Maven入门--概念与实例（转载）

« 用 Maven 做项目管理 | 首页 | maven2.0学习笔记（转载） »

最近由于工作原因在研究、应用Maven，有了一些体会就写成了此文。本文虽然是Maven2的入门文章，但并不涉及Maven的历史、下载与安装，这些内容可以到Maven的官方网站上了解。本文主要是关注Maven中的重要概念，并以一个实例来阐述使用Maven的 基本方法。文末有例子代码下载的链接。

1 关键名词

Project：任何您想build的事物，Maven都可以认为它们是工程。这些工程被定义为工程对象模型(POM，Poject Object Model)。一个工程可以依赖其它的工程；一个工程也可以由多个子工程构成。

POM：POM(pom.xml)是Maven的核心文件，它是指示Maven如何工作的元数据文件，类似于Ant中的build.xml文件。POM文件位于每个工程的根目录中。

GroupId：groupId是一个工程的在全局中唯一的标识符，一般地，它就是工程名。groupId有利于使用一个完全的包名，将一个工程从其它有类似名称的工程里区别出来。

Artifact：artifact 是工程将要产生或需要使用的文件，它可以是jar文件，源文件，二进制文件，war文件，甚至是pom文件。每个artifact都由groupId和 artifactId组合的标识符唯一识别。需要被使用(依赖)的artifact都要放在仓库(见Repository)中，否则Maven无法找到 (识别)它们。

Dependency：为了能够build或运行，一个典型的Java工程会依赖其它的包。在Maven中，这些被依赖的包就被称为dependency。dependency一般是其它工程的artifact。

Plug-in：Maven是由插件组织的，它的每一个功能都是由插件提供的。插件提供goal(类似于Ant中的target)，并根据在POM中找到的元数据去完成工作。主要的Maven插件要是由Java写成的，但它也支持用Beanshell或Ant脚本写成的插件。

Repository：仓库用于存放artifact，它可以是本地仓库，也可以是远程仓库。Maven有一个默认的远程仓库--central，可以从http://www.ibiblio.org/maven/下载其中的artifact。在Windows平台上，本地仓库的默认地址是*User_Home*\.m2\repository。

Snapshot：工程中可以(也应该)有一个特殊版本，它的版本号包括*SNAPSHOT*字样。该版本可以告诉Maven，该工程正处于开发阶段，会经常更新(但还未发布)。当其它工程使用此类型版本的artifact时，Maven会在仓库中寻找该artifact的最新版本，并自动下载、使用该最新版。

2 Maven Build Life Cycle

最近更新

IDC称今年Linux软件全球市场规模将增长21%
商业周刊：Twitter丰产不丰收前景受质疑
Amazon CloudFront Content Delivery Network
工信部李毅中：进一步降低电信资费
Hitwise：搜索引擎仍是新闻网站最大流量来源
PHP168 V6热点功能抢先预览
求职最受欢迎的十大技能
人民币国际结算试点花开沪粤
法国互联网广告价格或将持续走低
VMWare推出在线"程序成本"计算器

软件项目一般都有相似的开发过程：准备，编译，测试，打包和部署，Maven将上述过程称为Build Life Cycle。在Maven中，这些生命周期由一系列的短语组成，每个短语对应着一个(或多个)操作；或对应着一个(或多个)goal(类似于Ant中的 target)。

如编译源文件的命令mvn compile中的compile是一个生命周期短语。同时该命令也可以等价于mvn compiler:compile，其中的compiler是一个插件，它提供了compile(此**compile**与**mvn compile**中的**compile**意义不同)goal；compiler还可提供另一个goal--testCompile，该goal用于编译junit测试类。

在执行某一个生命周期时，Maven会首先执行该生命周期之前的其它周期。如要执行compile，那么将首先执行validate，generate- source，process-source和generate-resources，最后再执行compile本身。关于*Maven*中默认的生命周期短语，请见参考资源*[6]*中的附录*B.3*。

3 标准目录布局

Maven为工程中的源文件，资源文件，配置文件，生成的输出和文档都制定了一个标准的目录结构。Maven鼓励使用标准目录布局，这样就不需要进行额外 的配置，而且有助于各个不同工程之间的联接。当然，Maven也允许定制个性的目录布局，这就需要进行更多的配置。关于*Maven*的标准目录布局，请见参考资源*[6]*中的附录*B.1*。

4 Maven的优点

[1]build逻辑可以被重用。在Ant中可能需要多次重复地写相同的语句，但由于POM的继承性，可以复用其它的POM文件中的语句。这样既可以写出清晰的build语句，又可以构造出层次关系良好的build工程。

[2]不必关注build工作的实现细节。我们只需要使用一些build生命周期短语就可以达到我们的目标，而不必管Maven是如何做到这些的。如，只需要告诉Maven要安装(install)，那么它自然就会验证，编译，打包，及安装。

[3]Maven会递归加载工程依赖的artifact所依赖的其它artifact，而不用显示的将这些artifact全部写到dependency中。

[4]如果完全使用Maven的标准目录布局，那么可以极大地减少配置细节。

5 实例

5.1 构想

由于只是阐述Maven的基本使用方法，所以本文将要设计的实例，只是一个简单的Maven demo。该实例包含两个工程：普通应用程序工程(app)和Web应用工程(webapp)。app工程提供一个简单的Java类；webapp工程只 包含一个Servlet，并将使用app中的Java类。

该Demo的目标是能够正确地将webapp制成war包，以供部署时使用。要能够正确制作war，自然首先就必须要能够正确的编译源代码，且要将App模块制成jar包。本文创建的工程所在的目录是D:\maven\demo。

5.2 App工程

可以使用Maven的archetype插件来创建新工程，命令如下：

```
D:\maven\demo>mvn archetype:create -DgroupId=ce.demo.mvn -DartifactId=app
```

该工程的groupId是ce.demo.mvn，那么该工程的源文件将放在Java包ce.demo.mvn中。artifactId是app，那么该工程根目录的名称将为app。

当第一次执行该命令时，Maven会从central仓库中下载一些文件。这些文件包含插件archetype，以及它所依赖的其它包。该命令执行完毕后，在目录D:\maven\demo下会出现如下目录布局：

```
app
|-- pom.xml
```

```
`-- src
  |-- main
  |   `-- java
  |       `-- ce
  |           `-- demo
  |               `-- mvn
  |                   `-- App.java
  `-- test
      `-- java
          `-- ce
              `-- demo
                  `-- mvn
                      `-- AppTest.java
```

因本文暂时不涉及JUnit测试，故请将目录app\src\test目录删除。然后再修改App.java文件，其完全内容如下：

```
package ce.demo.mvn;
public class App {
    public String getStr(String str) {
        return str;
    }
}
```

其实，如果我们能够清楚地知道Maven的标准目录布局，就可以不使用archetype插件来创建工程原型；如果我们要定制个性的目录布局，那么就更没有必要使用archetype插件了。

5.3 WebApp工程

我们仍然如创建app工程一样使用archetype插件来创建webapp工程，命令如下：

D:\maven\demo>mvn archetype:create -DgroupId=ce.demo.mvn -DartifactId=webapp -DarchetypeArtifactId=maven-archetype-webapp

第一次运行此命令时，也会从central仓库中下载一些与Web应用相关的artifact(如javax.servlet)。此命令与创建app的命 令的不同之处是，多设置了一个属性archetypeArtifacttId，该属性的值为maven-archetype-webapp。即告诉 Maven，将要创建的工程是一个Web应用工程。创建app工程时没有使用该属性值，是由于archetype默认创建的是应用程序工程。同样的，执行 完该命令之后，会出现如下标准目录布局：

```
webapp
|-- pom.xml
`-- src
```

```
  `-- main
     `-- webapp
        |-- index.jsp
        |-- WEB-INF
           `-- web.xml
```

根据5.1节的构想，webapp工程将只包含一个Servlet，所以我们不需要index.jsp文件，请将其删除。此时大家可以发现，目前的目录布 局中并没有放Servlet，即Java源文件的地方。根据参考资源[6]中的附录B.1，以及app工程中Java源文件的布局，可以知道Servlet（它仍然是一个Java类文件）仍然是放在webapp\src\main\java目录中，请新建该目录。此处的Servlet是一个简单HelloServlet，其完整代码如下：

```
package hello;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import ce.demo.mvn.App;  // 引用app工程中的App类

public class HelloServlet extends HttpServlet {
    private static final long serialVersionUID = -3696470690560528247L;
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        App app = new App();
        String str = app.getStr("CE Maven Demo");
        PrintWriter out = response.getWriter();
        out.print("<html><body>");
        out.print("<h1>" + str);
        out.print("</body></html>");
    }
}
```

5.4 POM文件

大家可以发现，在前面新建工程时，我们并没有提到各个工程中的pom.xml文件。现在将要讨论这个问题。我们先看看app工程中的POM文件，其完整内容如下：

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>ce.demo.mvn</groupId>
  <artifactId>app</artifactId>
  <packaging>jar</packaging>
  <version>1.0</version>
  <name>CE Maven Demo -- App</name>
</project>
```

大家可以发现此我帖出来的内容与实际由archetype插件生成的POM文件的内容有些不同，但基本上是一致的。只是为了使文件中的语句更清晰，此处删 除了一些冗余的内容，并修改了该工程的version和name的值，以与此例子的背景来符合。在目前情况下modelVersion值将被固定为 4.0.0，这也是Maven2唯一能够识别的model版本。groupId，artifactId的值与创建工程时使用的命令中的相关属性值是一致 的。packaging的值由工程的类型决定，如应用程序工程的packaging值为jar，Web应用工程的packaging值为war。上述情况 也可以从webapp的POM文件中看出，下面将看看这个pom的完整内容。

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>ce.demo.mvn</groupId>
  <artifactId>webapp</artifactId>
  <packaging>war</packaging>
  <version>1.0</version>
  <name>CE Maven Demo -- WebApp</name>

  <dependencies>
     <dependency>
        <groupId>ce.demo.mvn</groupId>
        <artifactId>app</artifactId>
        <version>1.0</version>
     </dependency>
    <dependency>
       <groupId>javax.servlet</groupId>
       <artifactId>servlet-api</artifactId>
       <version>2.4</version>
       <scope>provided</scope>
    </dependency>
  </dependencies>
</project>
```

比较app与webapp中的POM，除前面已经提过的packaging的差别外，我们还可以发现webapp中的POM多了dependencies项。由于webapp需要用到app工程中的类(见HelloServlet源代码)，它还需要javax.servlet包(因为该包并不默认存在于 jsdk中)。故，我们必须要将它们声明到依赖关系中。

5.5 执行

上述两个工程创建完毕后，就需要执行一些命令来看看会有什么结果出现。我们首先进入app目录，并执行命令mvn compile，然后会在该目录下发现新生成的目录target\classes，即编译后的class文件(包括它的包目录)就放在了这里。再执行命令mvn package，在目录target中就会生成app-1.0.jar文件。该文件的全名由如下形式确定：**artifactId-version.packaging**。根据第2章的叙述可以知道，执行命令mvn package时，将首先将产生执行命令mvn compile之后的结果，故如果要打包，那么只需要执行mvn package即可。

在app工程中执行完之后，就需要进入webapp工程了。进入webapp目录，此次将只执行mvn package命令(隐示地跳过了compile过程)。此次命令的执行并不成功，会出现如下问题：

```
D:\maven\demo\webapp>mvn package
......
Downloading: http://repo1.maven.org/maven2/ce/demo/mvn/app/1.0/app-1.0.pom
[INFO] ------------------------------------------------------------------------
[ERROR] BUILD ERROR
[INFO] ------------------------------------------------------------------------
[INFO] Error building POM (may not be this project's POM).
Project ID: ce.demo.mvn:app
Reason: Error getting POM for 'ce.demo.mvn:app' from the repository: Error transferring file
  ce.demo.mvn:app:pom:1.0
from the specified remote repositories:
  central (http://repo1.maven.org/maven2)
......
```

由加粗的内容可知，Maven正试图从central仓库下载app工程的artifact，但central仓库肯定不会有这个artifact，其结 果只能是执行失败!由第1章artifact名词的解释可知，被依赖的artifact必须存在于仓库(远程或本地)中，但目前webapp所依赖的 app必不存在于仓库中，所以执行只能失败。

解决这个问题有两种方法：[1]将app-1.0.jar安装到仓库中，使它成为一个artifact；[2]构建一个更高层次的工程，使app和webapp成为这个工程的子工程，然后从这个更高层次工程中执行命令。

第一种方法比较简单（见http://www.blogjava.net/jiangshachina/admin/EditPosts.aspx中的第一个主题），此处将详细讨论第2种方法(见5.6节)。

5.6 更高层次工程

我们可以将app和webapp的上一级目录demo作为这两个工程的更高层次一个工程。为了使demo目录成为一个demo工程，只需要在这个目录下添加一个pom.xml文件，该文件内容如下：

```
<project>
    <modelVersion>4.0.0</modelVersion>
    <groupId>ce.demo</groupId>
    <artifactId>mvn-demo</artifactId>
    <packaging>pom</packaging>
    <version>1.0</version>
    <name>CE Maven Demo</name>

    <modules>
        <module>app</module>
        <module>webapp</module>
    </modules>
</project>
```

与app和webapp中的POM相比，demo的POM使用了modules项，modules用于声明本工程的子工程，module中的值对应于子工程的artifact名。而且该POM的packaging类型为pom。

有了demo工程后，我们只需要在demo目录下执行相关命令就可以了。通过如下命令即可验证:

[1]mvn clean – 消除工程(包括所有子工程)中产生的所有输出。这本文的实例中，实际上是删除target目录。由于之前的操作只有app工程产生了target目录，而webapp并没有，所以将只会删除app工程中的target目录。

[2]mvn package – 将工程制作成相应的包，app工程是作成jar包(app-1.0.jar)，webapp工程是作成war包(webapp-1.0.war)。打开webapp-1.0.war包，可以发现app-1.0.jar被放到了WEB-INF的lib目录中。

6 小结

通过以上的叙述与实例，应该可以对Maven有一个粗略的认识了。使用Maven关键是要弄清楚如何写pom.xml文件，就如同使用Ant要会写 build.xml文件一样。在POM中可以写入Ant的task脚本，也可以直接调用Ant的build.xml文件(推荐)，所以Maven也可以完 成Ant的绝大多数工作(但不必安装Ant)。

利用好Maven的继承特性及子工程的关系，可以很好地简化POM文件，并构建层次结构良好的工程，有利于工程的维护。

7 参考资源

[1]Maven官方网站. http://maven.apache.org

[2]Maven POM文件参考结构. http://maven.apache.org/ref/current/maven-model/maven.html

[3]Super POM. http://maven.apache.org/guides/introduction/introduction-to-the-pom.html

[4]Maven主要插件的列表. http://maven.apache.org/plugins

[5]Maven基本使用指南. http://maven.apache.org/guides/index.html

[6]Better Build with Maven. http://www.mergere.com/m2book_download.jsp -- 强烈推荐

[7]介绍Maven2. http://www.javaworld.com/javaworld/jw-12-2005 /jw-1205-maven_p.html

[8]揭秘Maven2 POM. http://www.javaworld.com/javaworld/jw-05-2006/jw-0529-maven.html

[9]Maven让事情变得简单. http://www-128.ibm.com/developerworks/cn/java/j-maven

[10]Maven文档集. http://docs.codehaus.org/display/MAVENUSER/Home

[11]有效利用Maven2的站点生成功能. http://www.matrix.org.cn/resource/article/44/44491_Maven2.html

文中例子程序下载：**http://www.blogjava.net/files/jiangshachina/maven.rar**

BOOKMARK

相关文档**(Relevant Entries)**

Java如何通过VC调用VB编写的COM

JBuilder9制作EXE文件

如何在Java中调用dll

JNI调用固有方法

Cache Your Data JDBC vs ADO NET

IntelliJ IDEA 5.0 新特性

用JavaHelp系统开发和交付更好的文档

《Eclipse集成开发工具》

WoW Powerleveling

Posted on December 15, 2006 9:22 PM | Permalink | | Comments (0) | | TrackBacks (0)

引用地址**(TRACKBACKS)**

TrackBack URL for this entry:
http://www.wujianrong.com/mt-tb.cgi/1499

发布评论**(ADD YOUR COMMENTS)**

感谢您参与评论；发表您的意见时请保持文章的相关性；不相关的或是单纯宣传的内容可能会被删掉。您的E-mail只是用来确认您发表的文章，不会出现在网页上。
Please keep your comments relevant to this blog entry. Email addresses are never displayed, but they are required to confirm your comments.

称呼(Name)： ☑记住我的个人信息(Remember)
邮箱(Email)：
网址(URL)：
评论(Add your comments)：

预览 提交