1、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。

2、maven 的核心概念介绍

2.1 Pom

pom 是指 project object Model。pom 是一个 xml，在 maven2 里为 pom.xml。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 文件中节点含义如下：

XHTML

```
project  pom 文件的顶级元素
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
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
artifactId  是项目 artifact 唯一的基地址名
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 lifecycle。
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
name  表示项目的展现名，在 maven 生成的文档中使用
url 表示项目的地址，在 maven 生成的文档中使用
description  表示项目的描述，在 maven 生成的文档中使用
dependencies  表示依赖，在子节点 dependencies 中添加具体依赖的 groupId  artifactId 和 version
build  表示 build 配置
parent  表示父 pom
```

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

2.2 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

2.3 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/

2.4 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。
Build Lifecycle 是由 phases 构成的，下面重点介绍 default Build Lifecycle 几个重要的 phase

```
Java
```

1 validate 验证项目是否正确以及必须的信息是否可用
2 compile 编译源代码
3 test 测试编译后的代码，即执行单元测试代码
4 package 打包编译后的代码，在 target 目录下生成 package 文件
5 integration-test 处理 package 以便需要时可以部署到集成测试环境
6 verify 检验 package 是否有效并且达到质量标准
7 install 安装 package 到本地仓库，方便本地其它项目使用
8 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)，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

2.5 Goal

goal 代表一个特定任务

```
Java
```

1 A goal represents a specific task (finer than a build phase) which contributes to the building and managing of a project.

如
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。

3、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

3.1 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

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

下面介绍利用 maven 自带的集中 Archetype 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

3.2 quick start 工程

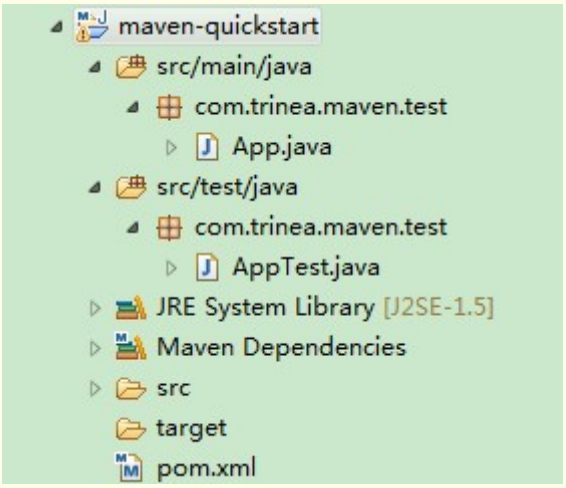
创建一个简单的 quick start 项目，指定 -DarchetypeArtifactId 为 maven-archetype-quickstart，如下命令

Xml 代码 收藏代码

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：



现在我们可以利用 2.4 的 build Lifecycle 进行一些操作，先命令行到工程根目录下

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

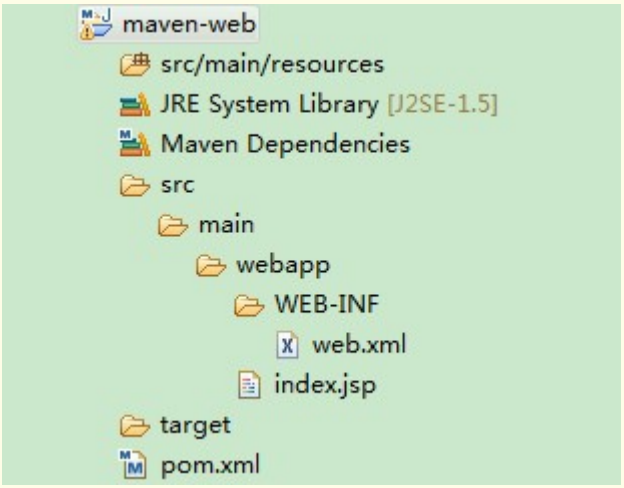
打包并安装到本地仓库 mvn install，此时本机仓库会新增 maven-quickstart-1.0-SNAPSHOT.jar 文件。

3.3 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

XHTML

```
1 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

对于 apache 的 log4j 没有 log4j.properties 文件或是目录错误，会报如下异常

XHTML

```
1 log4j:WARN No appenders could be found for logger (org.apache.commons.httpclient.HttpClient).
2 log4j:WARN Please initialize the log4j system properly.
```

4、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

4.1 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

4.2 首先是 2.4 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

4.3 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

4.4 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）

4.5 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。。

5、maven 扩展

maven 常用插件配置和使用

6、maven 配置

为了修改 maven 创建项目默认以来的 jdk 版本，看了下 maven 配置

maven2.0 默认使用 jdk1.5 导致反省、@override 等 annotation 不可用。可用两种方法修改 jdk 版本

第一种：修改项目的 pom.xml，影响单个项目，治标不治本

XHTML

```
1  <build>
2    <plugins>
3      <plugin>
4        <groupId>org.apache.maven.plugins</groupId>
5        <artifactId>maven-compiler-plugin</artifactId>
6        <configuration>
7          <source>1.6</source>
```

```
            <target>1.6</target>
            <encoding>UTF-8</encoding>
        </configuration>
    </plugin>
  </plugins>
</build>
```

pom 中增加 build 配置，指定 java 版本为 1.6

第二种：修改 maven 配置，影响 maven 建立的所有项目

到 maven 安装目录的 conf 文件夹下，修改 settings.xml 文件，如下：

Java

```
<profiles>
  <profile>
    <id>jdk-1.6</id>
    <activation>
      <activeByDefault>true</activeByDefault>
      <jdk>1.6</jdk>
    </activation>
    <properties>
      <maven.compiler.source>1.6</maven.compiler.source>
      <maven.compiler.target>1.6</maven.compiler.target>
      <maven.compiler.compilerVersion>1.6</maven.compiler.compilerVersion>
    </properties>
  </profile>
</profiles>
```

这样便能对所有默认的 maven 项目指定 jdk 为 1.6

到此为止，休息会儿

参考资料：

Maven 官方文档：http://maven.apache.org/guides/index.html

maven 安装：http://maven.apache.org/download.html