apache-maven-3.6.0\resource</localRepository> //指定本地仓库位置

2配置中央仓库：最好把默认的一起配置上：<mirrors>标签下

注意：默认的中央仓库会一些必要的组件下载不成功，但是阿里云仓库有人说tomcat的maven插件会下载不了

<mirrors>

阿里云中央仓库

<mirror>

<id>nexus-aliyun</id>

<mirrorOf>\*</mirrorOf>

<name>Nexus aliyun</name>

<url>http://maven.aliyun.com/nexus/content/groups/public</url>

</mirror>

默认中央仓库 ：两个都配上把，不然会出现：安装工程不到本地仓库的问题

<mirror>

<id>mirrorId</id>

<mirrorOf>repositoryId</mirrorOf>

<name>Human Readable Name for this Mirror.</name>

<url>http://my.repository.com/repo/path</url>

</mirror>

</mirrors>

### 第三步 eclipse环境配置

打开eclipse进行配maven环境

1、Eclipse有默认的进行修改自己的

Window --- preferences --- maven --- installations中进行添加

2、Eclipse 指定settings.xml配置文件

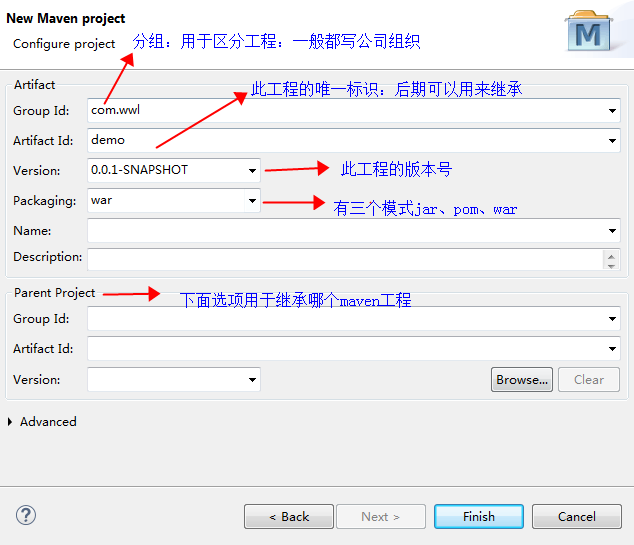
Window --- preferences --- maven ---User Settings

注意：单配置：User Settings(open file) 即可(也可以都配置),然后进行点击一下Update Settings 保存

### 第四步 建立maven工程

右键new ---other ---Maven ---Maven Project





创建好后因为war包工程会报错：src ---main---webapp下没有web.xml

快速创建web.xml:右键工程----选择java EE tools ---Generate………

### 第五步pom.xml的配置

pom.xml的配置：使用JDK1.7、内置tomcat和导包操作

<!-- jar包操作 -->

<dependencies>

<dependency> junit的jar包

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.9</version>

<scope>test</scope> //正常包不用设置此标签：设置scope为测试包

</dependency>

<dependency> servlet的api 2.5版本不是3.0

<groupId>javax.servlet</groupId>

<artifactId>servlet-api</artifactId>

<version>2.5</version>

<scope>provided</scope> //正常包不用设置此标签：设置scope为编译时需要用到，打包为工程不要

</dependency>

<dependency> jsp解析运行相关的api

<groupId>javax.servlet</groupId>

<artifactId>jsp-api</artifactId>

<version>2.0</version>

<scope>provided</scope> //正常包不用设置此标签：设置scope为编译时需要用到，打包为工程不要

</dependency>

</dependencies>

<!—tomcat和jdk操作 -->

<build>

<plugins>

<plugin>

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

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

<version>3.5.1</version>

<configuration>

<source>1.7</source>

<target>1.7</target>

<!-- 指明编译源代码时使用的字符编码，

maven编译的时候默认使用的GBK编码，

通过encoding属性设置字符编码，

告诉maven这个项目使用UTF-8来编译 -->

<encoding>utf8</encoding>

</configuration>

</plugin>

<plugin>

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

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

<configuration>

<!-- 指定端口 -->

<port>8080</port>

<!-- 请求路径 -->

<path>/</path>

</configuration>

</plugin>

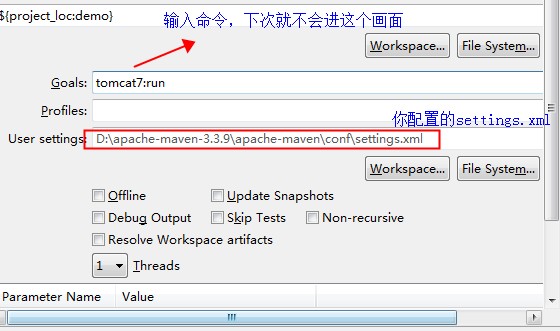
</plugins>

</build>

### 第六步 运行工程和获取工程war包

1、运行工程

右键工程名----选择Run As ---maven-bulid



2内置的tomcat获取war包

安装工程：右键工程名----选择Run As ---maven-install

此时你的本地仓库会根据你的组id加工程名为文件夹进行保存：如com.wwl,进行打包保存war路径：

路径：D:\apache-maven-3.3.9\maven\_resource\com\wwl\demo\0.0.1-SNAPSHOT\



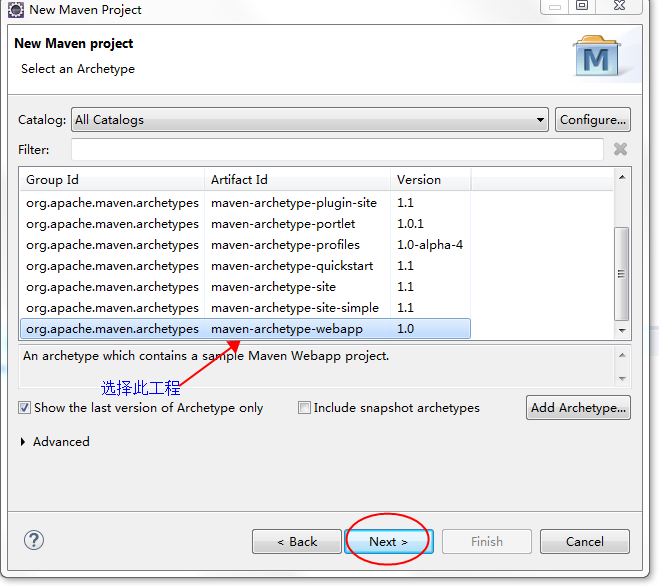
### 自定义tomcat

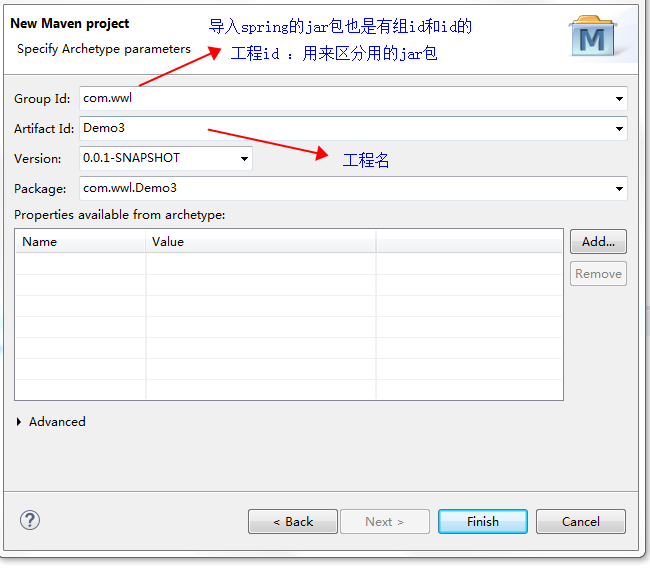
第五步中pom中不添加tomcat插件，然后用第二种方式的第二步配置Tomcat即可

## 第二种 Maven工程创建

### 第一步 建立Maven工程

1、右键new ---other ---Maven ---Maven Project





### 第二步：配置tomcat

注意：配置完第一步此时会报错的：缺少tomcat(就和普通工程添加tomcat一样)

右键工程：Properties 🡪java Build Path 🡪Add Library 🡪Server Runtime 🡪选择你配置好的tomcat即可

### 第三步：优化

Maven完整的工程是有4个包文件夹以下：

src/main/java

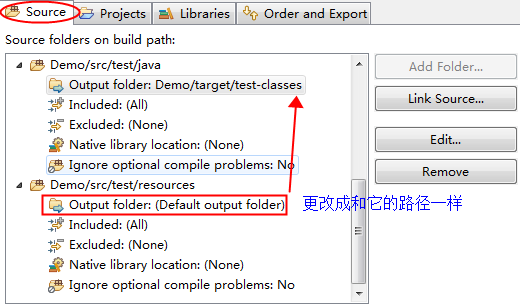
src/ main /resources

src/test/ java

src/test/resources

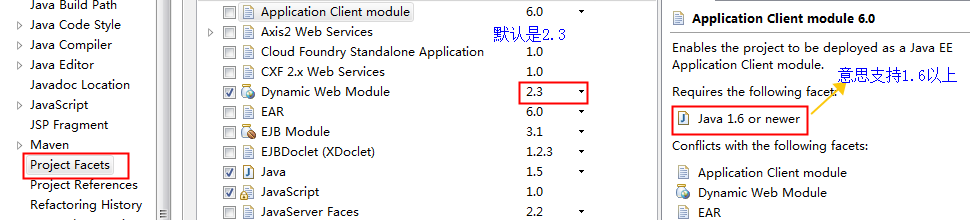
1、缺少test包文件：src/test/resources

配置此包路径：右键工程：Properties 🡪java Build Path🡪Source



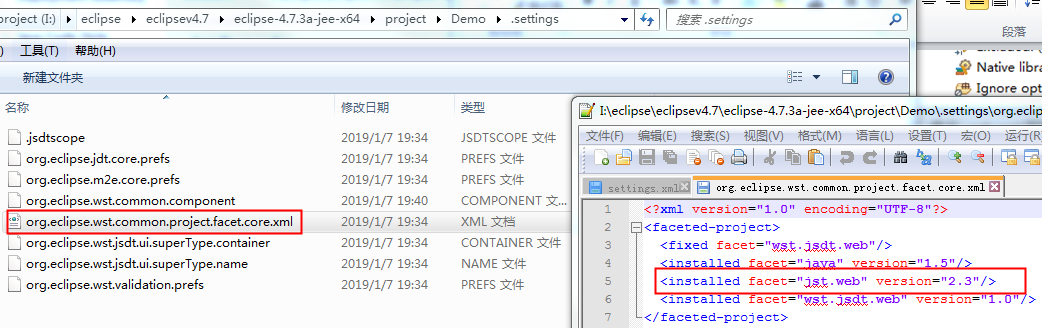
2、优化web.xml容器版本

右键工程：Properties 🡪Project Facets



注意：要更改先更改JDk版本，JDK版本更改也要在pom配置，修改完后就需要把web.xml更改

小知识：可以在工程文件夹下直接配置：.settings文件夹下有个可以打开的文本xml



第四步 部署操作就和普通工程一样

# 建立分布式工程步骤

前几步：配置maven的环境，eclipse的配置看建立[单一工程步骤](#_建立单一工程步骤)

第一步 首先创建一个pom工程bill-parent作为所有的父类只是用来管理jar版本号

<modelVersion>4.0.0</modelVersion>

<groupId>com.wwl</groupId>

<artifactId>bill-parent</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>pom</packaging>

<!-- 集中定义依赖版本号：要换版本号直接这里更换即可-->

<properties> //第一步建立管理版本号标签：这标签就和properties文件一样：key和value

<!-- mybatis和spring整合包版本号 -->

<mybatis.spring.version>1.3.2</mybatis.spring.version> //标签名字可以自定义：下面会进行引用

</properties>

<dependencyManagement> 第二步 建立此标签：可以认为不要加载标签内的资源jar

<dependencies>

<!-- mybatis和spring整合包 -->

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis-spring</artifactId>

<version>${mybatis.spring.version}</version> 引用properties标签的资源：和tl标签引用一样

</dependency>

</dependencies>

</dependencyManagement>

<!—第三步 java编译插件 :直接父类配置好-->

<plugin>

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

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

<version>3.5.1</version>

<configuration>

<!-- 源代码使用的开发版本 -->

<source>1.7</source>

<!-- 需要生成的目标class文件的编译版本 -->

<target>1.7</target>

<encoding>UTF-8</encoding>

</configuration>

</plugin>

<pluginManagement> 第四步 tomcat插件维护：可以认为不要加载标签内的插件

<plugins>

<plugin>

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

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

<version>2.2</version>

</plugin>

</plugins>

</pluginManagement>

第二步 建立表现层工程bill-web，选择war包继承第一步的工程

Pom.xml的配置

<parent> <!-- 继承父类的标签 -->

<groupId>com.wwl</groupId> <!-- 父类组id -->

<artifactId>bill-parent</artifactId> <!-- 父类的id也就是工程名 -->

<version>0.0.1-SNAPSHOT</version> <!-- 父类的版本号 -->

</parent>

<artifactId>bill-web</artifactId> <!--自己工程的id也就是工程名 -->

<packaging>war</packaging> <!-- war会出现标签-->

<dependencies>

引用父类的jar包：就不用版本号了

<dependencies>

<!-- mybatis和spring整合包 -->

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis-spring</artifactId>

</dependency>

</dependencies>

省略tomcat插件内容，跟单一工程配置一摸一样

第三步 依次创建pom工程用于管理：interface工程、service工程、dao工程

其它工程差不多形式

注意事项：

1、创建的工程由于都是独立的：要进行关联就跟导jar形式一样

如：因service工程需要调dao工程的代码：也就是依赖于dao工程jar操作如下：

第一：先对dao工程进行安装：选中dao工程右键 🡪 As run🡪 Maven install此时本地仓库就会此jar包

可以根据组id在本地仓库中找到相应的jar包

第二：service层的pom中引入dao工程：

<dependencies>

<dependency>

<groupId>com.wwl</groupId>

<artifactId>bill-dao</artifactId>

<version>0.0.1-SNAPSHOT</version>

</dependency>

</dependencies>

1. 资源文件如：application-config.xml、jdbc.properties,放在本工程src/main/resources文件夹下即可

3、最后注意war包工程会统一把资源收集起来

1)资源文件war工程Maven install命令打包的时候会自动打包到它的war工程中

2)相应的依赖的工程都会以jar文件在WEB-INF/lib/下;安装到本地仓库的war工程确实都有(把war包放到自己的tomcat下运行时没有问题的)，但是在本地进行测试工程是不行的会报错找不到相关的类，建议还是统一把资源文件放在war工程中

# 三种工程

1.pom工程：用在父级工程或聚合工程中。用来做jar包的版本控制。

2.war工程：将会打包成war，发布在服务器上的工程。如网站或服务。

3.jar工程：将会打包成jar用作jar包使用。

pom工程

wwl-parent：

taotao-parent是整个项目的父工程，它是一个pom工程。用来做整个项目的版本控制，也就是项目中所有要使用到的jar包的版本都集中由父工程管理。这样你在写其他工程pom文件中maven依赖时就不需要写版本号了。当然所有的项目都要先继承它才行。

wwl-manager：聚合工程

创建taotao-manager用来做聚合工程，它也是一个pom工程。创建四个model分别是wwl-manager-pojo,wwl-manager-dao,wwl-manager-service,wwl-manager-web，同时会自动生成4个独立的maven工程。聚合工程只是用来帮助其他模块构建的工具，本身并没有实质的内容。具体每个工程代码的编写还是在生成的工程中去写。

使用聚合工程taotao-manager的意义就是：原本这些模块也是一个个独立的工程，现在将它们聚合到wwl-manager中，这样我们构建项目的时候就只要构建wwl-manager一个就行了。我们只要使用maven构建这个聚合工程wwl-manager就行了而不用去操心模块的构建，比如install时只要install taotao-manager就行。总之就是简化操作。编码工作还是在对应的wwl-manager-pojo, wwl-manager-dao,wwl-manager-service,wwl-manager-web。工程中进行的。

war工程

wwl-manager-web工程都是要部署在服务器上的，所以要打包成war形式。这些工程有的是用户通过浏览器直接访问，有的是通过发布服务被别的工程调用。

jar工程

如wwl-common这个就是打包成jar的工程。它就是存放一些其他工程都会使用的类，工具类。我们可以在其他工程的pom文件中去引用它，和引用别的jar包没什么区别。

<dependency>

<groupId>com.wwl</groupId> 工程名字

<artifactId>taotao-common</artifactId> 要引用的id

<version>0.0.1-SNAPSHOT</version>

</dependency>

# pom标签

下面导包过程分析：资源都在本地仓库

<groupId>、<artifactId>、<version>的值都是文件夹名称

<dependencies>

<dependency>

<groupId>org.mybatis</groupId> //本地仓库的两个文件夹名称org和mybatis：具体到artifactId的文件夹路径

<artifactId>mybatis-spring</artifactId> // mybatis-spring文件夹下维护者各版本的文件夹

<version>1.3.1</version> //1.3.1版本名称的文件夹

</dependency>

</dependencies>

意思就是：本地仓库/org/mybatis/mybatis-spring/1.3.1/的文件夹下找此jar资源

# 常用的配置

JDK版本

JDK7配置一般配置在父工程pom中

<build>

<plugins>

<plugin>

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

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

<version>3.5.1</version>

<configuration>

<source>1.7</source>

<target>1.7</target>

<!-- 指明编译源代码时使用的字符编码，

maven编译的时候默认使用的GBK编码，

通过encoding属性设置字符编码，

告诉maven这个项目使用UTF-8来编译 -->

<encoding>utf8</encoding>

</configuration>

</plugin>

</plugins>

</build>

tomcat配置

tomcat7 一般先配置在父工程成，使用pluginManagement不进行依赖，那个war工程移过去用

<pluginManagement>

<plugins>

<plugin>

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

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

<configuration>

<!-- 指定端口 -->

<port>8080</port>

<!-- 请求路径 -->

<path>/</path>

</configuration>

</plugin>

</plugins>

</pluginManagement>

WEB层要用到的jsp插件：jstl、servlet-api、jsp-api

<!-- JSP相关 -->

<dependency> <!—Jstl 标签 -->

<groupId>jstl</groupId>

<artifactId>jstl</artifactId>

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

</dependency>

<dependency> <!-- servlet(tomcat中自带，编译中使用) -->

<groupId>javax.servlet</groupId>

<artifactId>servlet-api</artifactId>

<version>${servlet-api.version}</version>

<scope>provided</scope>

</dependency>

<dependency> <!-- JSP (tomcat中自带，编译中使用) -->

<groupId>javax.servlet</groupId>

<artifactId>jsp-api</artifactId>

<version>${jsp-api.version}</version>

<scope>provided</scope>

</dependency>

Spring框架

只要导spring-context包，5个核心包和1个log包都会自动关联导出

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>4.3.20.RELEASE</version>

</dependency>

配置了此包，maven会把：context、aop、beans、core、commons-logging、expression都会自动导出