# 建立工程步骤

第一步：maven环境配置：最终要运行的是maven软件中bin目录的mvn命令

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

第三步：打开eclipse进行配maven

Eclipse有默认的进行修改自己的

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

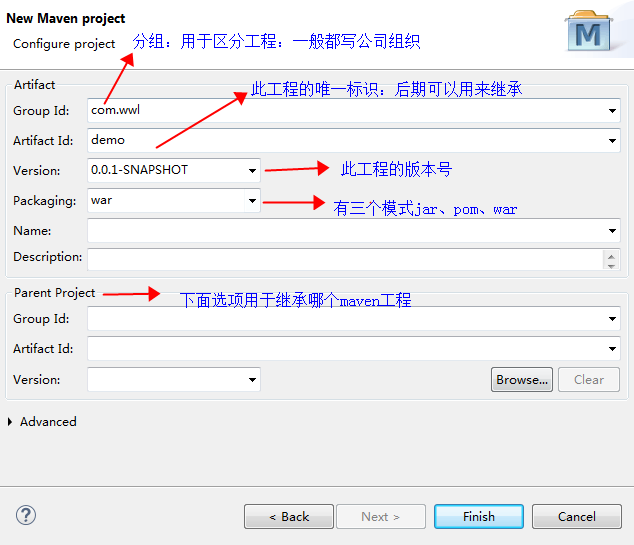
Eclipse 指定settings.xml配置文件

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

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：使用JDK1.7、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>

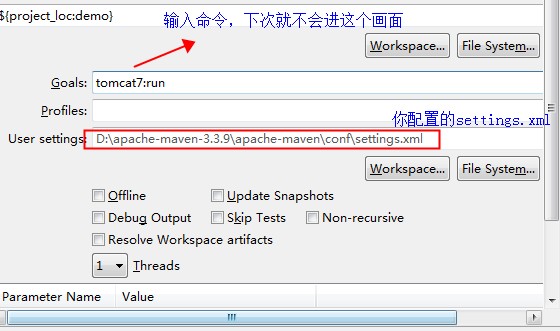
</plugins>

</build>

第六步 运行工程和获取工程war包

1在maven下运行工程

右键工程名----选择Run As ---maven-bulid



2获取war包

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\



# 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>

# 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>