# **maven的用途**

maven是一个**项目构建和管理的工具**，提供了帮助管理 **构建**、**文档**、**报告**、**依赖**、**scms**、**发布**、**分发**的方法。可以方便的编译代码、进行依赖管理、管理二进制库等等。

maven的好处在于可以将项目过程规范化、自动化、高效化以及强大的可扩展性

利用maven自身及其插件还可以获得代码检查报告、单元测试覆盖率、实现持续集成等等。

# **maven的核心概念介绍**

## **Pom**

pom是指project object Model。pom是一个xml，在maven2里为[pom.xml](http://maven.apache.org/guides/introduction/introduction-to-the-pom.html)。是maven工作的基础，在执行task或者goal时，maven会去项目根目录下读取pom.xml获得需要的配置信息

pom文件中包含了**项目的信息和maven build项目所需的配置信息**，通常有项目信息(如版本、成员)、项目的依赖、插件和goal、build选项等等

**pom是可以继承的**，通常对于一个大型的项目或是多个module的情况，子模块的pom需要指定父模块的pom

pom文件中节点为

1. project pom文件的顶级元素
2. modelVersion 所使用的object model版本，为了确保稳定的使用，这个元素是强制性的。除非maven开发者升级模板，否则不需要修改
3. groupId 是项目创建团体或组织的唯一标志符，通常是域名倒写，如groupId  org.apache.maven.plugins就是为所有maven插件预留的
4. artifactId 是项目artifact唯一的基地址名
5. packaging artifact打包的方式，如jar、war、ear等等。默认为jar。这个不仅表示项目最终产生何种后缀的文件，也表示build过程使用什么样的<a target="\_blank" href="http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html#Built-in\_Lifecycle\_Bindings">lifecycle</a>。
6. version artifact的版本，通常能看见为类似0.0.1-SNAPSHOT，其中SNAPSHOT表示项目开发中，为开发版本
7. name 表示项目的展现名，在maven生成的文档中使用
8. url表示项目的地址，在maven生成的文档中使用
9. description 表示项目的描述，在maven生成的文档中使用
10. dependencies 表示依赖，在子节点dependencies中添加具体依赖的groupId artifactId和version
11. build 表示build配置
12. parent 表示父pom

其中**groupId:artifactId:version唯一确定了一个artifact**

## **Artifact**

这个有点不好解释，大致说就是一个项目将要产生的文件，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:artifactId:version**组成的标识符唯一识别。需要被使用(依赖)的artifact都要放在仓库(见Repository)中

## **Repositories**

**Repositories是用来存储Artifact的**。如果说我们的项目产生的Artifact是一个个小工具，那么Repositories就是一个仓库，里面有我们自己创建的工具，也可以储存别人造的工具，我们在项目中需要使用某种工具时，在pom中声明dependency，编译代码时就会根据dependency去下载工具（Artifact**）**，供自己使用。

对于自己的项目完成后可以通过mvn install命令将项目放到仓库（Repositories）中

仓库分为本地仓库和远程仓库，远程仓库是指远程服务器上用于存储Artifact的仓库，本地仓库是指本机存储Artifact的仓库，对于windows机器本地仓库地址为系统用户的.m2/repository下面。

对于需要的依赖，在pom中添加dependency即可，可以在maven的仓库中搜索：<http://mvnrepository.com/>

## **Build Lifecycle**

是指一个项目build的过程。maven的Build Lifecycle分为三种，分别为**default**（处理项目的部署）、**clean**（处理项目的清理）、**site**（处理项目的文档生成）。他们都包含不同的[lifecycle](http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html" \l "Lifecycle_Reference" \t "_blank)。

Build Lifecycle是由**phases**构成的，下面重点介绍default Build Lifecycle几个重要的phase

 validate 验证项目是否正确以及必须的信息是否可用

 compile 编译源代码

 test 测试编译后的代码，即执行单元测试代码

 package 打包编译后的代码，在target目录下生成package文件

 integration-test 处理package以便需要时可以部署到集成测试环境

 verify 检验package是否有效并且达到质量标准

 install 安装package到本地仓库，方便本地其它项目使用

 deploy 部署，拷贝最终的package到远程仓库和替他开发这或项目共享，在集成或发布环境完成

**以上的phase是有序的（注意实际两个相邻phase之间还有其他phase被省略，完整phase见**[lifecycle](http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html" \l "Lifecycle_Reference" \t "_blank)**），下面一个phase的执行必须在上一个phase完成后**

若直接以某一个phase为goal，将先执行完它之前的phase，如mvn install

将会先validate、compile、test、package、integration-test、verify最后再执行install phase

## **Goal**

**goal代表一个特定任务**

mvn package表示打包的任务，通过上面的介绍我们知道，这个任务的执行会先执行package phase之前的phase

mvn deploy表示部署的任务

mven clean install则表示先执行clean的phase（包含其他子phase），再执行install的phase。

# **maven用法**

主要讲下Archetype以及几种常用项目的创建

maven创建项目是根据Archetype（原型）创建的。下面先介绍下Archetype

## **Archetype**

**原型对于项目的作用就相当于模具对于工具的作用**，我们想做一个锤子，将铁水倒入模具成型后，稍加修改就可以了。

类似我们可以根据项目类型的需要使用不同的Archetype创建项目。通过Archetype我们可以**快速标准的创建项目**。利用Archetype创建完项目后都有[标准的文件夹目录结构](http://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html" \t "_blank)

既然Archetype相当于模具，那么当然可以自己再造模具了啊，[创建Archetype](http://maven.apache.org/guides/mini/guide-creating-archetypes.html" \t "_blank)

下面介绍利用[maven自带的集中Archetype](http://maven.apache.org/guides/introduction/introduction-to-archetypes.html" \t "_blank)创建项目。创建项目的goal为mvn archetype:generate，并且指定archetypeArtifactId，其中archetypeArtifactId见[maven自带的archetypeArtifactId](http://maven.apache.org/guides/introduction/introduction-to-archetypes.html" \l "Provided_Archetypes" \t "_blank)

## **quick start工程**

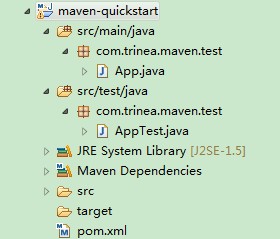
创建一个简单的quick start项目，指定**-DarchetypeArtifactId为maven-archetype-quickstart**，如下命令

mvn archetype:generate -DgroupId=com.trinea.maven.test -DartifactId=maven-quickstart -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

其中DgroupId指定groupId，DartifactId指定artifactId，DarchetypeArtifactId指定ArchetypeId，

DinteractiveMode表示是否使用交互模式，交互模式会让用户填写版本信息之类的，非交互模式采用默认值

这样我们便建好了一个简单的maven项目，目录结构如下：



编译 mvn  compile

打包 mvn package，此时target目录下会出现maven-quickstart-1.0-SNAPSHOT.jar文件，即为打包后文件

打包并安装到本地仓库mvn install，此时本机仓库会新增maven-quickstart-1.0-SNAPSHOT.jar文件。

## **web工程**

创建一个简单的web项目，只需要修 **-DarchetypeArtifactId为maven-archetype-webapp即可**，如下命令

mvn archetype:generate -DgroupId=com.trinea.maven.web.test -DartifactId=maven-web -DarchetypeArtifactId=maven-archetype-webapp -DinteractiveMode=false

其他：

**src\main\resources文件夹是用来存放资源文件的**，maven工程默认没有resources文件夹，如果我们需要用到类似log4j.properties这样的配置文件，就需要在src\main文件夹下新建resources文件夹，并将log4j.properties放入其中。

test需要用到资源文件，类似放到src\test下

# **maven常用参数和命令**

主要介绍maven常用参数和命令以及简单故障排除

## **mvn常用参数**

mvn -e 显示详细错误

mvn -U 强制更新snapshot类型的插件或依赖库（否则maven一天只会更新一次snapshot依赖）

mvn -o 运行offline模式，不联网更新依赖

mvn -N仅在当前项目模块执行命令，关闭reactor

mvn -pl module\_name在指定模块上执行命令

mvn -ff 在递归执行命令过程中，一旦发生错误就直接退出

mvn -Dxxx=yyy指定java全局属性

mvn -Pxxx引用profile xxx

## **Build Lifecycle中介绍的命令**

mvn test-compile 编译测试代码

mvn test 运行程序中的单元测试

mvn  compile 编译项目

mvn package 打包，此时target目录下会出现maven-quickstart-1.0-SNAPSHOT.jar文件，即为打包后文件

mvn install 打包并安装到本地仓库，此时本机仓库会新增maven-quickstart-1.0-SNAPSHOT.jar文件。

**每个phase都可以作为goal**，也可以联合，如之前介绍的mvn clean install

## **maven 日用三板斧**

mvn archetype:generate 创建maven项目

mvn package 打包，上面已经介绍过了

mvn package -Prelease打包，并生成部署用的包，比如deploy/\*.tgz

mvn install 打包并安装到本地库

mvn eclipse:eclipse 生成eclipse项目文件

mvn eclipse:clean 清除eclipse项目文件

mvn site 生成项目相关信息的网站

## **maven插件常用参数**

mvn -Dwtpversion=2.0 指定maven版本

mvn -Dmaven.test.skip=true 如果命令包含了test phase，则忽略单元测试

mvn -DuserProp=filePath 指定用户自定义配置文件位置

mvn -DdownloadSources=true -Declipse.addVersionToProjectName=true eclipse:eclipse 生成eclipse项目文件，尝试从仓库下载源代码，并且生成的项目包含模块版本（注意如果使用公用POM，上述的开关缺省已打开）

## **maven简单故障排除**

mvn -Dsurefire.useFile=false如果执行单元测试出错，用该命令可以在console输出失败的单元测试及相关信息

set MAVEN\_OPTS=-Xmx512m -XX:MaxPermSize=256m 调大jvm内存和持久代，maven/jvm out of memory error

mvn -X maven log level设定为debug在运行

mvndebug 运行jpda允许remote debug

mvn --help 这个就不说了。。