Maven简单介绍

博客介绍：<https://www.cnblogs.com/jingmoxukong/p/5591368.html>

# 利用Maven获取dependency信息与下载jar包

Maven 仓库：<http://mvnrepository.com/> 查询dependency信息

jar包搜索中心：<http://search.maven.org/>

# Maven介绍

## Maven是什么？

Maven 是一个**项目构建和项目管理工具**，**它负责管理项目开发过程中的几乎所有的东西**。它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。

## Maven的功能

**jar包、api文档、源码从哪里来？**

**如何关联源码？(每换一次工程都需要手动关联一次)**

**如何管理这些资源？**

**版本**：Maven有自己的版本定义和规则；

**构建**：Maven支持许多种的**应用程序类型**，对于每一种支持的应用程序类型都定义好了一组构建规则和工具集。

**输出物管理**：maven可以管理项目构建的产物，并将其加入到用户库中。这个功能可以用于项目组和其他部门之间的交付行为。

**依赖关系**：maven对依赖关系的特性进行细致的分析和划分，避免开发过程中的依赖混乱和相互污染行为。

**文档和构建结果**：maven的site命令支持各种文档信息的发布，包括构建过程的各种输出，javadoc，产品文档等。

**项目关系：一个大型的项目通常有几个小项目或者模块组成，用maven可以很方便地管理。**

**移植性管理**：maven可以针对不同的开发场景，**输出不同种类的输出结果**。

# Maven的生命周期

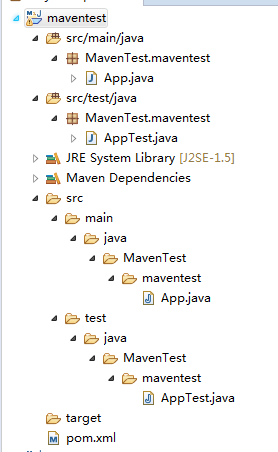
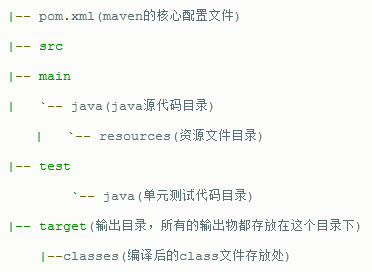
maven把项目的构建划分为不同的生命周期(lifecycle)。粗略一点的话，它这个过程(phase)包括：**编译、测试、打包、集成测试、验证、部署**。maven中所有的执行动作(goal)都需要指明自己在这个过程中的执行位置，然后maven执行的时候，就依照过程的发展依次调用这些goal进行各种处理。

这个也是**maven的一个基本调度机制**。一般来说，位置稍后的过程都会依赖于之前的过程。当然，maven同样提供了配置文件，可以依照用户要求，跳过某些阶段。

# Maven的标准工程结构

Maven的标准配置文件为：**pom.xml**

**Project Object Model (POM) ：工程对象模型**。



# 约定优于配置

所谓的"**约定优于配置**"，在maven中并不是完全不可以修改的，他们只是一些配置的默认值而已。**但是除非必要，并不需要去修改那些约定内容**。

每一个阶段的任务都知道怎么正确完成自己的工作，**比如compile任务就知道从src/main/java下编译所有的java文件，并把它的输出class文件存放到target/classes中。**

对maven来说，采用"约定优于配置"的策略可以减少修改配置的工作量，也可以降低学习成本，更重要的是，给项目引入了统一的规范。

# Maven的版本规范

maven使用如下**几个要素**来唯一定位某一个输出物：

## groudId：团体、组织的标识符。

团体标识的约定是，它以创建这个项目的组织名称的**逆向域名**(reverse domain name)开头。一般对应着JAVA的包的结构。例如org.apache

## artifactId ：单独项目的唯一标识符。

比如我们的tomcat, commons等。不要在artifactId中包含点号(.)。

## version ：一个项目的特定版本。

## packaging ：项目的类型，默认是jar，描述了项目打包后的输出。

**类型为jar的项目产生一个JAR文件**，**类型为war的项目产生一个web应用**。

如<packaging>jar</packaging>。

## Maven的版本规范

maven有自己的**版本规范**，一般是如下定义 <major version>.<minor version>.<incremental version>-<qualifier> ，比如1.2.3-beta-01。要说明的是，maven自己判断版本的算法是**major,minor,incremental部分用数字比较**，qualifier部分用字符串比较，所以要小心 alpha-2和alpha-15的比较关系，最好用 alpha-02的格式。

maven在版本管理时候可以使用几个特殊的字符串 **SNAPSHOT，LATEST，RELEASE**。比如"1.0-SNAPSHOT"。各个部分的含义和处理逻辑如下说明：

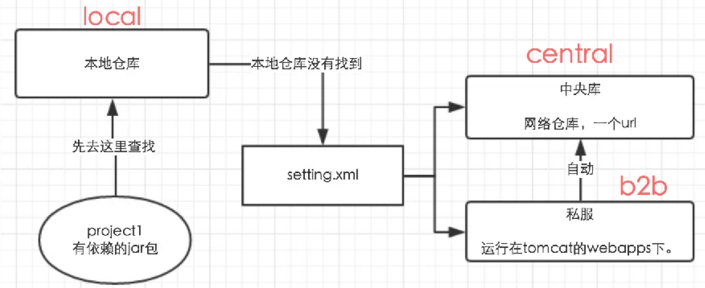
**SNAPSHOT**：这个版本一般用于**开发过程**中，表示**不稳定的版本**。

**LATEST**：指某个特定构件的最新发布，这个发布可能是一个发布版，也可能是一个snapshot版，具体看哪个时间最后。

**RELEASE**：指最后一个发布版。

# Maven的下载依赖的原理

Maven项目首先从settings.xml文件指定的localRepository本地仓库下载依赖，若本地仓库没有，则会从settings.xml文件指定的**私服仓库**寻找，最后从Maven的中央仓库下载。Maven的中央仓库URL为：<http://mvnrepository.com/>。



# Maven的安装

官网：<http://maven.apache.org/index.html>

Maven下载地址：<http://maven.apache.org/download.cgi>

安装Maven十分简单，下载“**Binary zip archive**”压缩包，解压即可。(需要配置环境变量)

配置好之后，就可以通过mvn命令进行操作。不过一般在集成开发环境IDE中添加Maven插件进行开发。



## 配置Maven的环境变量：

**注意：在安装Maven之前，必须先确保机器中已经安装了JDK，并配置了JDK的环境变量。**

配置Maven的环境变量和JDK的环境变量类似：

### 解压“apache-maven-3.5.3-bin.zip”压缩包，可以修改目录名称。

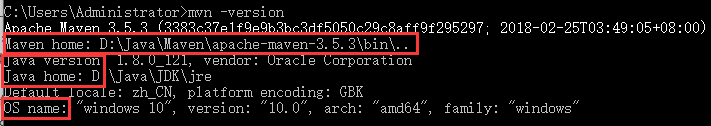
### 在环境变量中添加MAVEN\_HOME，如



### 在Path环境变量的变量值末尾添加：%MAVEN\_HOME%\bin

### 在cmd中验证：mvn –version，若出现maven的版本信息，则配置成功。

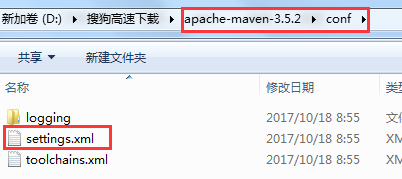
同时显示，Java version、Java home、OS name等信息。



## 本地仓库位置（localRepository）的配置

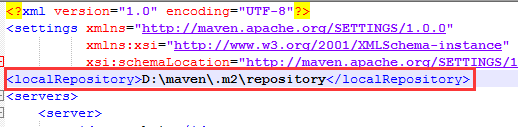
从**中央仓库**下载的jar包，都会统一存放到**本地仓库**中。我们**需要配置本地仓库的位置**。

打开maven安装目录的**conf文件夹下**的**setting.xml**文件。

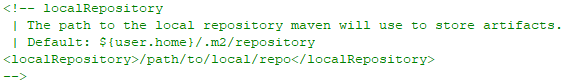


可以参照下图配置本地仓储位置。

**<localRepository>D:\maven\.m2\repository</localRepository>**



默认的localRepository位置：**${user.home}/.m2/repository**



## 在Eclipse中安装Maven插件

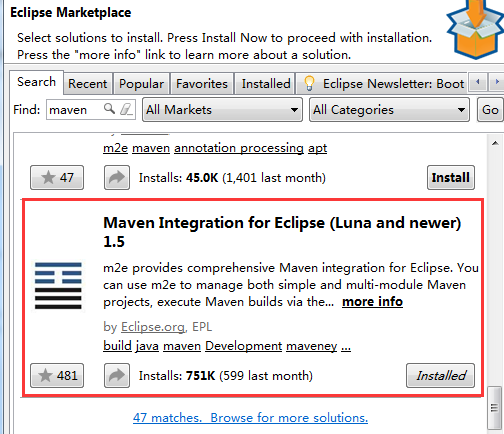
需要两步：1、**安装maven插件**;2、加载本地安装maven的**setting.xml**文件。

### 首先安装maven插件

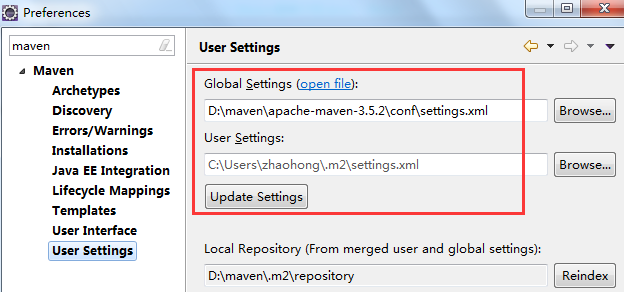
在Eclipse中**创建Maven工程**，需要安装Maven插件。

一般较新版本的Eclipse都会带有Maven插件，如果你的Eclipse中已经有Maven插件，可以跳过这一步骤。

点击**Help -> Eclipse Marketplace**，搜索maven关键字，选择安装红框对应的Maven插件。



### 添加Maven的settings.xml配置文件

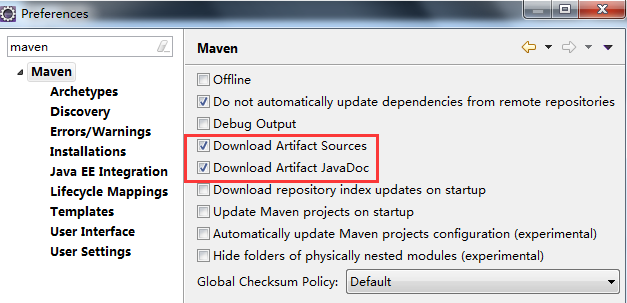


OK,这样就可以在Eclipse中直接创建Maven工程了。

说明：**Eclipse中安装Maven插件后，需要一个settings.xml配置文件，这个配置文件放在任意位置都可以；在这个配置文件中需要指定本地仓库localRepository的位置**。下载的maven压缩包，我们只用到了conf目录下的settings.xml文件，其他文件都没有用到。

## Eclipse中配置自动下载源码及文档

**Window—Preferences--Maven**



## 对于maven-web工程，不利用tomcat，利用插件模拟运行

见后面Maven工程中添加插件。

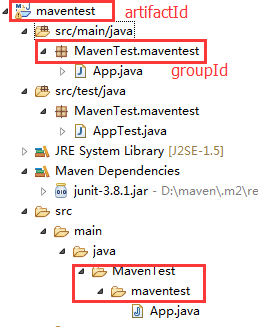
# 创建Maven工程

## 创建maven工程

**File -> New -> Maven Project -> Next**，在接下来的窗口中会看到一大堆的项目模板，选择合适的模板。

**groupId**是**项目组织唯一的标识符**，实际对应JAVA的包的结构，是**main目录里java的目录结构**。

**artifactId**就是**项目的唯一的标识符**，实际对应项目的名称，就是**项目根目录的名称**。

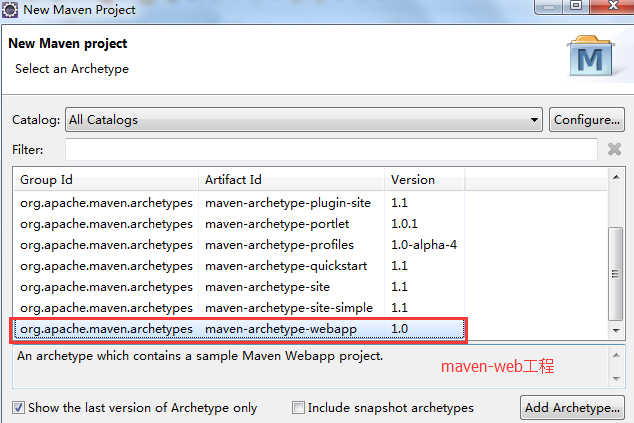


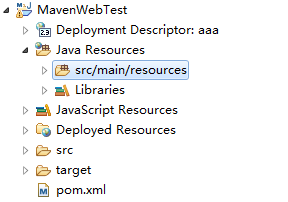
一般，普通的java-Maven工程，需要有三个文件（或目录）：

**根目录下pom.xml; src/main/java目录；src/main/resources目录。**

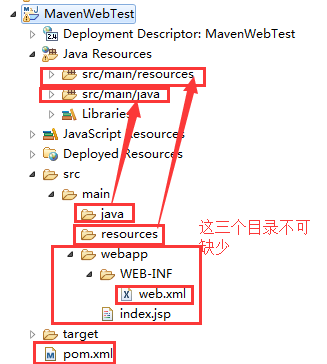
**上面自动创建的没有resources目录，需要手动创建即可，resources目录下存放配置文件。**

## 创建maven-web工程





缺失src/main/java目录，需要在main中创建java文件夹即可。



注意：默认创建的Maven-web工程，jsp文件可能无法编译，报错，原因是没有Servlet类，**解决方法是在pom文件下添加javax.servlet » javax.servlet-api的依赖就可以**。web.xml文件中的版本过低，可以通过单独创建一个动态web工程，指定最高版本，且指定生成**web.xml**文件，复制到maven-web工程下就可以了。

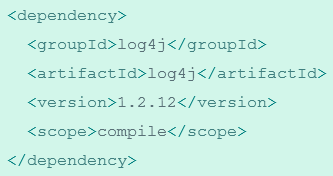
## Maven工程下有用的文件就两个：src文件夹和pom.xml

**其他文件都是Eclipse自动生成的，对项目没有作用，上传服务器的时候，只需要保留这两个文件就可以。**拷贝maven工程的时候，也要把其他文件删掉，因为不同主机的eclipse路径等都不一样，所以那些文件也就不能共享。



# 使用Maven如何添加外部依赖jar包

在Maven工程中添加**依赖jar包**，很简单，只要在**POM文件**中引入对应的**<dependency>标签**即可。



<dependency>标签最常用的四个属性标签：

### groupId：项目组织唯一的标识符，实际对应JAVA的包的结构。

### artifactId：项目唯一的标识符，实际对应项目的名称，就是项目根目录的名称。

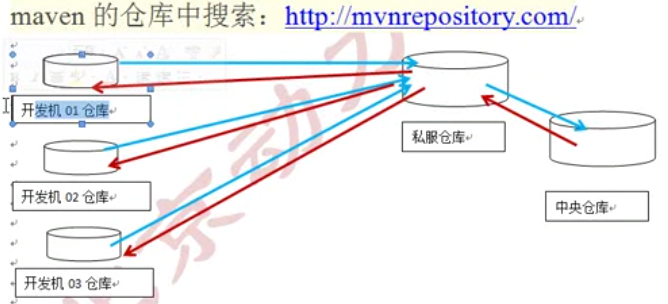
### version：jar包的版本号。可以直接填版本数字，也可以在properties标签中设置属性值。

### scope：jar包的作用范围。可以填写compile、runtime、test、system和provided。用来在编译、测试等场景下选择对应的classpath。

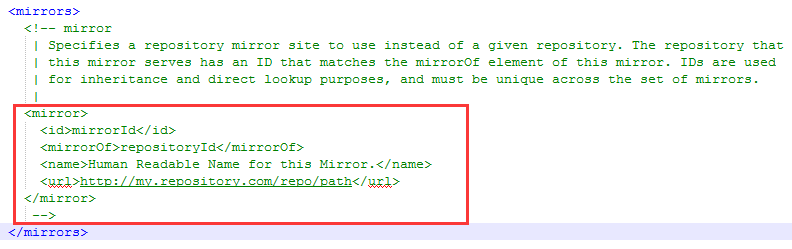
# 从Maven仓库中寻找Jar包

Maven的仓库地址为：<http://mvnrepository.com/>

可以在该仓库站点搜寻你想要的jar包版本。



在**settings.xml**中的**mirrors**标签可以指定镜像位置，建立私服仓库。

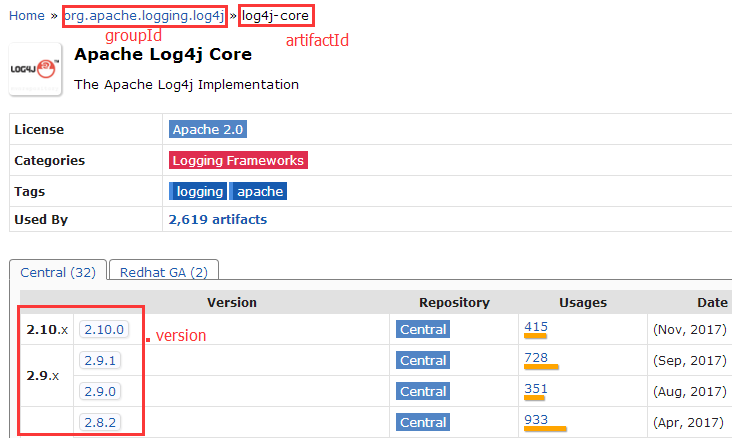


**私服仓库就是公司自己建立的一个仓库，节约了从中央仓库下载的时间。**

例如，想要使用log4j，可以找到需要的版本号，然后**拷贝对应的maven标签信息**，将其添加到**pom .xml**文件中。



版本号：点进去，就可以查看版本号信息。



想要哪一个version，就直接点进去，然后就可以复制dependency信息到pom文件就可以。



最后，添加上**scope配置**就可以了。

# Maven工程中添加插件

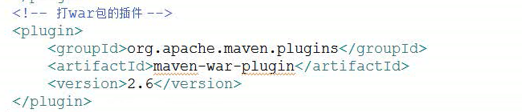
在pom.xml文件的<build></build>内部添加**<plugins></plugins>**，内部就可以添加所需要的插件。

## 编译插件

这个Java编译插件取代Eclipse内部的编译工具。



## 打war包插件与打jar包插件

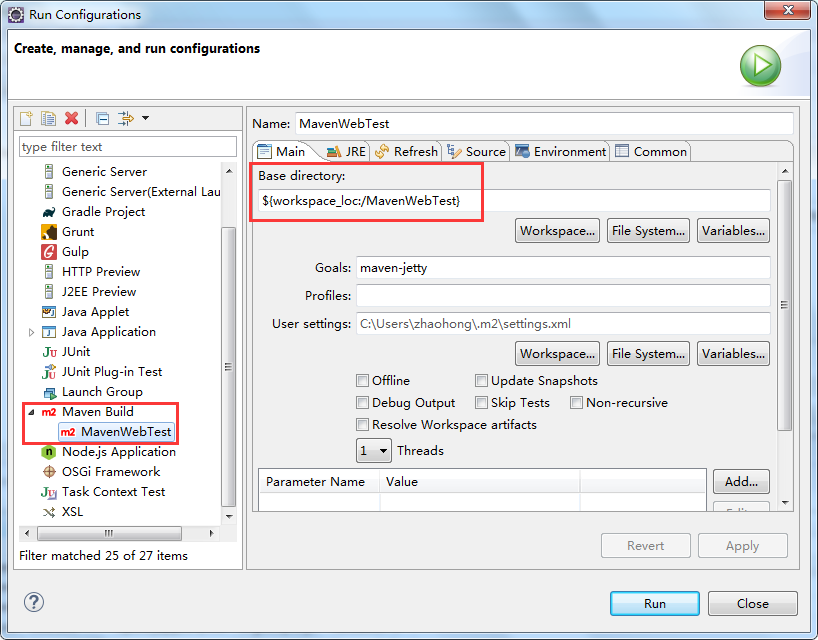




## 模拟web应用插件：jetty

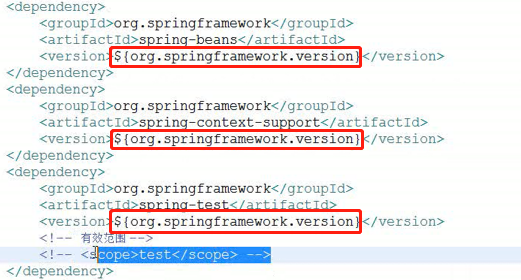


**Run As--RunConfigurations**：



# 在pom.xml文件中配置依赖

Spring需要引入很多jar包，且版本需要一致，为了便于管理和升级，version值采用变量值传入。这样，以后需要升级spring，直接修改这个变量值就可以。这个变量值在properties中指定。





# 测试依赖：junit

