# maven的安装

1.1．maven安装包下载：<http://maven.apache.org/download.cgi>

1.2．maven的环境配置：

A.新建环境变量：M2\_HOME值：apache-maven-3.5.3.zip解压后所在的路径

B.添加到环境变量 – PATH : %M2\_HOME%\bin

1.3.验证：

# 二.maven的主要思想（为什么使用它）

2.1 生产环生境下开发不再是一个项目一个工程，而是每一个模块创建一个工程，而多个模块整合在一起就需要 使用到像 Maven 这样的构建工具。

2.2 为什么要使用 Maven？它能帮助我们解决什么问题？

A.添加第三方 jar 包 在今天的 JavaEE 开发领域，有大量的第三方框架和工具可以供我们使用。要使用这些 jar 包最简单 的方法就是复制粘贴到 WEB-INF/lib 目录下。但是这会导致每次创建一个新的工程就需要将 jar 包重复复 制到 lib 目录下，从而造成工作区中存在大量重复的文件，让我们的工程显得很臃肿。 而使用 Maven 后每个 jar 包本身只在本地仓库中保存一份，需要 jar 包的工程只需要以坐标的方式 简单的引用一下就可以了。不仅极大的节约了存储空间，让项目更轻巧，更避免了重复文件太多而造成 的混乱。

B.包之间的依赖关系：jar 包往往不是孤立存在的，很多 jar 包都需要在其他 jar 包的支持下才能够正常工作，我们称之为 jar 包之间的依赖关系。

而引入 Maven 后，Maven 就可以替我们自动的将当前 jar 包所依赖的其他所有 jar 包全部导入进来， 无需人工参与，节约了我们大量的时间和精力。

C.获取第三方 jar 包 JavaEE 开发中需要使用到的 jar 包种类繁多，几乎每个 jar 包在其本身的官网上的获取方式都不尽相 同。为了查找一个 jar 包找遍互联网，身心俱疲，没有经历过的人或许体会不到这种折磨。不仅如此， 费劲心血找的 jar 包里有的时候并没有你需要的那个类，又或者又同名的类没有你要的方法——以不规 范的方式获取的 jar 包也往往是不规范的。 使用 Maven 我们可以享受到一个完全统一规范的 jar 包管理体系。你只需要在你的项目中以坐标的 方式依赖一个 jar 包，Maven 就会自动从中央仓库进行下载，并同时下载这个 jar 包所依赖的其他 jar 包 “玩转”Java 系列

# 三 maven是什么

3.1. maven是一款服务于java平台的自动化构建工具。

3.2.构建：概念，以“java源文件”，“框架配置文件” ，“jsp”，“h5”等资源为原材料，去“生成”一个可以运行的工程。

A. 构建过程的几个主要环节

(1)清理：删除以前的编译结果，为重新编译做好准备

(2)编译：将 Java 源程序编译为字节码文件。

(3)测试：针对项目中的关键点进行测试，确保项目在迭代开发过程中关键点的正确性。

(4)报告：在每一次测试后以标准的格式记录和展示测试结果。

(5)打包：将一个包含诸多文件的工程封装为一个压缩文件用于安装或部署。Java 工程对应 jar 包，Web 工程对应 war 包。

(6)安装：在 Maven 环境下特指将打包的结果——jar 包或 war 包安装到本地仓库中。

(7)部署：将打包的结果部署到远程仓库或将 war 包部署到服务器上运行。

# 四．安装maven核心程序

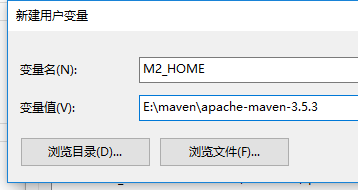
4.1.检查java\_home环境变量



4.2解压核心压缩包apache-maven-3.5.3.zip，放在非中文无空格的路径下

4.3配置maven的环境变量

(1).MAVEN\_HOME或MAVEN



(3).PATH配置E:\maven\apache-maven-3.5.3\bin

4.4验证：运行mvn-v命令去查看maven的版本



# 五 maven的核心概念

5.1约定的目录结构

5.2Pom

5.3坐标

5.4依赖

5.5仓库

5.6生命周期/插件/目标

5.7继承

5.8聚合

# 六 第一个Maven工程

6.1创建约定的目录结构

A.根目录：工程名

B.src目录：源码

C.pom.xml文件：maven工程的核心配置文件

D.main目录：存放主程序

E.test目录：存放测试程序

F.java目录：存放java源文件

G.resource目录：存放框架配置或其他工具的配置文件

6.2为什么要遵守规定的目录结构

A.Maven要负责我们项目的自动化构建，以编译为例，Maven必须找到java源文件放在那里

B.如果我们自定义的东西想要框架或工具知道，有两种方法

(1).以配置的方式告诉框架。

(2).遵守框架的约定

(3).约定>配置>编码

# 七 常见的Maven命令

7.1 注意：执行与构建工程相关的命令，必须进入pom.xml所在的目录

7.2常见的命令

A.mvn clean :清理

B.mvn compile 编译

C.mvn test-compile ：编译测试程序

D.mvn test：执行测试

E.mvn package ：执行打包

F.mvn install：安装

G.mvn insite :生成站点

# 八 关于联网的问题

8.1Maven的核心程序中金进定义了抽象的生命周期，但是具体的工作必须有特定的插件来完成，而插件本身并不包含在Maven的核心程序中

8.2当我们执行的Maven命令用到某些插件时，Maven核心程序首先会在本地仓库中找

8.3本地仓库的默认位置：[系统中当前用户的加目录]\.m2\repository C:\Users\汪进银\.m2\repository

8.4如果在本地仓库找不到，那么他会自动连接外网在中央仓库下载

8.5如果没网则会构建失败

8.6修改默认的本地仓库位置，可以让Maven核心程序到我们事先准备好的目录下查找插件

A..找到 Maven解压目录\conf\settings.xml

B.在setting.xml文件中找到localRepository标签

C.将

< >/path/to/local/repo</ localRepository>从注释中取出

D.将标签体内容修改为已经准备Maven仓库目录：

# 九pom

9.1 含义：Project Object Model项目对象模型

Dom Document Object Model 文档对象模型

9.2pom.xml对于Maven工程是核心配置文件，与构建过程相关的一切设置都在这个文件中配置，相当于web.xml

# 十 坐标

10.1数学中的坐标

在平面上，使用 x, y两个向量可以唯一定位平面中任何一个点

在空间中，使用x ,y ,z三个向量可以唯一确定空间中任一个点

10.2Maven的坐标

使用下面三个向量在仓库中唯一定位一个Maven工程

A.groupId：公司或 组织域名倒序+项目名

<groupid>……….</ groupid>

B.artifactId:模块名

<artifactid >Hello</artifactid>

C.version：版本

<version>1.0.0</version>

10.3Maven工程的坐标与仓库中路径的对应关系

# 十一 仓库

11.1仓库的分类：

A.本地仓库：当前电脑上部署的仓库目录，为当前电脑上所有Maven工程服务

B.远程仓库：(1).私服：搭建在局域网中，为局域网范围内的所有Maven工程服务

(2)中央仓库：架设在 Internet 上，为全世界所有 Maven 工程服务。

(3)中央仓库镜像：架设在各个大洲，为中央仓库分担流量。减轻中央仓库的压力，同时更快的响应用户请求。

11.2仓库总保存的内容：Maven工程

A.Maven 的插件

B.我们自己开发的项目的模块

C.第三方框架或工具的 jar 包

# 十二 依赖

12.1 Maven解析依赖信息时会先到本地仓库去查找被依赖的jar包

对于我们自己开发的Maven工程，使用Install命令安装后就可以进去仓库

12.2依赖的范围

A. compile:

(1).对主程序是否有效： 有效

(2)．对测试程序是否有效：有效

(3).是否参与打包：参与

B.test

(1).对主程序是否有效： 无效

(2)．对测试程序是否有效：有效

(3).是否参与打包：不参与

典型的例子 junit – api

C.provided

(1).对主程序是否有效： 有效

(2)．对测试程序是否有效：有效

(3).是否参与打包：不参与

(4).是否参与部署：不参与

典型的例子 servlet – api

开发

Tomcat

运行

部署

Compile依赖范围

Maven工程（WEB工程）

# 十三 生命周期

13.1什么是 Maven 的生命周期？

A. Maven 生命周期定义了各个构建环节的执行顺序，有了这个清单，Maven 就可以自动化的执行构建命 令了。

B.Maven的核心程序中定义了抽象的生命周期，生命周期各个阶段的具体执行顺序室友插件来完成的

C.Maven核心程序为了更好的实现自动化构建，按照这一特点执行生命周期中的各个节点： 不管执行那个阶段都是从最开始执行的

D.插件和目标：

|  |  |  |
| --- | --- | --- |
| 生命周期阶段 | 插件目标 | 插件 |
| Compile | Compile | Maven-compile-plugin |
| Test-compile | TestCompile | Maven-compile-plugin |

13.2三个独立生命周期

A．Clean Lifecycle 在进行真正的构建之前进行一些清理工作。

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

C．Site Lifecycle 生成项目报告，站点，发布站点。

# 十四 在eclipse中使用Maven

14.1Maven插件

14.2Maven插件的设置

A.installations:指定Maven核心程序的设置，不建议使用插件自带的Maven程序，而建议使用我们解压的那个

B．user setting：指定conf/setting.xml 的位置。进而获取本地仓库的位置

14.3基本操作

A.创建Maven 版的java工程

B.创建Maven版的web工程

C.如何执行maven命令

14.4设置jdk的版本：在setting.xml

[1]打开settings.xml文件找到profiles标签

<profile>

<id>jdk-1.8</id>

<activation>

<activeByDefault>true</activeByDefault>

<jdk>1.8</jdk>

</activation>

<properties>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

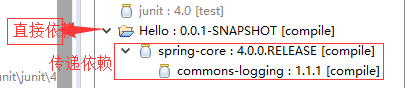
<maven.compiler.compilerVersion>1.8</maven.compiler.compilerVersion>

</properties>

</profile>

# 十五.依赖（高级）

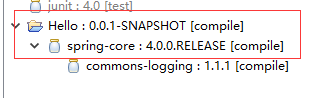
1. 依赖的传递性



(1).好处：可以传递的依赖不必再每个模块都重复声明，在“最下面”的工程中依赖一次即可

(2)非compile的依赖不能传递

B．依赖的排除的设置方式



不稳定jar包，不希望加入

确实需要的依赖

|  |
| --- |
| <exclusions>  <exclusion>  <groupId>commons-logging</groupId>  <artifactId>commons-logging</artifactId>  </exclusion>  </exclusions> |

1. 依赖的原则
2. 解决jar包冲突问题
3. 情景设定1：路径就近原则（同时存在两个不同版本的jar时）
4. 情景设定2：验证路径相同时先声明这优先.
5. 统一管理依赖的版本

对于spring各个jar包的依赖版本都是4.0，如果需要升级为4.1 该怎么办？

建议配置方式

[1]使用properties标签统一声明版本号

|  |
| --- |
| <properties>  <atguigu.spring.version>4.0.0.RELEASE</atguigu.spring.version> </properties> |

[2]在需要统一版本的位置，使用$(自定义标签名)引用声明的版本号

|  |
| --- |
| <version>${ atguigu.spring.version }<version/> |

# 十六 继承

16.1 现状

Hello依赖junit 4.0

HelloFriend依赖的junit4.0

MakeFriend依赖的junit 4.9

由于test范围内的依赖不能传递，所有必然会分散在各个模块工程中，很容易造成版本不一致

16.2需求：统一管理各个模块工程中的对junit依赖的版本

16.3 解决思路：将junit依赖版本统一提到“父”工程中，在子工程中声明不指定版本。以父工程中统一设定的来管理，同时也便于管理

16.4操作步骤

（1）.创建一个Maven工程作为父工程：注意：打包的方式pom

|  |
| --- |
| <groupId>com.nanhua.maven</groupId>  <artifactId>Parent</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>pom</packaging> |

（2）在子工程声明对父工程的引用

|  |
| --- |
| <!-- 在子工程中声明父工程 -->  <parent>  <groupId>com.nanhua.maven</groupId>  <artifactId>Parent</artifactId>  <version>0.0.1-SNAPSHOT</version>    <!-- 一当前工程的pom.xml文件为基准的父工程pom.xml文件的相对路径 -->  <relativePath>../Parent/pom。xml </relativePath>  </parent> |

（3）将子工程的坐标中与父工程中重复的内容删除



1. 在父工程中统一junit的依赖

|  |
| --- |
| <!-- 统一配置的管理 -->  <dependencyManagement>  <dependencies>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.0</version>  <scope>test</scope>  </dependency>  </dependencies>  </dependencyManagement> |

1. 在执行安装时先安装父工程

# 十七 聚合

17.1作用：一键安装各个模块工程

|  |
| --- |
| <!-- 配置聚合 -->  <modules>  <!-- 指定各个子工程的相对路径 -->  <module>../Hello</module>  <module>../HelloFriend</module>  <module>../MakeFriend</module>  </modules> |

17.2在聚合工程的pom.xml中运行安装