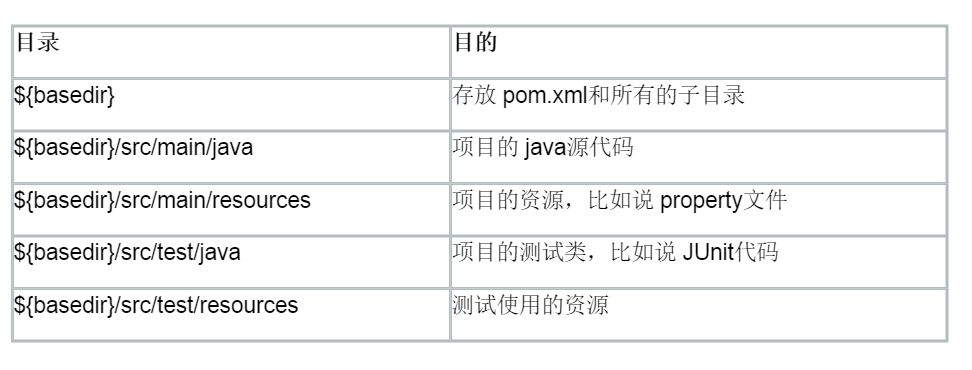
Maven 是一个项目管理和构建自动化工具。

POM 全称是 Project Object Model ，即项目对象模型。 pom.xml 是 maven 的项目描述文件。 pom.xml 文件以 xml 的形式描述项目的信息，包括项目名称、版本、项目 id 、项目的依赖关系、编译环境、持续集成、项目团队、贡献管理、生成报表等等。

POM (Project Object Model)：一个项目所有的配置都放置在 POM 文件中：定义项目的类型、名字，管理依赖关系，定制插件的行为等等

POM三个重要关系：继承、依赖、合成

结构：



${basedir}/target/classes 编译后的classes

${basedir}/target JAR 文件

配置环境变量

建立项目：（groupId、artifactId、package、version）POM的坐标

~$mvn archetype:generate -DgroupId=com.mycompany.helloworld -DartifactId=helloworld -Dpackage=com.mycompany.helloworld -Dversion=1.0-SNAPSHOT

modelVersion 描述这个 POM 文件是遵从哪个版本的项目描述符，目前只能是 4.0.0 。

groupId 针对一个项目的普遍唯一识别符。通常用一个完全正确的包的名字来与其他项目的类似名字来进行区分（比如： org.apache.maven) 。

artifactId 在给定 groupID 的 group 里面为 artifact 指定的标识符是唯一的 ， artifact 代表的是被制作或者被一个 project 应用的组件 ( 产出物 ) 。

version 当前项目产生的 artifact 的版本

scope: compile （编译范围） , provided （已提供范围） , runtime （运行时范围） , test （测试范围） , system （系统范围）

exclusions ： 如果 X 需要 A,A 包含 B 依赖，那么 X 可以声明不要 B 依赖，只要在 exclusions 中声明 exclusion.

构建项目：~$mvn package （必须进入该项目内）

mvn clean install（区别？）安装jar到maven库

运行项目：~$java -cp target/helloworld-1.0-SNAPSHOT.jar com.mycompany.helloworld.App

Pom目录结构：<http://my.oschina.net/zh119893/blog/232896>

Guide: http://maven.apache.org/guides/introduction/introduction-to-the-pom.html

常用命令：<http://www.cnblogs.com/phoebus0501/archive/2011/05/10/2042511.html>

Eclipse中的maven : http://www.cnblogs.com/xing901022/p/4170248.html

Tree 查看结构

创建webapp：

C:\Users\Lee\system-parent>mvn archetype:generate -DgroupId=com.mycompany.hellow

orld -DartifactId=system-web -DarchetypeArtifactId=maven-archetype-webapp

启动插件服务器：mvn jetty:run

项目构建：http://www.cnblogs.com/xdp-gacl/p/4242221.html

核心概念：

（http://www.oracle.com/technetwork/cn/community/java/apache-maven-getting-started-2-405568-zhs.html）

### Maven 插件：

我们用了 mvn archetype:generate 命令来生成一个项目。那么这里的 archetype:generate 是什么意思呢？archetype 是一个插件的名字，generate是目标(goal)的名字。这个命令的意思是告诉 maven 执行 archetype 插件的 generate 目标。

### Maven生命周期

maven 能支持不同的生命周期，但是最常用的是默认的Maven生命周期 (default Maven lifecycle )。如果你没有对它进行任何的插件配置或者定制的话，那么上面的命令 mvn package 会依次执行默认生命周期中直到包括 package 阶段前的所有阶段的插件目标：

process-resources 阶段：resources:resources

compile 阶段：compiler:compile

process-classes 阶段：(默认无目标)

process-test-resources 阶段：resources:testResources

test-compile 阶段：compiler:testCompile

test 阶段：surefire:test

prepare-package 阶段：(默认无目标)

package 阶段：jar:jar

### Maven依赖管理

实际开发中我们会有复杂得多的依赖关系，因为被依赖的 jar 文件会有自己的依赖关系。那么我们是不是需要把那些间接依赖的 jar 文件也都定义在POM中呢？答案是不需要，因为 maven 提供了传递依赖的特性。

所谓传递依赖是指 maven 会检查被依赖的 jar 文件，把它的依赖关系纳入最终解决的依赖关系链中。

在 POM 的 dependencies 部分中，scope 决定了依赖关系的适用范围。

我们的例子中 junit 的 scope 是 test，那么它只会在执行 compiler:testCompile and surefire:test 目标的时候才会被加到 classpath 中，在执行 compiler:compile 目标时是拿不到 junit 的。

我们还可以指定 scope 为 provided，意思是 JDK 或者容器会提供所需的jar文件。比如说在做web应用开发的时候，我们在编译的时候需要 servlet API jar 文件，但是在打包的时候不需要把这个 jar 文件打在 WAR 中，因为servlet容器或者应用服务器会提供的。

scope 的默认值是 compile，即任何时候都会被包含在 classpath 中，在打包的时候也会被包括进去。

### Maven 库

但是不是所有的 jar 文件都是可以从默认的远程库下载的，比如说我们自己开发的项目。这个时候，有两个选择：要么在公司内部设置定制库，要么手动下载和安装所需的jar文件到本地库。

本地库是指 maven 下载了插件或者 jar 文件后存放在本地机器上的拷贝。在 Linux 上，它的位置在 ~/.m2/repository，在 Windows XP 上，在 C:\Documents and Settings\username\.m2\repository ，在 Windows 7 上，在 C:\Users\username\.m2\repository。当 maven 查找需要的 jar 文件时，它会先在本地库中寻找，只有在找不到的情况下，才会去远程库中找。

运行下面的命令能把我们的 helloworld 项目安装到本地库：

$mvn install

一旦一个项目被安装到了本地库后，你别的项目就可以通过 maven 坐标和这个项目建立依赖关系。比如如果我现在有一个新项目需要用到 helloworld，那么在运行了上面的 mvn install 命令后，我就可以如下所示来建立依赖关系：

Xml 代码

<dependency>

<groupId>com.mycompany.helloworld</groupId>

<artifactId>helloworld</artifactId>

<version>1.0-SNAPSHOT</version>

</dependency>