**一、前言**

我们在《万能框架spring》前四天中都用到了maven，接下去要讲述在SSX这样的架构下我们的“单元测试”是怎么进行的，但是在此之前我们再来深入入解一下maven，因为我们的单元测试需要用到的是junit+ant+junitreport这样的组合。而。。。。。。由于我们已经使用了maven，那么我们如何可以延续经典的junit+ant这样的单元测试的组合呢？其答案就是把maven和ant再进行组合一下。

**二、用Maven任务在Ant中使用Maven依赖**

Maven的一个重要特性是使用中央仓库，存储依赖和识别应用程序所需的库。Maven2还支持传递依赖（transitivedependency），这是一个强大的概念，它将用户需要声明的依赖限制到最小程度。Ant不支持“即开即用”的依赖管理。在Ant项目中，应用程序所需的全部程序库通常都存放在一个项目目录中。

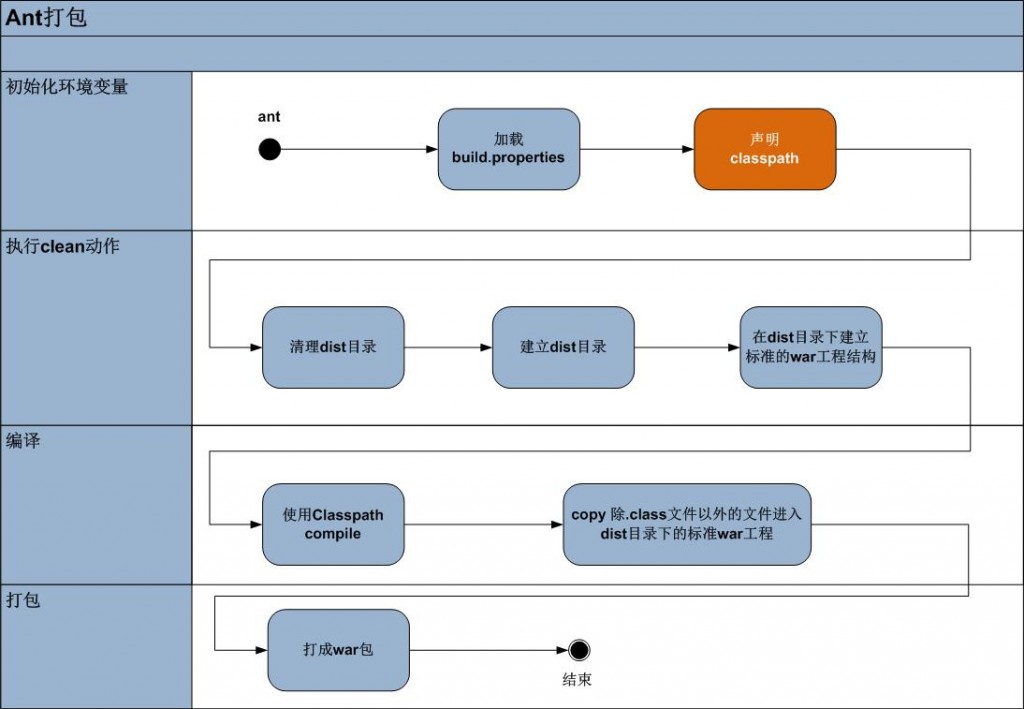
**三、使用maven-ant-task**

**3.1安装maven-ant-task**

我们先下载maven-ant-tasks-2.1.3.jar（我也已经上传到了“我的资源”里了）；

下载完后把它放到我们的ant的lib目录下；

**3.2 打包工程**

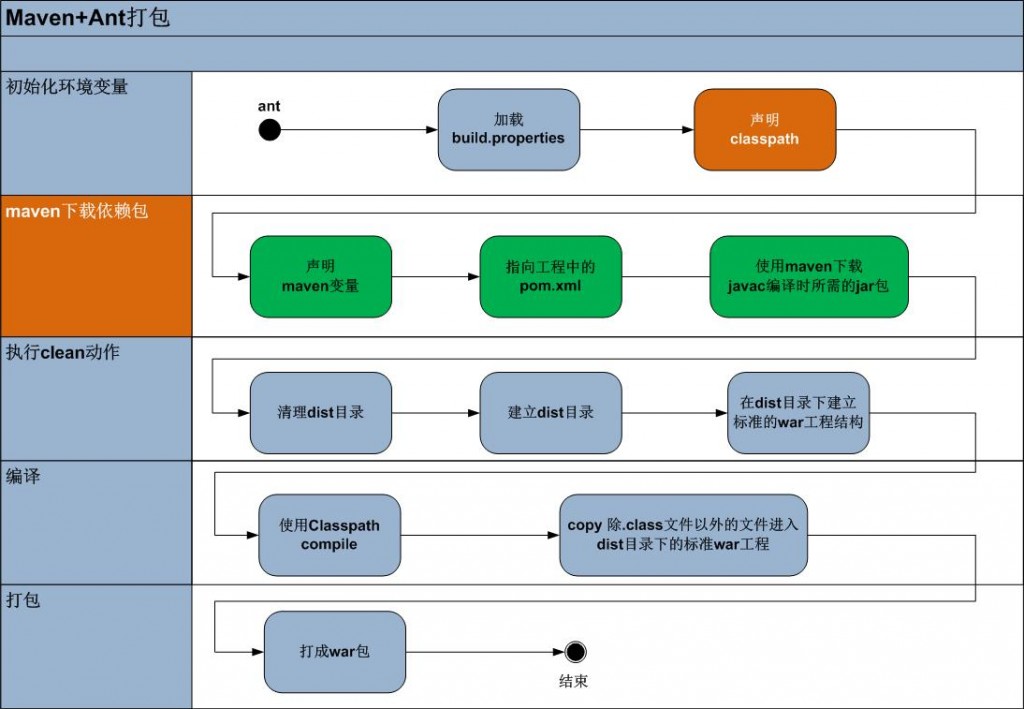


这是一个传统的没有使用maven依赖库的ant的打包流程，从图上看很清晰，对吧？

在前面说了，ant打包时是需要所有的.jar文件都是事先已经在工程的WebContent/WEB-INF/lib目录下了，这也是大都数人所熟悉的。

但是Maven打包，就一个包依赖声明文件pom.xml，然后在打包时缺什么jarMaven会自动帮我们去下载，所以说Maven真是一个好东西，使用maven-ant-task我们可以把“**声明classpath**”这块来用maven-ant-task来做，即其它还是使用ant，就是在声明编译时即javac时需要用到的那些依赖包时再使用maven的特性。

因此，我们再来看上述这个纯ant工程打包流程在引入了maven-ant-task后会变成什么样子吧。

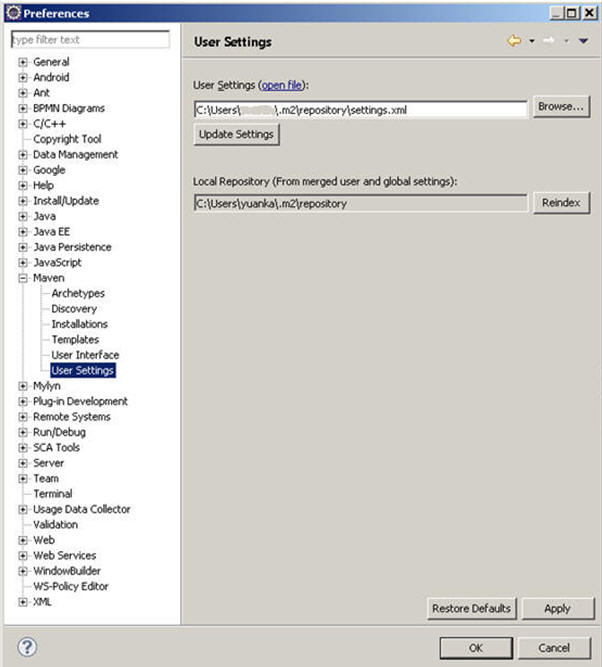


**四、动手改造工程**

我们此处以myssh2的工程来做例子，原来的pom.xml文件不动，增加ant的打包文件即build.xml文件。

**4.1maven环境的重设置**

我们前面的几天教程用的都是一个叫maveneclipse(m2 eclipse)的插件



它默认会把所有的maven依赖库下载到你的**C:\Users\yourusername\.m2\repository**这个目录下（**以win7为例，如果你用的是XP那么这个目录应该在你的c:\documents and settings目录下**），因此我们需要手动把我们的maven安装目录的下的conf\目录下的settings.xml文件也拷贝到

C:\Users\yourusername\.m2\repository目录下，还记得吗？

现在，我们在我们的apache-maven的安装目录即c:\maven目录下建立我们的repository，并且让m2eclipse插件也将repository指向C:\maven\repository。

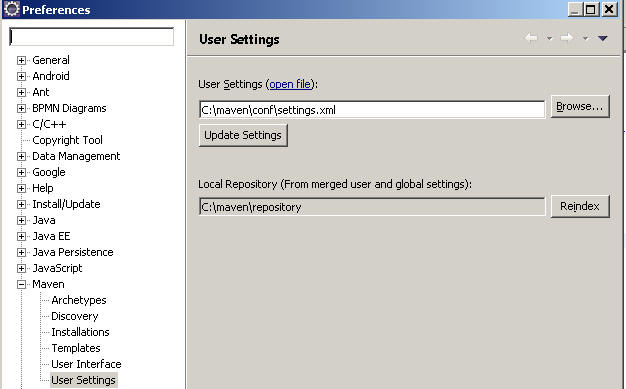
**更改c:\maven\conf\settings.xml文件**



我们在原有的settings.xml文件中增加了如下的一段话

|  |
| --- |
| **<localRepository>C:\maven\repository</localRepository>** |

然后回到eclipse点开window->preference->Maven->UserSettings，将我们的m2eclipse的设置配置成如下的样子



**注意：**

*Local Repository再也不是“****C:\Users\yourusername\.m2\repository****”了，而是指向了我们的*

*c:\maven\repository目录了。*

*这样做还有一个好处就是你在重新装机时你的maven的local repository不会因为documents and settings目录的丢失而丢失了。*

**4.2编写build.xml**

**build.properties文件**

|  |
| --- |
| **# ant**  **appName=myssh2**  **webAppName=myssh2**  **webAppQAName=myssh2-UT**  **local.dir=C:/eclipsespace/${appName}**  **src.dir=${local.dir}/src/main/java**  **dist.dir=${local.dir}/dist**  **webroot.dir=${local.dir}/src/main/webapp**  **lib.dir=${local.dir}/lib**  **classes.dir=${webroot.dir}/WEB-INF/classes**  **resources.dir=${local.dir}/src/main/resources** |

请注意此处的**${}**这样的placeholder（替换符）的使用技巧

**build.xml文件（红色加粗部分）**

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <project name="myssh2" default="buildwar"  **xmlns:artifact="urn:maven-artifact-ant">**    **<property file="build.properties" />**  **<path id="maven-ant-tasks.classpath" path="C:/ant/lib/maven-ant-tasks-2.1.3.jar" />**  **<typedef resource="org/apache/maven/artifact/ant/antlib.xml" uri="urn:maven-artifact-ant"**  **classpathref="maven-ant-tasks.classpath" />**  **<artifact:pom id="maven.project" file="pom.xml" />**  **<artifact:dependencies filesetId="deps.fileset.compile" useScope="compile">**  **<!--<pom file="pom.xml"/>-->**  **<pom refid="maven.project" />**  **</artifact:dependencies>**  **<path id="compile.classpath">**  **<fileset dir="${lib.dir}">**  **<include name="\*.jar" />**  **</fileset>**  **</path>**     <target name="clean" description="Delete old build and dist directories">           <delete dir="${dist.dir}" />           <mkdir dir="${dist.dir}" />           <mkdir dir="${dist.dir}/${webAppName}" />           <mkdir dir="${dist.dir}/${webAppName}/WEB-INF" />           <mkdir dir="${dist.dir}/${webAppName}/WEB-INF/lib" />            <mkdir dir="${dist.dir}/${webAppName}/WEB-INF/classes" />            <mkdir dir="${dist.dir}/${webAppName}/css" />            <mkdir dir="${dist.dir}/${webAppName}/images" />            <mkdir dir="${dist.dir}/${webAppName}/jsp" />     </target>  **<target name="download-libs" depends="clean">**  **<copy todir="${lib.dir}">**  **<fileset refid="deps.fileset.compile" />**  **<mapper type="flatten" />**  **</copy>**  **</target>**    <target name="compile" description="Compile java sources" depends="download-libs">  **<javac debug="true" destdir="${dist.dir}/${webAppName}/WEB-INF/classes"**  **includeAntRuntime="false" srcdir="${src.dir}">**  **<classpath refid="compile.classpath" />**  **</javac>**             <copy todir="${dist.dir}/${webAppName}/WEB-INF/lib">                         <fileset dir="${lib.dir}">                           <include name="\*.jar" />                          </fileset>            </copy>             <copy todir="${dist.dir}/${webAppName}/WEB-INF/classes">                         <fileset dir="${resources.dir}">                          <include name="\*\*/\*.\*" />                          </fileset>            </copy>            <copy todir="${dist.dir}/${webAppName}/css">                         <fileset dir="${webroot.dir}/css">                           <include name="\*\*/\*.\*" />                         </fileset>            </copy>            <copy todir="${dist.dir}/${webAppName}/images">                        <fileset dir="${webroot.dir}/images">                          <include name="\*\*/\*.\*" />                        </fileset>            </copy>            <copy todir="${dist.dir}/${webAppName}/jsp">                       <fileset dir="${webroot.dir}/jsp">                             <include name="\*\*/\*.\*" />                       </fileset>           </copy>           <copy todir="${dist.dir}/${webAppName}">                      <fileset dir="${webroot.dir}">                        <include name="\*.\*" />                     </fileset>          </copy>          <copy todir="${dist.dir}/${webAppName}/WEB-INF">                   <fileset dir="${webroot.dir}/WEB-INF">                        <include name="\*.\*" />                   </fileset>          </copy>    </target>    <target name="buildwar" depends="compile">          <war warfile="${dist.dir}/${webAppName}.war">            <fileset dir="${dist.dir}/${webAppName}"/>          </war>    </target>  </project> |

对照着**Maven+Ant打包**的流程图看这个build.xml文件，一点也不难看懂，就一个过程化的批处理脚本而己。

关键在于：

|  |
| --- |
| **<path id="maven-ant-tasks.classpath" path="C:/ant/lib/maven-ant-tasks-2.1.3.jar" />**  **<typedef resource="org/apache/maven/artifact/ant/antlib.xml" uri="urn:maven-artifact-ant"**  **classpathref="maven-ant-tasks.classpath" />**  **<artifact:pom id="maven.project" file="pom.xml" />**    **<artifact:dependencies filesetId="deps.fileset.compile" useScope="compile">**  **<!--<pom file="pom.xml"/>-->**  **<pom refid="maven.project" />**  **</artifact:dependencies>**    **<path id="compile.classpath">**  **<fileset dir="${lib.dir}">**  **<include name="\*.jar" />**  **</fileset>**  **</path>**  **<target name="download-libs" depends="clean">**  **<copy todir="${lib.dir}">**  **<fileset refid="deps.fileset.compile" />**  **<mapper type="flatten" />**  **</copy>**  **</target>** |

此处声明了：

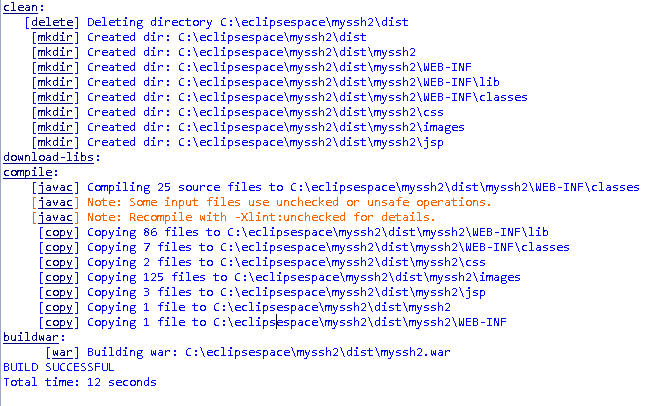
1）  我们在ant的build.xml文件中使用maven插件

2）  我们的工程所用到的依赖库即所有的.jar文件在工程（myssh2）原有工程的pom.xml文件中已经声明过依赖关系了，build.xml文件只要使用它就可以了。

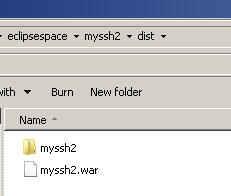
3）  我们的ant在使用javac进行编译时所需的库在${lib.dir}，见build.properties即在c:\eclipsespace\myssh2\lib目录内。

4）  c:\eclipsespace\myssh2\lib目录内所有的jar则是通过“download-libs”这个target得来的。

下面是build.xml文件被运行后的输出效果即在eclipse中打开build.xml文件后右键选RunAs->Ant Build的输出：



去我们的工程的dist目录下查看，就可以看到一个myssh2.war工程



把这个工程直接扔到tomcat的webapp目录下，运行tomcat后登录<http://localhost:8080/myss2> 后一切正常，结束本天的教程。

**PS：**

一般我不喜欢把一个.war包扔到tomcat下去，而是喜欢把符合war目录结构的这样的一个目录扔到tomcat的webapp目录下，直到这个工程成品了，即通过测试后再打成war包以便于tomcat或者是weblogic、WAS的布署，因此对于我来说在**<target name="compile"depends="download-libs">这步完成后就够用了。**