DMS项目自动化发布文档（1/2）Maven项目操作手册

**新保软件（上海） 金海天**

**2012/10/29**

**目录**

[Maven简介 2](#_Toc341451391)

[Maven安装配置 2](#_Toc341451392)

[Maven的Eclipse插件安装配置 3](#_Toc341451393)

[Maven Integration for Eclipse (m2e) 3](#_Toc341451394)

[Maven Integration for WTP 5](#_Toc341451395)

[Maven项目及其特征 6](#_Toc341451396)

[Maven项目在Eclipse中的操作 7](#_Toc341451397)

[创建Maven项目 7](#_Toc341451398)

[可视化pom.xml配置 9](#_Toc341451399)

[Deployment Assembly及Server 11](#_Toc341451400)

[Maven指令使用，Run As… 18](#_Toc341451401)

[普通Web项目转化为Maven Web项目 21](#_Toc341451402)

[示例，DMS项目结构 21](#_Toc341451403)

[转换，Maven化的DMS项目结构 22](#_Toc341451404)

[替换jar包 25](#_Toc341451405)

[配置plugin 27](#_Toc341451406)

# Maven简介

Apache Maven是目前非常流行的一个项目构建工具，它能够按照配置自动地完成项目生命周期中的各种操作，并且提供强大的依赖管理功能。

Maven的依赖管理功能可以非常有效地解决项目组中jar包的寻找及同步问题，并为多模块及彼此依赖的项目开发提供强有力的支持。Maven的众多插件能够满足开发者对于项目生命周期的某个阶段的各种特殊要求，开发者还能自己编写插件来完成新的需求。Maven的流行也让许多其他的工具平台发展出集成Maven的接口，这也让Maven的功能更加多元化及强大，更加地被开发者广泛接受。

有关Maven这款工具的详细信息，请至其官网<http://maven.apache.org/>查阅。

# Maven安装配置

首先在官网上下载最新版本的Maven，如apache-maven-3.0.4-bin.zip。解压该压缩至安装位置，如C:\Program Files\Apache Software Foundation，则Maven的安装路径即C:\Program Files\Apache Software Foundation\apache-maven-3.0.4。然后在系统环境变量中添加用户变量M2\_HOME，变量值即为Maven的安装路径。在Path变量中加入%M2\_HOME%\bin，并确保Path中已经正确地配置了Java环境变量。打开命令行，执行mvn –version查看Maven是否已经安装成功。

Maven的配置文件有两处，

全局配置：$M2\_HOME\conf\settings.xml；

用户配置：${user.home}\.m2\settings.xml；

如果两处配置文件都存在，则他们共同起效，并且当两处配置有冲突时，用户配置优先于全局配置。用户可以把全局配置文件复制到用户配置文件的路径，然后再做修改。

Maven的配置文件主要可以管理以下配置，

**HTTP代理；**

**服务器用户密码；**

**Maven库的位置；**

**多profile配置切换；**

详细的配置方法，访问<http://maven.apache.org/settings.html>查阅。

# Maven的Eclipse插件安装配置

Eclipse是目前Java项目开发人员普遍使用的IDE，支持许多插件功能，Maven作为非常流行的构建工具自然也不例外。

## Maven Integration for Eclipse (m2e)

Maven Integration for Eclipse (m2e)为Eclipse提供以下强大的Maven支持功能，

**pom.xml文件的可视化编辑配置（包括依赖树状图及解析完的jar包列表）；**

**依赖管理与Eclipse中classpath的集成；**

**自动下载或更新依赖构件(jar包)并自动尝试下载其Javadoc与源码；**

**创建或导入Maven项目，并支持从一个pom.xml文件实体化项目；**

**浏览或搜索远程Maven库；**

**让Eclipse识别嵌套的多模块Maven项目；**

在Eclipse Marketplace中安装完该插件后，打开Window->Preferences->Maven中配置下载及更新策略，如下图

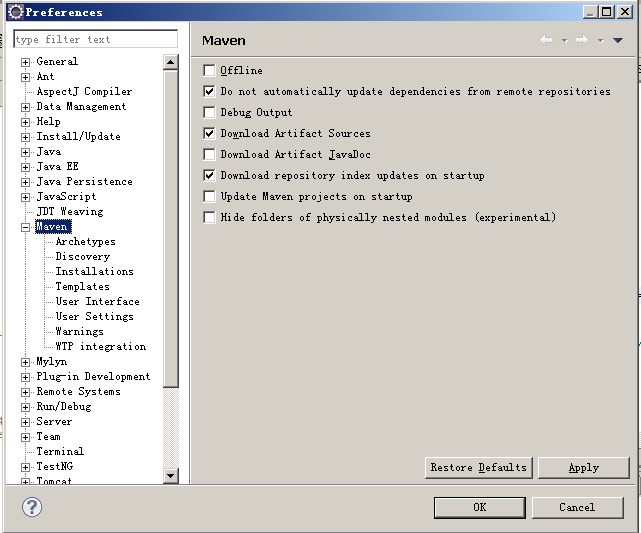


图1

在Window->Preferences->Maven->User Settings中添加Maven的配置文件的为路径，插件会自动按照该路径的文件中的配置更新本地Maven库的位置以及其他设置，一般在此处添加的是用户配置文件，如下图，

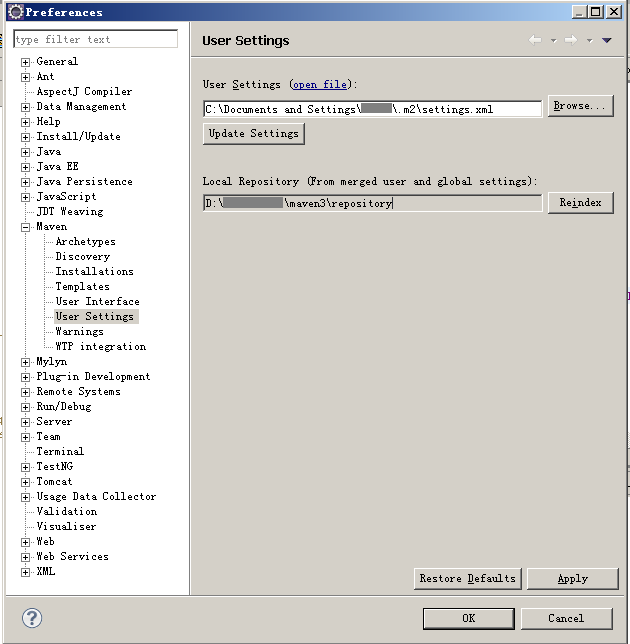


图2

## Maven Integration for WTP

Maven Integration for WTP为m2e与Eclipse Web Tools Platform提供强有力的整合，能够让Web项目进行动态部署的同时兼顾所有m2e插件提供的特性。

在Eclipse Marketplace中安装完该插件后，发现Window->Preferences->Maven->WTP Integration可供配置（如图1、图2所示，新版本插件可能会显示为Java EE Integration），说明插件安装成功。

# Maven项目及其特征

一个项目只需要在项目根目录处包含一个pom.xml文件，该项目即可称之为一个Maven项目。POM是Project Object Model的缩写，将整个项目抽象为以XML语言为表现形式的文件，包含了一个项目所需的所有必要信息，如项目辨识（发布组织，项目名称，项目版本，分辨标志，发布格式），与其他项目的关系，项目依赖的配置（jar包），构建时各个插件的设置，SCM信息，项目寻找依赖构件及插件的库位置，以及项目发布的位置等其他信息。有关pom.xml文件详细的配置方法，请前往<http://maven.apache.org/pom.html>查阅。

按照规范，一个标准的Maven项目应符合如下结构，

|  |  |
| --- | --- |
| src\main\java | 项目源代码文件夹 |
| src\main\resources | 项目资源文件夹 |
| src\main\webapp | 项目Web文件夹 |
| src\test\java | 项目测试用例文件夹 |
| src\test\resource | 项目测试资源文件夹 |
| target | 项目构建输出文件夹 |
| pom.xml | 项目对象模型 |

表1

一个符合上述规范结构的Maven项目能直接使用Maven的各种指令，如编译、测试、打包、安装、发布，因为Maven默认会在特定的构建运行阶段去这些文件夹中寻找相应的文件并进行操作。而不符合该规范结构的Maven项目，需要借助其他的插件配置以在特定的构建运行阶段做调整。

# Maven项目在Eclipse中的操作

当在Eclipse中安装完Maven的相关插件后，我们就能享受到Maven项目在Eclipse IDE中开发时插件所带来的强大功能和便利。

## 创建Maven项目

m2e插件可以方便的为我们创建一个符合Maven标准结构规范的Maven项目。在Eclipse中，选择File->New->Other…，在对话框中选择Maven Project点击Next，如下图

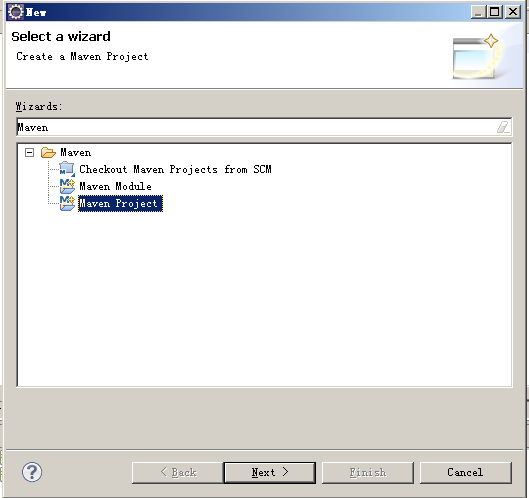


图3

在Select an Archetype步骤时，我们选择一个webapp项目的结构模版，继续，如下图

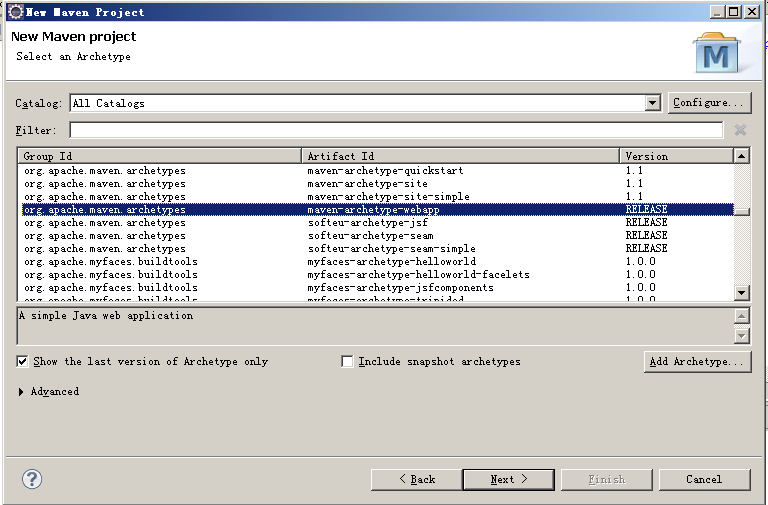


图4

最后输入项目辨识信息，完成项目创建，如下图

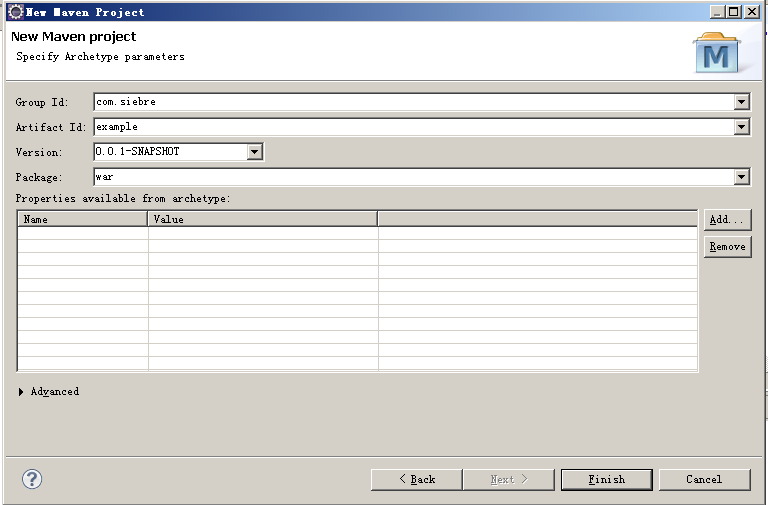


图5



图6

## 可视化pom.xml配置

双击Maven项目的树状结构中的pom.xml文件，即可打开pom.xml的可视化配置界面，如下图

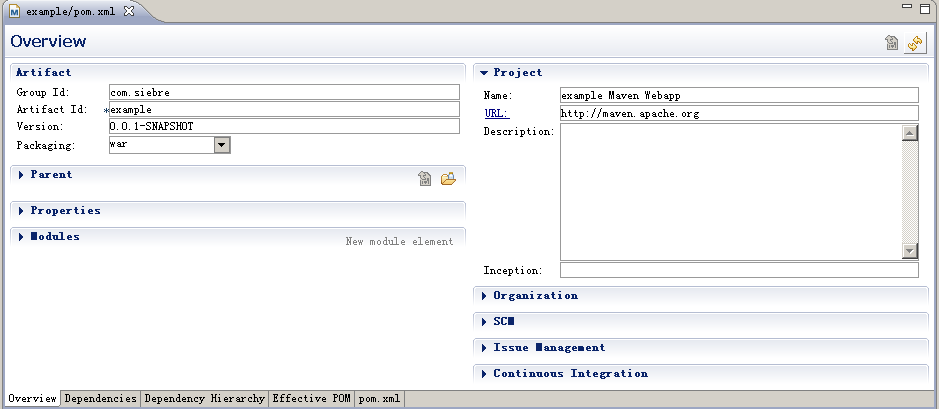


图7

该编辑器将XML语言抽象为图形界面方便开发者进行配置。我们可以在Dependencies标签页中点击Add…并利用插件提供的搜索功能来找寻我们需要的依赖构件（如jar包），如下图

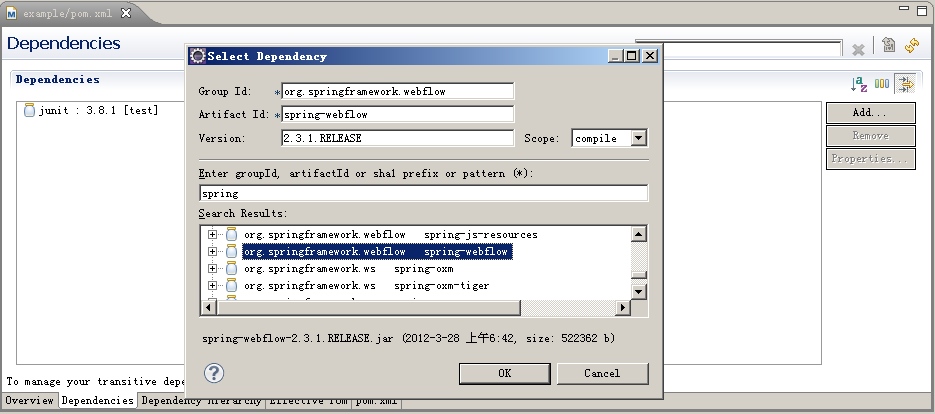


图8

Maven插件会根据pom.xml配置的变化自动下载该依赖构件，图中为spring-webflow，并会自动下载该依赖构件的子依赖构件（A->B, B->C …），此过程是闭包的。在下载jar包的同时，该插件还会尝试下载jar包对应的source包和JavaDoc包。我们可以在Dependency Hierarchy标签页中查看所有jar包互相依赖关系的树形列表以及解析后的完整依赖构件列表，如下图

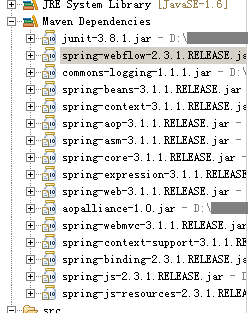


图9A

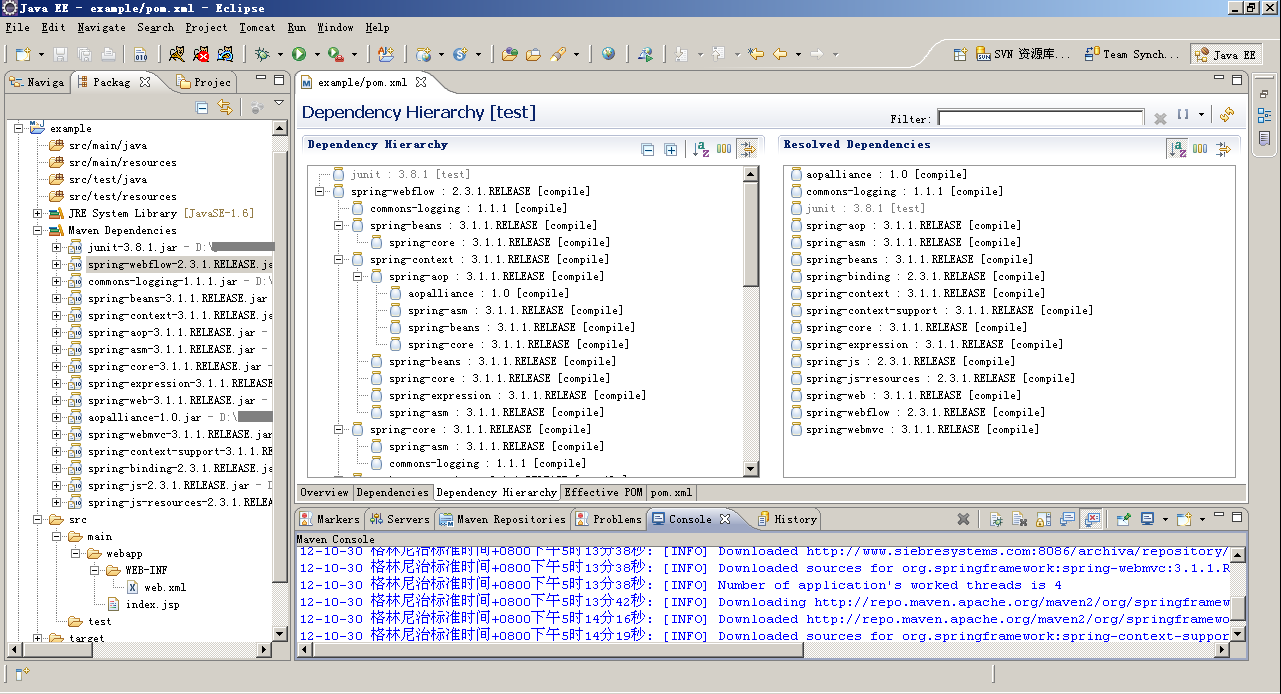


图9B

有时，直接编辑XML更方便（如配置插件时），我们可以在pom.xml标签页中直接进行编辑，并在Effective POM标签页中查看最终有效地配置，如下图

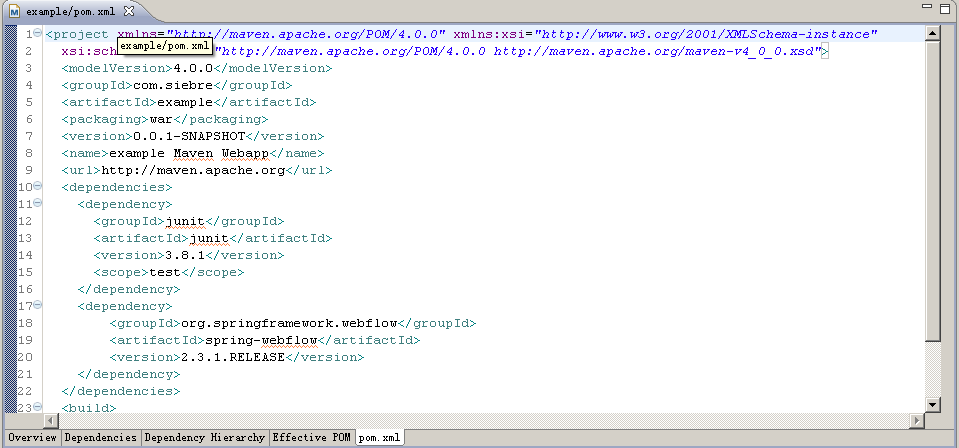


图10

## Deployment Assembly及Server

安装完Maven的WTP集成插件后，在开发过程中我们可以把Maven项目动态地发布到网络容器中。

右键项目进入Properties，选中Project Facets，确认Dynamic Web Module已被勾选，且Java的版本与IDE的Java编译器版本一致，如下图

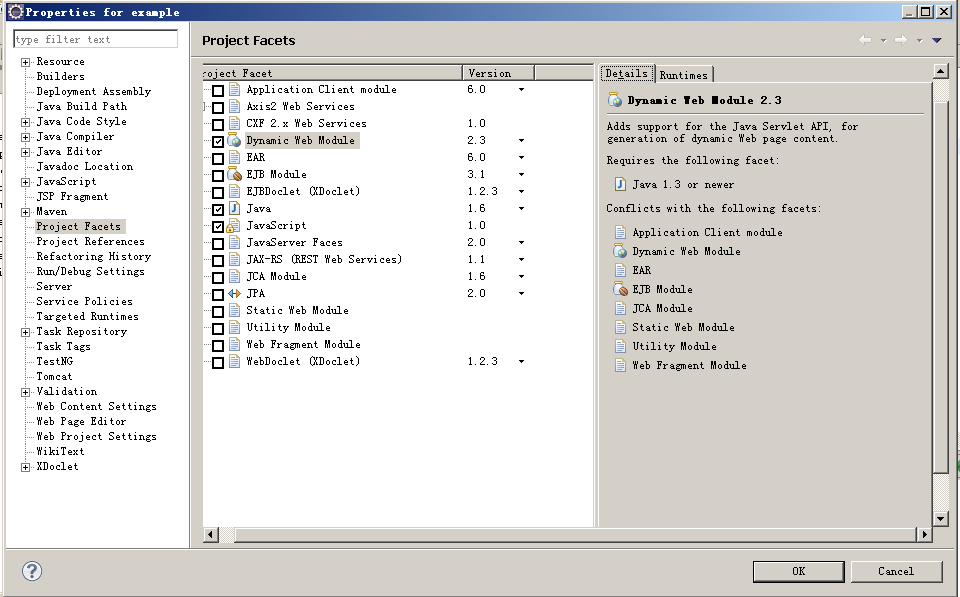


图11

选中Deployment Assembly，对各文件夹及Maven依赖管理的服务器部署位置进行设置，如下图

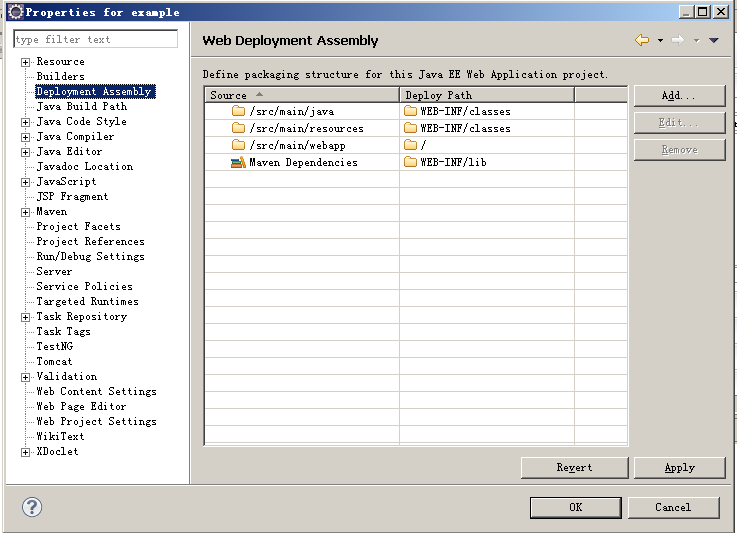


图12

接下来对服务器进行配置，打开Window->Show View->Servers，右键视图选择New->Server，选择一个服务器，如Tomcat V6.0，点击Next，如下图

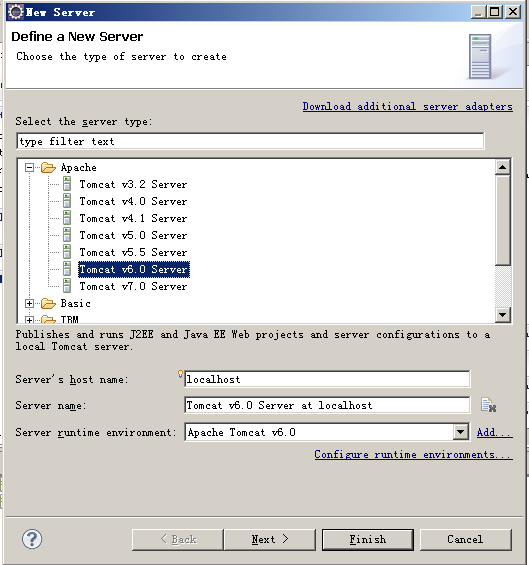


图13

在随后的视图中，将之前创建的Maven项目Add进服务器中，点击Finish，如下图

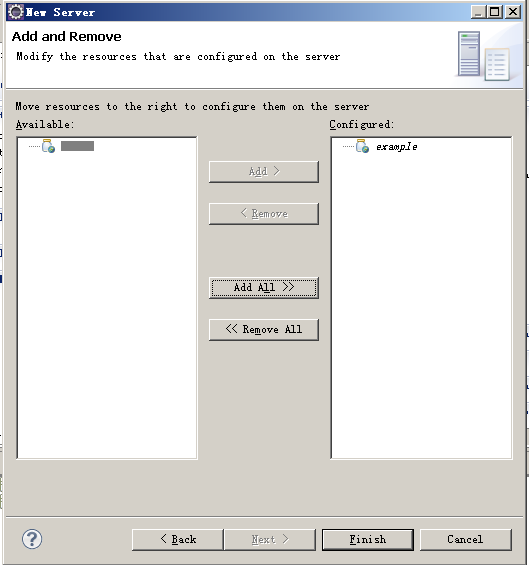


图14

创建完服务器后，双击该服务器进入配置页面，在Server Locations中点选Use Tomcat installation并填写项目部署路径如examplepath，保存配置。右键服务器选择Publish将项目中的文件按照Deployment Assembly中指定的部署时的相对路径推送到服务器中，如下图

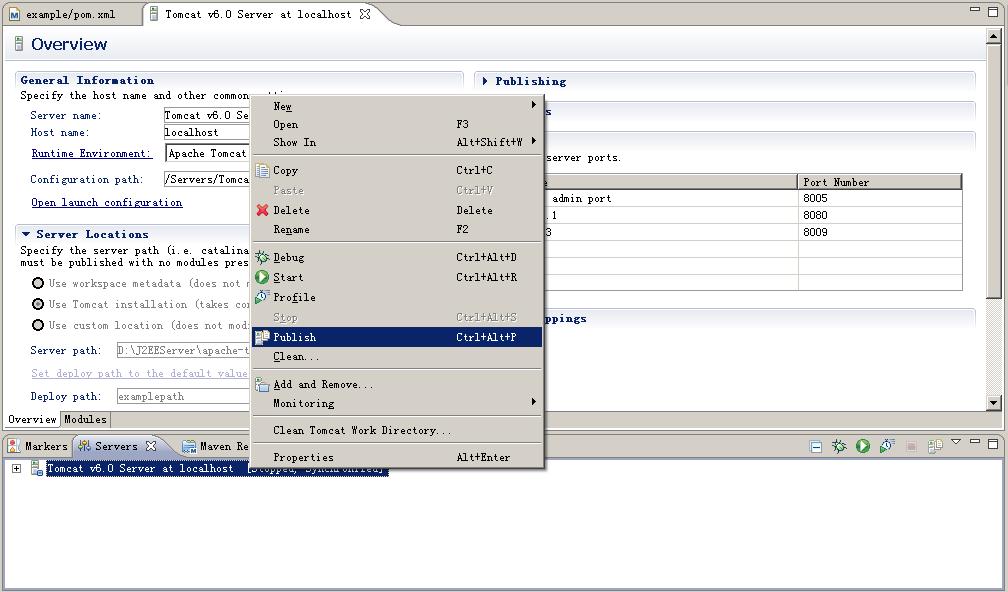


图15

推送完后，到服务器安装路径中查看，可以发现项目已经按照配置的路径被推送过来了，并且在WEB-INF\lib文件夹下，Maven管理的依赖构件也都被插件推送成功。通过观察，Scope为compile的依赖构件都被推送成功，而Scope为test的依赖构件没有被插件推送，说明Maven的WTP插件运行正常。见下图

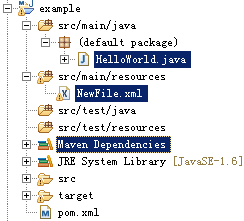


图16A

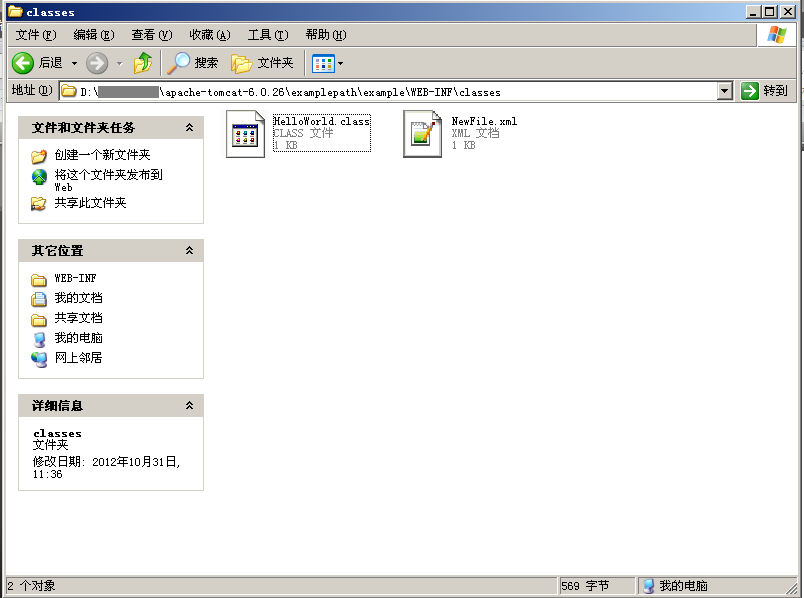


图16B

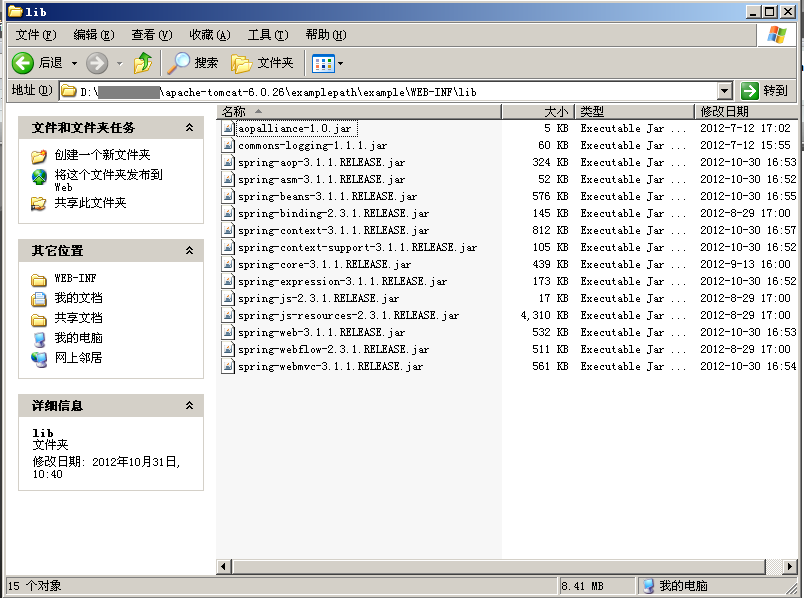


图16C

## Maven指令使用，Run As…

安装了m2e后，在IDE中可以直接对项目运行mvn指令。

右键项目选择Run As->Run Configurations…，如下图

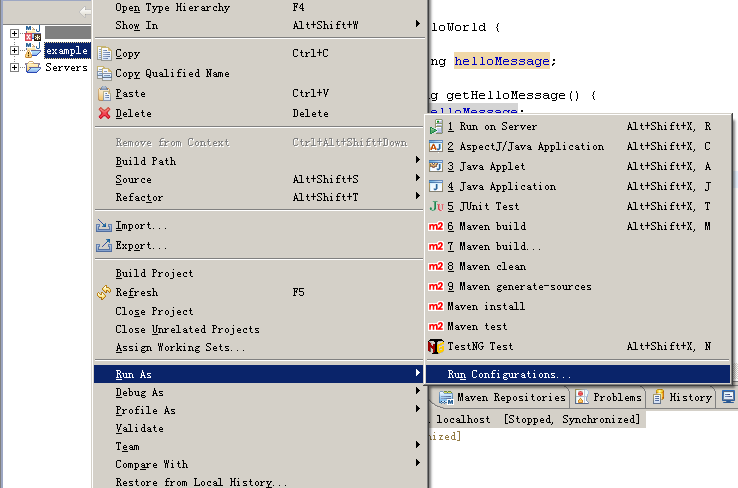


图17

右键选择Maven Build->New，在出现的配置页面中输入想要执行的mvn指令（图中为clean package），确认项目路径并为该操作命名后应用，如下图

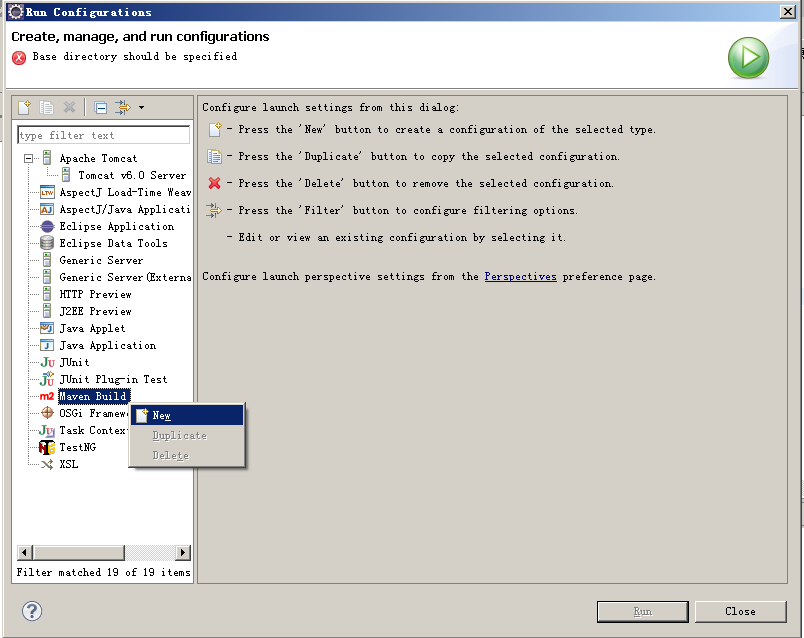


图18A

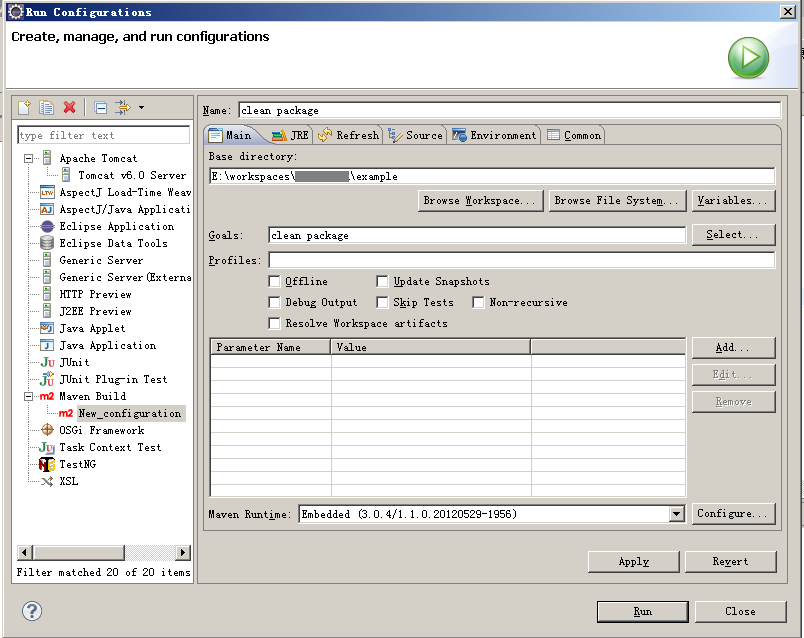


图18B

运行过程和结果会在Maven Console中显示，如下图

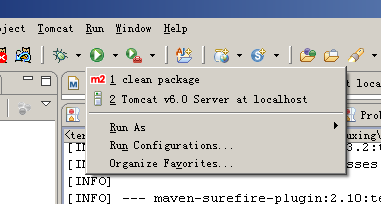


图19A

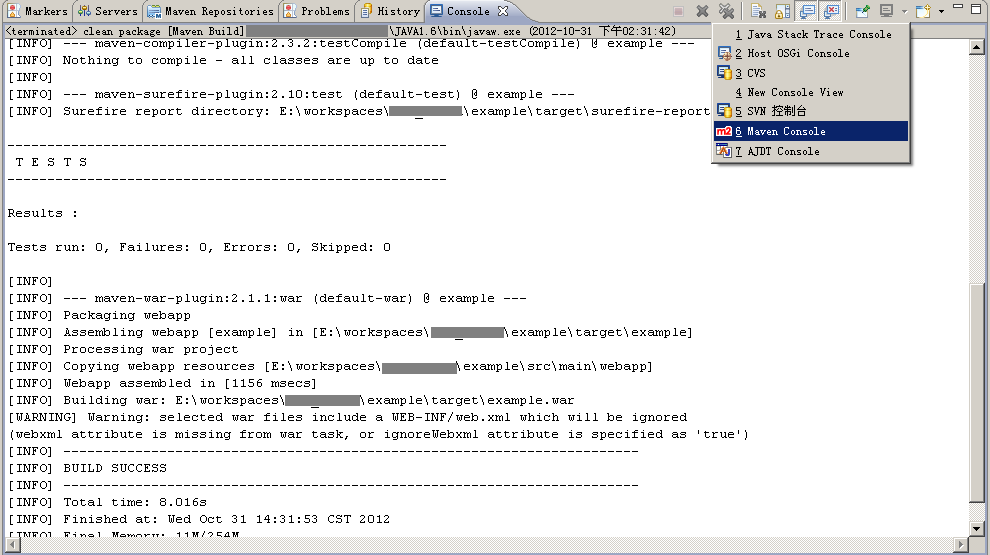


图19B

# 普通Web项目转化为Maven Web项目

要将过去的Web项目转化为Maven项目，首先需要调整的是项目的文件结构。接下来将会展示如何将DMS的项目转换为Maven项目。

## 示例，DMS项目结构

DMS原项目结构如下图，

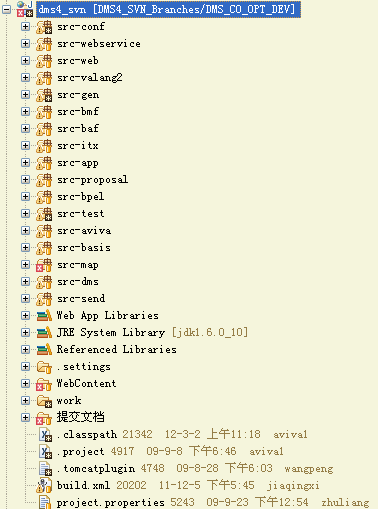


图20

其中



为源代码及资源文件夹



为原jar包引用库



为原项目的Web资源文件夹，WEB-INF/lib文件夹中包含项目用到的所有jar包，WEB-INF/classes文件夹中存放编译后的.class文件

## 转换，Maven化的DMS项目结构

如前文所述，按照Maven项目的[标准结构规范](#表1)，调整后的DMS项目应该形同[图6](#图6)中的项目结构。然而，由于原DMS项目的源代码及资源文件夹共有10个且彼此独立，很难把它们合并且整理分类到src\main\java及src\main\resources文件夹下，即使把这10个文件夹合并、整理分类为源码及资源两个文件夹，原来的开发人员也可能难以适应。所以DMS的项目结构调整的过程中，我们对源码及资源文件夹的结构做出妥协，保留原有10个文件夹的结构。相较于源码和资源文件夹，Web资源文件夹的调整则要简单许多，只需将原本的Web Content转移至src\main下并重命名为webapp。

转换前，先要在项目根目录处手动创建一个最基本的pom.xml文件（只含有项目发布者、项目名、版本及打包格式）。右键项目选择Configure->Convert to Maven Project，如下图，

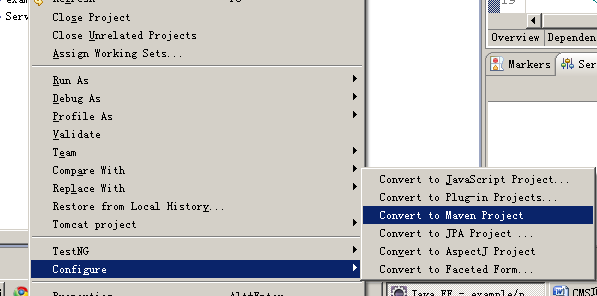


图21

需要注意的是，Convert操作进行前请务必确认项目classpath中没有variable的引用，否则m2e插件的转换操作将无法成功，记录变量，之后一起进行jar包替换，如下图

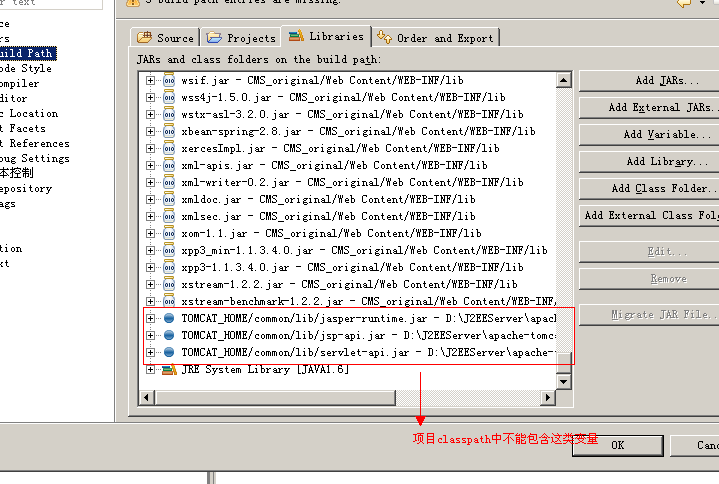


图22

调整完的DMS项目结构如下图

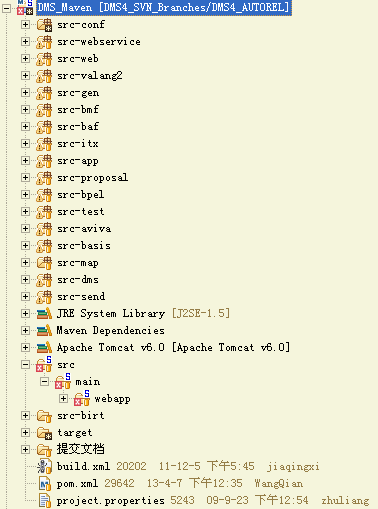
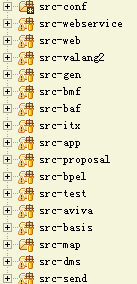


图23

其中



源代码及源文件夹结构被保留，之后将通过设置相应的Maven插件在项目构建的对应阶段进行调整以弥补该问题



原有的Referenced Libraries已经完全被Maven的依赖管理所取代从而支持Maven的自动化构建过程



原有的Web Content文件夹按照标准文件结构规范被转移至该处，其中WEB-INF/lib文件夹中的jar包被全部替换，故删除，WEB-INF/classes文件夹由于Maven编译输出的路程已经变更，故删除



Maven构建过程中的输出目标路径，自动生成



新增的Maven项目配置文件

要在开发过程中利用动态部署测试，需设置Project Facets及Deployment Assembly，如下图

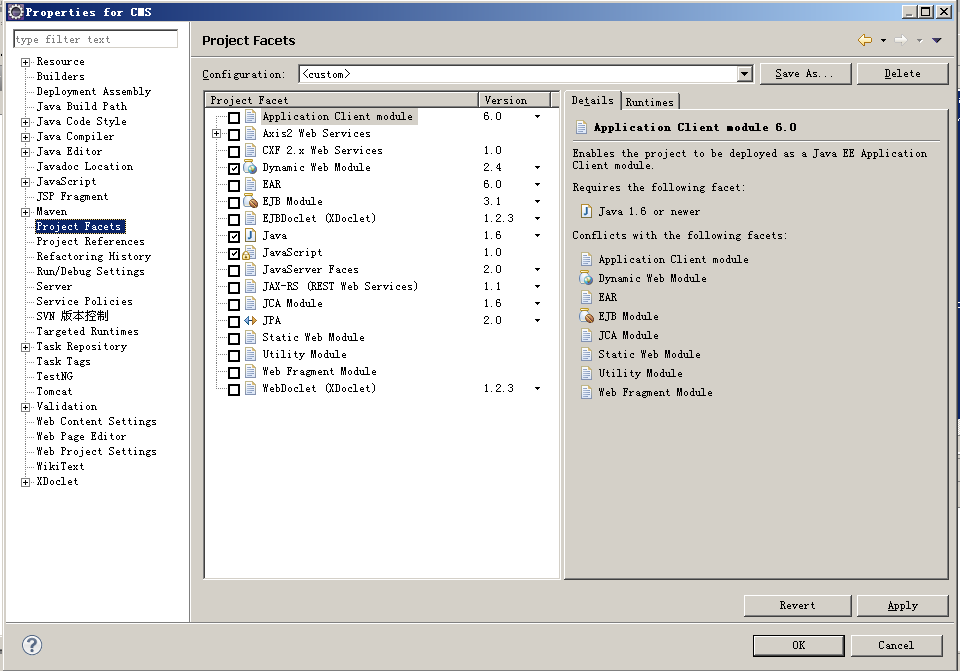


图24A

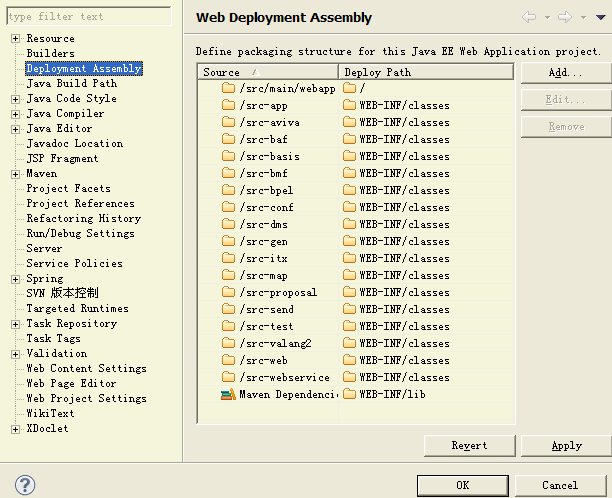


图24B

## 替换jar包

转换这类Web项目的工作中最为繁重的一项是将原项目Web Content\WEB-INF\lib文件夹中所有依赖的jar包替换并交由Maven管理。虽然在Eclipse IDE中由于m2e插件的支持（pom.xml文件编辑器、Maven库搜索、自动下载依赖构件并编译项目），替换jar包的效率得到显著提升，但是类似DMS这种规模的项目往往包含数百个jar包，所以工作量依然巨大。

该工作的具体操作步骤如下，

1. **获取要替换的jar包的版本（jar包的包名中、jar包内的**META-INF\MANIFEST.MF**文件内、其他手段）；**
2. **打开pom.xml文件编辑器的**Dependencies**标签页，点击**Add…**，输入关键字进行搜索，选择相应版本的jar包，保存pom.xml文件。若无法在搜索结果中找到相应版本，前往**<http://search.maven.org/>**进行搜索，然后在**Add…**中手动输入该jar包的版本，保存并让插件尝试下载。若仍找不到该版本，尝试下载最新版或接近原版本的jar包。如果找不到该jar包的任意版本，跳至4），检查该包是否必须保留；**
3. **待m2e插件下载该jar包完毕后，检查是否有该jar包传递性依赖的其他jar包被一同下载到**Maven Dependencies**中。可在pom.xml编辑器的**Dependency Hierarchy**标签页中的树形列表中查看新添加jar包的传递性依赖情况，然后在**Resolved Dependencies**列表中选择多余的因传递性依赖引入的jar包，右键选择**Exclude Maven Artifact…**，最后保存pom.xml文件；**
4. **从**WEB-INF\lib**文件夹中删除原jar包（转移到其他的临时位置），并在**Referenced Libraries**中去除该jar包的引用；**
5. **在IDE中**Build Project**（默认是自动Build），查看是否有新的错误。若没有则在**Server**中**Republish**该项目，启动该应用，检查是否正常。若没有问题，则该jar包替换成功（或不是必须的）；若出现问题则失败，无法在Maven依赖管理中找到可替换的jar包，记录该jar包，并回退删除及取消引用的操作，以继续其他jar包的替换；**
6. **重复1 )至5）直到所有jar包都被该流程处理过，此时原**WEB-INF\lib**文件夹中的所有jar包应被划分到以下几类，**
   1. **被成功替换；**
   2. **无法被替换但不是必须；**
   3. **无法被替换且必须；**
7. **对于无法被替换但又必须的jar包，我们需要将其手动地上传至预先搭建的公司Maven库中，再交由Maven的依赖管理以完成替换，即2）和4）；**
8. **最后删除WEB-INF/lib文件夹；**

**我在替换时确认某包是否必须的一种方法是利用IDE的文件搜索功能搜索该包中的路径有没有被该项目中的java或xml文件引用。**

## 配置plugin

替换完jar包后，该DMS项目还需要在pom.xml中对插件进行配置以完成下面的功能，

**在Maven编译插件工作时，告知Maven编译插件DMS项目的Java版本及编码格式**，配置如下

**<plugin>**

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

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

**<version>2.5.1</version>**

**<configuration>**

**<source>1.6</source>**

**<target>1.6</target>**

**<compilerVersion>1.6</compilerVersion>**

**<encoding>GBK</encoding>**

**</configuration>**

**</plugin>**

**在Maven构建过程中的读取源码文件及读取资源文件的阶段，告知Maven该项目除了**src\main\java**及**src\main\resources**这两个默认路径以外的源码及资源文件夹位置（同样可以指定测试源码及测试资源文件夹的默认路径以外的位置）**，配置如下

**<build>**

**<plugins>**

**<plugin>**

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

**<configuration>**

**<source>1.6</source>**

**<target>1.6</target>**

**<encoding>GBK</encoding>**

**</configuration>**

**</plugin>**

**<plugin>**

**<groupId>org.codehaus.mojo</groupId>**

**<artifactId>build-helper-maven-plugin</artifactId>**

**<version>1.7</version>**

**<executions>**

**<execution>**

**<id>add-source</id>**

**<phase>generate-sources</phase>**

**<goals>**

**<goal>add-source</goal>**

**</goals>**

**<configuration>**

**<sources>**

**<source>src-conf</source>**

**<source>src-webservice</source>**

**<source>src-web</source>**

**<source>src-valang2</source>**

**<source>src-gen</source>**

**<source>src-bmf</source>**

**<source>src-baf</source>**

**<source>src-itx</source>**

**<source>src-app</source>**

**<source>src-proposal</source>**

**<source>src-bpel</source>**

**<source>src-test</source>**

**<source>src-aviva</source>**

**<source>src-basis</source>**

**<source>src-map</source>**

**<source>src-dms</source>**

**<source>src-send</source>**

**</sources>**

**</configuration>**

**</execution>**

**<execution>**

**<id>add-resource</id>**

**<phase>generate-resources</phase>**

**<goals>**

**<goal>add-resource</goal>**

**</goals>**

**<configuration>**

**<resources>**

**<resource>**

**<directory>src-conf</directory>**

**<excludes>**

**<exclude>\*\*/\*.java</exclude>**

**</excludes>**

**</resource>**

**<resource>**

**<directory>src-webservice</directory>**

**<excludes>**

**<exclude>\*\*/\*.java</exclude>**

**</excludes>**

**</resource>**

**<resource>**

**<directory>src-web</directory>**

**<excludes>**

**<exclude>\*\*/\*.java</exclude>**

**</excludes>**

**</resource>**

**<resource>**

**<directory>src-valang2</directory>**

**<excludes>**

**<exclude>\*\*/\*.java</exclude>**

**</excludes>**

**</resource>**

**<resource>**

**<directory>src-gen</directory>**

**<excludes>**

**<exclude>\*\*/\*.java</exclude>**

**</excludes>**

**</resource>**

**<resource>**

**<directory>src-bmf</directory>**

**<excludes>**

**<exclude>\*\*/\*.java</exclude>**

**</excludes>**

**</resource>**

**<resource>**

**<directory>src-baf</directory>**

**<excludes>**

**<exclude>\*\*/\*.java</exclude>**

**</excludes>**

**</resource>**

**<resource>**

**<directory>src-itx</directory>**

**<excludes>**

**<exclude>\*\*/\*.java</exclude>**

**</excludes>**

**</resource>**

**<resource>**

**<directory>src-app</directory>**

**<excludes>**

**<exclude>\*\*/\*.java</exclude>**

**</excludes>**

**</resource>**

**<resource>**

**<directory>src-proposal</directory>**

**<excludes>**

**<exclude>\*\*/\*.java</exclude>**

**</excludes>**

**</resource>**

**<resource>**

**<directory>src-bpel</directory>**

**<excludes>**

**<exclude>\*\*/\*.java</exclude>**

**</excludes>**

**</resource>**

**<resource>**

**<directory>src-test</directory>**

**<excludes>**

**<exclude>\*\*/\*.java</exclude>**

**</excludes>**

**</resource>**

**<resource>**

**<directory>src-aviva</directory>**

**<excludes>**

**<exclude>\*\*/\*.java</exclude>**

**</excludes>**

**</resource>**

**<resource>**

**<directory>src-basis</directory>**

**<excludes>**

**<exclude>\*\*/\*.java</exclude>**

**</excludes>**

**</resource>**

**<resource>**

**<directory>src-map</directory>**

**<excludes>**

**<exclude>\*\*/\*.java</exclude>**

**</excludes>**

**</resource>**

**<resource>**

**<directory>src-dms</directory>**

**<excludes>**

**<exclude>\*\*/\*.java</exclude>**

**</excludes>**

**</resource>**

**<resource>**

**<directory>src-send</directory>**

**<excludes>**

**<exclude>\*\*/\*.java</exclude>**

**</excludes>**

**</resource>**

**</resources>**

**</configuration>**

**</execution>**

**</executions>**

**</plugin>**

**</plugins>**

**</build>**

至此，该项目转换为Maven项目的工作已全部完成。再下一个文档中，将展示如何搭建公司的Maven库，以及如何利用改造完的DMS项目的Maven特性进行自动化发布。