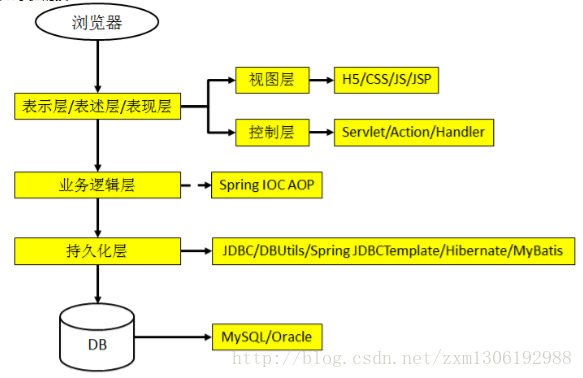
Maven

# Maven介绍

## Maven作用

自动化构建工具，服务于Java平台的项目构建和依赖管理

构建：编译（生成字节码文件）🡪部署（将可运行的程序放在服务器上）🡪搭建



## Maven优点

Maven可以将一个项目拆分为多个工程

将java依赖的jar包统一保存在仓库中，避免其他工程重复下载

Maven将所有的第三方jar包按照规范统一存放在中央仓库中

Maven会自动把被依赖的jar包导入进来

## 搭建的各个环节

清理：将之前的class文件清理，为下次编译做准备

编译：将java源码编译成字节码文件

测试：对应规定模块，自动运行测试文件，调用junit程序

报告：打印测试程序运行结果

打包：动态web工程打war包，java工程打jar包

安装：Maven-将打包得到的文件复制到仓库指定位置

部署：将动态web工程的war包复制到servlet容器的指定目录，使其可以运行

## Maven使用

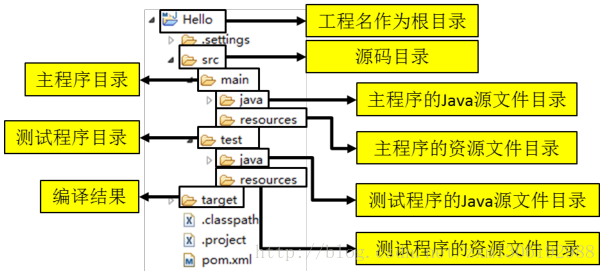
确定JAVA\_HOME环境变量

解压maven程序

配置Maven环境变量：MAVE\_HOME，path

# Maven的核心概念

## 约定的目录结构



pom.xml文件为Maven工程的核心配置文件

### 为什么要遵循约定的目录结构呢？

我们在开发中如果需要让第三方工具或框架知道我们自己创建的资源在哪，那么基本上就是两种方式：

①以配置文件的方式明确告诉框架 如 < param-value>classpath:spring-context.xml < /param-value>

②遵循框架内部已经存在的约定 如log4j的配置文件名规定必须为 log4j.properties 或 log4j.xml ；Maven 使用约定的目录结构

### pom.xml文件

<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.ccnu.maven</groupId>

<artifactId>MyMaven</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>servlet-api</artifactId>

<version>2.5</version>

<scope>provided</scope>

</dependency>

</dependencies>

</project>

### Maven常用命令

注意：执行与构建过程相关的Maven命令，必须进入pom.xml 所在的目录。

常用命令

【1】mvn clean : 清理

【2】mvn compile : 编译主程序

【3】mvn test-compile : 编译测试程序

【4】mvn test : 执行测试

【5】mvn package : 打包

【6】mvn install ： 安装

【7】mvn site ：生成站点

### 联网问题

* Maven 的核心程序中仅仅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。而插件本身不包含在Maven核心程序中。
* 当我们执行的Maven命令需要用到某些插件时，Maven核心程序会首先到本地仓库中查找。
* 本地仓库的默认位置：[系统登陆用户的家目录] \ .m2\repository
* Maven核心程序如果在本地仓库中找不到需要的插件，那么它会自动连接外网，到中央仓库下载。
* 如果此时无法连接外网，则构建失败。
* 修改默认本地仓库的位置可以让Maven核心程序到我们事先准备好的目录下查找插件

①找到Maven解压目录\conf\settings.xml

②在setting.xml 文件中找到 localRepository 标签

③将 < localRepository>/path/to/local/repo< /localRepository>从注释中取出

④将标签体内容修改为自定义的Maven仓库目录

## POM

含义：Project Object Model 项目对象模型

DOM ：Document Object Model 文档对象模型

pom.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进行配置。

重要程度相当于web.xml 对于动态web工程

## 坐标

Maven的坐标：使用下面三个向量在仓库中唯一定位一个Maven工程

①groupid:公司或组织域名倒序+项目名

< groupid>com.atguigu.maven< /groupid>

②artifactid:模块名

< artifactid>Hello< /artifactid>

③version：版本

< version>1.0.0< /version>

Maven 工程的坐标与仓库中路径的对应关系，以spring为例

< groupId>org.springframework< /groupId>

< artifactId>spring-core< /artifactId>

< version>4.0.0.RELEASE< /version>

org/springframework/spring-core/4.0.0.RELEASE/spring-core-4.0.0.RELEASE.jar

注意：我们自己的 Maven 工程必须执行安装操作才会进入仓库。安装的命令是：mvn install

## 依赖

当 A jar 包用到了 B jar 包中的某些类时，A 就对 B 产生了依赖，这是概念上的描述。Maven解析依赖信息时会到仓库中查找被依赖的jar包。

对于我们自己开发的Maven工程，要使用mvn install 命令安装后就可以进入仓库。



### 依赖的范围

**scope：compile、test、provided**

compile范围依赖

》对主程序是否有效：有效

》对测试程序是否有效：有效

》是否参与打包：参与

》是否参与部署：参与

》典型例子：spring-core

test范围依赖

》对主程序是否有效：无效

》对测试程序是否有效：有效

》是否参与打包：不参与

》是否参与部署：不参与

》典型例子：Junit

provided范围依赖

》对主程序是否有效：有效

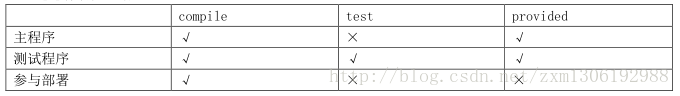
》对测试程序是否有效：有效

》是否参与打包：不参与

》是否参与部署：不参与

》典型例子：Servlet-api.jar

总结



### 依赖的传递

A依赖B，B依赖C，A能否使用C呢？要看B依赖C的范围是不是compile



### 依赖排除

如果我们当前工程中引入了一个依赖是A，而A又依赖了B，那么Maven会自动将A依赖的B引入当前工程，但是个别情况下B有可能是一个不稳定版本，或对当前工程有不良影响。这时我们可以在引入A的时候将B排除。

<dependency>

<groupId>com.atguigu.maven</groupId>

<artifactId>HelloFriend</artifactId>

<version>0.0.1-SNAPSHOT</version>

<type>jar</type>

<scope>compile</scope>

<exclusions>

< exclusion>

<groupId>commons-logging</groupId>

<artifactId>commons-logging</artifactId>

</ exclusion>

</exclusions>

</dependency>

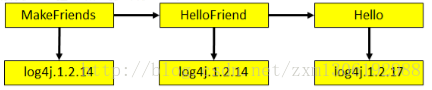
### 统一管理版本

①统一声明版本号  
这里写图片描述  
其中 atguigu.spring.version 部分是自定义标签。

②引用前面声明的版本号  
  
这里写图片描述

③其他用法  
这里写图片描述

### jar包冲突

①路径最短者优先  


②路径相同时先声明者优先  


## 仓库

仓库的分类

①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

②远程仓库

（1）私服：搭建在局域网环境中，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：为了分担中央仓库流量，提升用户访问速度

仓库中保存的内容：Maven工程

①Maven自身所需要的插件

②第三方框架或工具的jar包

③我们自己开发的Maven工程

不管是什么样的 jar 包，在仓库中都是按照坐标生成目录结构，所以可以通过统一的方式查询或依赖。

## 生命周期/插件/目标

各个构建环节执行的顺序：不能打乱顺序，必须按照既定的正确顺序来执行。

Maven的核心程序中定义了抽象的生命周期，生命周期中各个阶段的具体任务是由插件来完成的。

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各个阶段：不论现在要执行生命周期中的哪一阶段，都是从这个生命周期最初的位置开始执行。

Maven有三套相互独立的生命周期，分别是：  
①Clean Lifecycle 在进行真正的构建之前进行一些清理工作。  
②Default Lifecycle 构建的核心部分，编译、测试、打包、安装、部署等等。  
③Site Lifecycle 生成项目报告，站点，发布站点。

他们相互独立。也可以直接运行 mvn clean install site 运行所有这三套生命周期。

每套生命周期都由一组阶段(Phase)组成，我们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 clean 是 Clean 生命周期的一个阶段。有 Clean 生命周期，也有 clean 阶段。

* Clean声明周期  
  ①pre-clean 执行一些需要在clean之前完成的工作  
  ②clean 移除所有上一次构建生成的文件  
  ③post-clean 执行一些需要在clean 之后立刻完成的工作
* Default声明周期  
  Default 生命周期是 Maven 生命周期中最重要的一个，绝大部分工作都发生在这个生命周期中。这里，只解释一些比较重要和常用的阶段：  
  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或部署到服务器上运行。
* Site生命周期  
  ①pre-site 执行一些需要在生成站点文档之前完成的工作  
  ②site 生成项目的站点文档  
  ③post-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  
  ④site-deploy 将生成的站点文档部署到特定的服务器上

这里经常用到的是 site 阶段和 site-deploy 阶段，用以生成和发布 Maven 站点，这可是 Maven 相当强大的功能，Manager 比较喜欢，文档及统计数据自动生成，很好看。

插件和目标

Maven的核心仅仅定义了抽象的声明周期，具体的任务都是交由插件完成的。

每个插件都实现多个功能，每个功能就是一个插件目标

Maven的生命周期与插件目标相互绑定，以完成某个具体的构建任务。可以将目标看做“调用插件功能的命令”

例如：

compile 就是插件 maven-compiler-plugin 的一个目标；

pre-clean 是插件 maven-clean-plugin 的一个目标。

## 继承

为了统一管理各个模块工程中对Junit依赖的版本，将Junit依赖统一提取到“父”工程中，在子工程中声明Junit依赖是不指定版本，以父工程中统一设定的为准。同时也便于修改。

方式：

①创建一个Maven工程作为父工程。注意：打包方式为pom



②在子工程中声明对父工程的引用



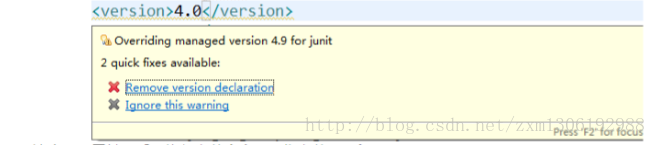
③将子工程的坐标中与父工程坐标中重复的内容删除



④在父工程中统一管理Junit的依赖



⑤在子工程中删除Junit依赖的版本号部分



注意：配置集成后，执行安装命令时要先安装父工程。

## 聚合

作用：一键安装各个模块工程。

配置方式：在一个“总的聚合工程”中配置各个参与聚合的模块



使用方式：在聚合工程的pom.xml 上点右键->run as->maven install

## Maven\_Web工程的自动部署（了解）

在pom.xml 中添加如下配置：

<!--配置当前工程构建过程中的特殊设置 -->

<build>

<finalName>AtguiguWeb</finalName>

<!-- 配置构建过程中需要使用的插件 -->

<plugins>

<plugin>

<!-- cargo是一家专门从事启动Servlet容器的组织 -->

<groupId>org.codehaus.cargo</groupId>

<artifactId>cargo-maven2-plugin</artifactId>

<version>1.2.3</version>

<!-- 针对插件进行的配置 -->

<configuration>

<!-- 配置当前系统中容器的位置 -->

<container>

<containerId>tomcat6x</containerId>

<home>D:\DevInstall\apache-tomcat-6.0.39</home>

</container>

<configuration>

<type>existing</type>

<home>D:\DevInstall\apache-tomcat-6.0.39</home>

<!-- 如果Tomcat端口为默认值8080则不必设置该属性 -->

<properties>

<cargo.servlet.port>8989</cargo.servlet.port>

</properties>

</configuration>

</configuration>

<!-- 配置插件在什么情况下执行 -->

<executions>

<execution>

<id>cargo-run</id>

<!-- 生命周期的阶段 -->

<phase>install</phase>

<goals>

<!-- 插件的目标 -->

<goal>run</goal>

</goals>

</execution>

</executions>

</plugin>

</plugins>

</build>

# Eclipse中使用Maven

1、Maven插件Eclipse已经内置。

2、Maven插件的设置： Window->Preferences->Maven

①installations : 指定Maven核心程序的位置。默认是插件自带的Maven程序，改为我们自己解压的那个。

②user settings : 指定Maven核心程序中 conf/settings.xml 文件的位置，进而获取本地仓库的位置。

## 创建Maven版的Java工程

创建的Maven工程默认使用的是JDK1.5，打开Maven核心程序settings.xml文件，找到profiles标签，加入如下配置，即可更改

<profile>

<id>jdk-1.7</id>

<activation>

<activeByDefault>true</activeByDefault>

<jdk>1.7</jdk>

</activation>

<properties>

<maven.compiler.source>1.7</maven.compiler.source>

<maven.compiler.target>1.7</maven.compiler.target>

<maven.compiler.compilerVersion>1.7</maven.compiler.compilerVersion>

</properties>

</profile>

## 创建Maven版的Web工程

1、New Maven project时，Packaging 选择 war

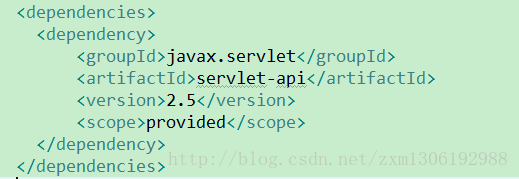
2、调整web目录结构，在项目上点右键 -> properties->Project Facets -> 把Dynamic Web Module 勾选去掉，并Apply -> 将Dynamic Web Module 重新勾选 -> 点击Further configuration available -> 修改 Content directory为src/main/webapp -> Apply 即在Maven工程上生成动态Web目录结构

3、新建jsp文件发现报错：The superclass “javax.servlet.http.HttpServlet” was not found on the Java Build Path

因为缺少Tomcat 运行时环境

以Maven方式添加：

在pom.xml文件中添加



4、jsp写入EL表达式时发现报错：javax.servlet.jsp cannot be resolved to a type

将JSPAPI导入

在pom.xml文件中添加



注意：scope一定要是provided，不然jar包冲突，运行时会报空指针异常

## 执行Maven命令

选中pom.xml 右键 Run As->Maven build…->Goals->输入 compile ->点击 run

# Maven 酷站

我们可以到 http://mvnrepository.com/搜索需要的 jar 包的依赖信息。