安装

1.安装jdk(java –version 查询版本)

2.下载并解压maven包 如apache-maven-3.5.0-bin.zip

3.将解压后文件夹里面的bin文件夹路径添加到环境变量path里面

4.命令行运行mvn –v 验证安装

Maven 配置文件

工程：工程目录\pom.xml activeProfiles / profiles元素

用户：C:\Users\ye\.m2\settings.xml

全局：maven安装目录\conf\ settings.xml

仓库

在 Maven 的术语中，仓库是一个位置（place），例如目录，可以存储所有的工程 jar 文件、library jar 文件、插件或任何其他的工程指定的文件。

Maven 仓库有三种类型：本地（local）、中央（central）、远程（remote）

本地仓库

默认本地仓库在C:\Users\ye\.m2\repository

在maven安装目录\conf\settings.xml配置本地仓库

<localRepository>E:/ws/maven</localRepository>

中央仓库

Maven 中央仓库是由 Maven 社区提供的仓库，其中包含了大量常用的库。

要浏览中央仓库的内容，maven 社区提供了一个 URL：http://search.maven.org/#browse。使用这个仓库，开发人员可以搜索所有可以获取的代码库。

远程仓库

如果 Maven 在中央仓库中也找不到依赖的库文件，它会停止构建过程并输出错误信息到控制台。为避免这种情况，Maven 提供了远程仓库的概念，它是开发人员自己定制仓库，包含了所需要的代码库或者其他工程中用到的 jar 文件。

Maven 依赖搜索顺序

当我们执行 Maven 构建命令时，Maven 开始按照以下顺序查找依赖的库：

步骤 1 － 在本地仓库中搜索，如果找不到，执行步骤 2，如果找到了则执行其他操作。

步骤 2 － 在中央仓库中搜索，如果找不到，并且有一个或多个远程仓库已经设置，则执行步骤 4，如果找到了则下载到本地仓库中已被将来引用。

步骤 3 － 如果远程仓库没有被设置，Maven 将简单的停滞处理并抛出错误（无法找到依赖的文件）。

步骤 4 － 在一个或多个远程仓库中搜索依赖的文件，如果找到则下载到本地仓库已被将来引用，否则 Maven 将停止处理并抛出错误（无法找到依赖的文件）。

外部依赖

外部依赖（library jar location）能够像其他依赖一样在 pom.xml 中配置。

指定 groupId 为 library 的名称。

指定 artifactId 为 library 的名称。

指定作用域（scope）为系统。

指定相对于工程位置的系统路径。

例子：

<dependency>

<groupId>ldapjdk</groupId>

<artifactId>ldapjdk</artifactId>

<scope>system</scope>

<version>1.0</version>

<systemPath>${basedir}\src\lib\ldapjdk.jar</systemPath>

</dependency>

插件与目标

一个Maven插件是一个单个或者多个目标的集合。Maven插件的例子有一些简单但核心的

插件，像Jar插件，它包含了一组创建JAR文件的目标，Compiler插件，它包含了一组编

译源代码和测试代码的目标，或者Surefire插件，它包含一组运行单元测试和生成测试

报告的目标。而其它的，更有专门的插件包括：Hibernate3插件，用来集成流行的持久

化框架Hibernate，JRuby插件，它让你能够让运行ruby称为Maven构建的一部分或者用

Ruby来编写Maven插件。Maven也提供了自定义插件的能力。一个定制的插件可以用Java

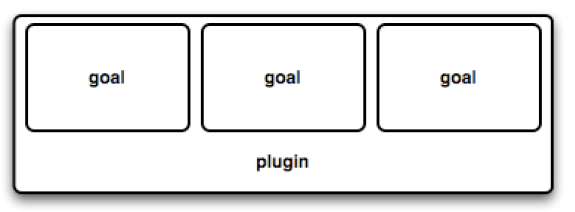
编写，或者用一些其它的语言如Ant，Groovy，beanshell和之前提到的Ruby。

一个目标是一个明确的任务，它可以作为单独的目标运行，或者作为一个大的构

建的一部分和其它目标一起运行。一个目标是Maven中的一个“工作单元(unit of

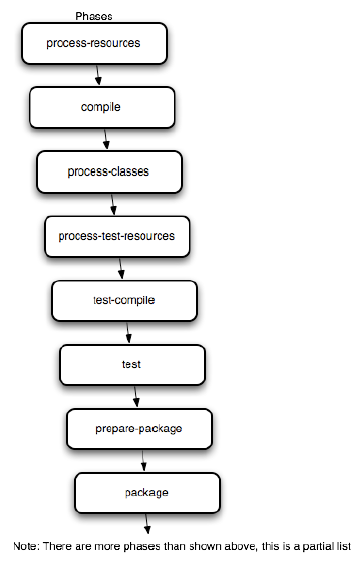
work)”。目标的例子包括Compiler插件中的compile目标，它用来编译项目中的所有

源文件，或者Surefire插件中的test目标，用来运行单元测试。



Maven生命周期 (Lifecycle)

生命周期是包含在一个项目构建中的一系列有序的阶段



插件目标可以附着在生命周期阶段上。随着Maven沿着生命周期的阶段移动，它会执

行附着在特定阶段上的目标。每个阶段可能绑定了零个或者多个目标。

Maven执行一个阶段的时候，它首先会有序的执行前面的所有阶段，到命令行指定的那个阶段为止

Maven坐标 (Coordinates)

groupId, artifactId, version和packaging 这些组合的标识符拼成了一个项目的坐标

Maven仓库的标准是按照下面的目录格式来存储构件，相对于仓库的根目录

/<groupId>/<artifactId>/<version>/<artifactId>-<version>.<packaging>

获取插件信息

mvn help:describe -Dplugin=插件前缀/ groupId:artifactId/ groupId:artifactId:version

如help插件

mvn help:describe -Dplugin=help

获取更详细信息

mvn help:describe -Dplugin=help -Dfull

获取单个目标信息

mvn help:describe -Dplugin=compiler -Dmojo=compile -Dfull

mvn命令语法

mvn 插件前缀:目标名 –D参数名=参数值

创建工程

可以将archetype-catalog.xml提前下载到C:\Users\ye\.m2 ，然后加上参数-DarchetypeCatalog=local比较快

mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-app -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false -DarchetypeCatalog=local

查看最终pom

mvn help:effective-pom