**Maven的作用是什么**

Maven主要做了两件事：

* 统一开发规范与工具
* 统一管理jar包

**pom.xml**

打开pom.xml，最基础的是这样的：

[复制代码](javascript:void(0);)

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.xrq.withmaven</groupId>

<artifactId>withmaven</artifactId>

<version>0.0.1-SNAPSHOT</version>

<build/>

</project>

[复制代码](javascript:void(0);)

**1、modelVersion**

　　指定了当前Maven模型的版本号，对于Maven2和Maven3来说，它只能是4.0.0

**2、groupId**

　　顾名思义，这个应该是公司名或是组织名。一般来说groupId是由三个部分组成，每个部分之间以"."分隔，第一部分是项目用途，比如用于商业的就是"com"，用于非营利性组织的就　　是"org"；第二部分是公司名，比如"tengxun"、"baidu"、"alibaba"；第三部分是你的项目名

**3、artifactId**

　　可以认为是Maven构建的项目名，比如你的项目中有子项目，就可以使用"项目名-子项目名"的命名方式

**4、version**

　　版本号，SNAPSHOT意为快照，说明该项目还在开发中，是不稳定的版本。在Maven中很重要的一点是，groupId、artifactId、version三个元素生成了一个Maven项目的基本坐标，这非常重要，我在使用和研究Maven的时候多次感受到了这点。

在上面的这些元素之外，还有一些元素，同样罗列一下：

**1、packing**

　　项目打包的类型，可以使jar、war、rar、ear、pom，默认是jar

**2、dependencies和dependency**

　　前者包含后者。前面说了，Maven的一个重要作用就是统一管理jar包，为了一个项目可以build或运行，项目中不可避免的，会依赖很多其他的jar包，在Maven中，这些依赖就被称为dependency。

**本地仓库**和**远程仓库**：

官方下载的本地仓库的配置在"%MAVEN\_HOME%\conf\settings.xml"里面，找一下"localRepository"就可以了；MyEclipse默认的本地仓库的地址在"{user.home}/.m2/repository"路径下，同样找一下"localRepository"就可以找到MyEclipse默认的本地仓库了。

　　本地仓库和远程仓库是这样的，Maven工程首先会从本地仓库中获取jar包，当无法获取指定jar包时，本地仓库会从远程仓库（中央仓库）中下载jar包，并放入本地仓库以备将来使用。

　　举个例子，比方说我的项目中用到了MyBatis，那么可以这么配置：

[复制代码](javascript:void(0);)

<dependencies>

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis</artifactId>

<version>3.2.5</version>

</dependency>

</dependencies>

[复制代码](javascript:void(0);)

　　之前有说过groupId、artifactId、version唯一标识一个Maven项目，有了这三个元素，我们就可以去远程仓库下载MyBatis3.2.5.jar到本地仓库了。回想我们之前的做法，如果要MyBatis的jar包，发现没有，然后去网上下载一个，需要另外的jar包，然后去网上下载一个，但是有了Maven，就方便多了，只需要配置jar包对应的dependency依赖，Maven会自动帮助我们去远程仓库中下载jar包到本地仓库中。

**3、properties**

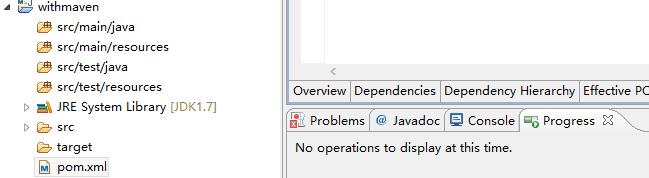
　　properties是用来定义一些配置属性的，例如project.build.sourceEncoding（项目构建源码编码方式），可以设置为UTF-8，防止中文乱码，也可定义相关构建版本号，便于日后统一升级。

**4、build**

　　build表示与构建相关的配置，比如build下有finalName，表示的就是最终构建之后的名称。

**Maven工程目录结构**

回到上面那张图：



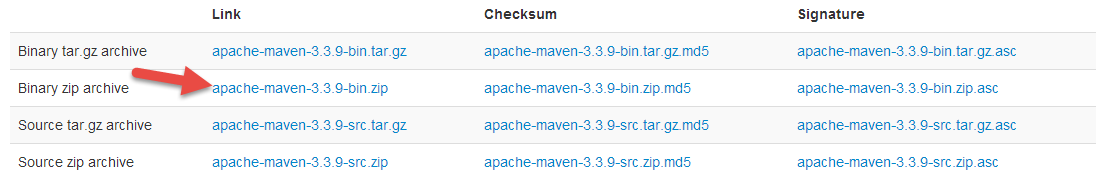
接着解释一下Maven的目录结构：

* main目录下是项目的主要代码，test目录下存放测试相关的代码
* 编译输出后的代码会放在target目录下
* src/main/java下存放Java代码，src/main/resources下存放配置文件
* 这里没有webapp，Web项目会有webapp目录，webapp下存放Web应用相关代码
* pom.xml是Maven项目的配置文件

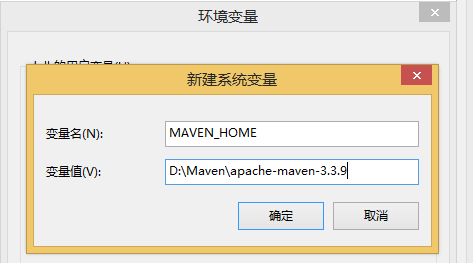
**Maven环境配置**

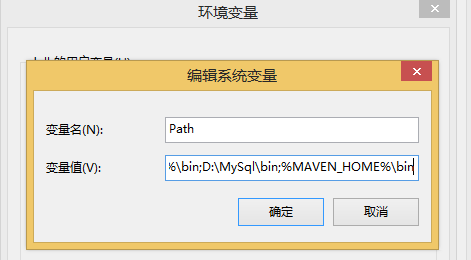
常用的开发工具Idea、MyEclipse里面都已经集成了Maven了，不过最好是从官网下一个配置到自己电脑里，开发工具里的可能有少许的Bug。

首先去Maven官网，下载Maven的包，地址为http://maven.apache.org/download.cgi，找到下面的部分，点击就可以下载了：

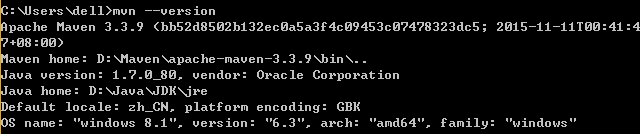


下载完解压，然后配置一下环境变量，和JDK的环境变量配置类似：





这样配置完就可以了，验证一下，windows+r打开命令窗口，输入"mvn --version"，如果有下面的内容这表示Maven配置OK：



**maven生命周期**

Maven生命周期可以从两方面来理解

第一，顾名思义，运行Maven的每个步骤都由它来定义的，这种预定义的默认行为使得我们使用Maven变得简单。

第二，这个模型是一种标准，在不同的项目中，使用Maven的接口是一样的，这样就不用去仔细理解每个项目的构建了，一般情况下，mvn clean install 这样的命令是通用的。

Maven有三套相互独立的生命周期，而且“相互独立”这三套生命周期分别是：

Clean Lifecycle 在进行真正的构建之前进行一些清理工作。

Default Lifecycle 构建的核心部分，编译，测试，打包，部署等等。

Site Lifecycle 生成项目报告，站点，发布站点。

它们是相互独立的，可以仅仅调用clean来清理工作目录，仅仅调用site来生成站点。当然也可以直接运行 mvn clean install site 运行所有这三套生命周期。

Clean和Site相对比较简单，每套生命周期都由一组阶段(Phase)组成，平时在命令行输入的命令总会对应于一个特定的阶段。比如，运行mvn clean ，这个的clean是Clean生命周期的一个阶段。要知道有Clean生命周期，也有clean阶段。Clean生命周期一共包含了三个阶段：

pre-clean  执行一些需要在clean之前完成的工作

clean  移除所有上一次构建生成的文件

post-clean  执行一些需要在clean之后立刻完成的工作

mvn clean 中的clean就是上面的clean，**在一个生命周期中，运行某个阶段的时候，它之前的所有阶段都会被运行**，也就是说，mvn clean 等同于 mvn pre-clean clean ，如果运行 mvn post-clean ，那么 pre-clean，clean 都会被运行。。

Site生命周期的各个阶段：

pre-site     执行一些需要在生成站点文档之前完成的工作

site    生成项目的站点文档

post-site     执行一些需要在生成站点文档之后完成的工作，并且为部署做准备

site-deploy     将生成的站点文档部署到特定的服务器上

经常用到的是site阶段和site-deploy阶段，用以生成和发布Maven站点。

Maven的最重要的Default生命周期，绝大部分工作都发生在这个生命周期中，重要和常用的阶段：

validate

generate-sources

process-sources

generate-resources

process-resources     复制并处理资源文件，至目标目录，准备打包。

compile     编译项目的源代码。

process-classes

generate-test-sources

process-test-sources

generate-test-resources

process-test-resources     复制并处理资源文件，至目标测试目录。

test-compile     编译测试源代码。

process-test-classes

test     使用合适的单元测试框架运行测试。这些测试代码不会被打包或部署。

prepare-package

package     接受编译好的代码，打包成可发布的格式，如 JAR 。

pre-integration-test

integration-test

post-integration-test

verify

install     将包安装至本地仓库，以让其它项目依赖。

deploy     将最终的包复制到远程的仓库，以让其它开发人员与项目共享。

**另外，运行任何一个阶段的时候，它前面的所有阶段都会被运行，这也就是为什么我们运行mvn install 的时候，代码会被编译，测试，打包。此外，Maven的插件机制是完全依赖Maven的生命周期的**