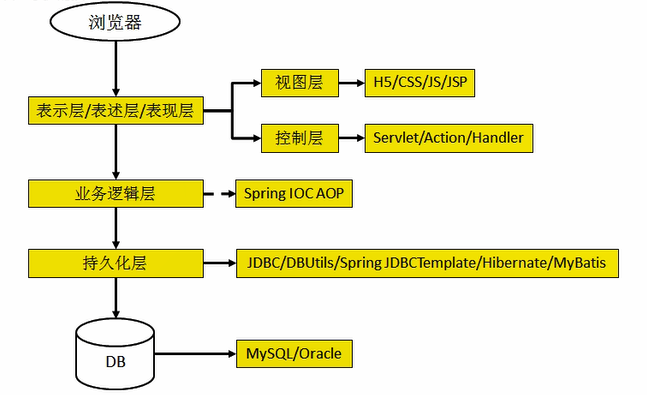
# Maven

Maven 是目前最流行的**自动化构建工具**，对于生产环境下多框架、多模块整合开发有重要作用。Maven 是一款在大型项目开发过程中不可或缺的重要工具。本课程将帮助你了解 Maven 的作用，常用命令，如何配置依赖，以及依赖的范围、依赖的传递性、依赖的排除、生命周期等重要概念，以及继承、聚合这样的 Maven 配置。更会手把手教你在 Eclipse 中玩转 Maven。

## 1/35目前掌握的技术



## 2/35目前的技术在开发中存在的问题

1. 一个项目就是一个工程

如果项目非常庞大,就不适合继续使用package来划分模块.最好是每一个模块对应一个工程,利于分工协作.

借助于Maven就可以将一个项目拆分成多个工程.

1. 项目中需要的jar包必须手动"复制","粘贴"到WEB-INF/lib目录下

带来的问题是:同样的jar包文件重复出现在不同的项目工程中,一方面浪费存储空间,另外也让工程比较臃肿.

借助于Maven,可以将jar包仅仅保存在”仓库”中,有需要使用的工程引用这个文件接口,并不需要真的把jar包复制过来.

1. Jar包需要别人替我们准备好,或到官网下载

不同技术的官网提供jar包下载的形式是五花八门的.

有些技术的官网就是通过Maven或者SVN等专门的工具来提供下载的.

如果是以不规范的方式下载的jar包,那么其中的内容很可能也是不规范的.

借助于Maven我们可以以一种规范的方式下载jar包,因为所有知名框架或者第三方工具的jar包以及按照统一的规范存放在了Maven的中央仓库中.

以规范的方式下载的jar包,内容也是可靠的.

Tips:”统一的规范”不仅是对IT开发领域非常重要,对于整个人类社会都是非常重要的.

1. 一个jar包依赖的其他jar包需要自己手动加入到项目中.

Fileupload包需要依赖io包.

如果所有的jar包之间的依赖关系都需要程序员自己非常清楚的了解,那么就会极大的增加学习的成本.

借助于Maven会自动将被依赖的jar包导入进来.

## 3/35构建的概念

[What]Maven是什么?

1. Maven是一款服务于Java平台的自动化的构建工具.

Make->Ant->Maven->Gradle

1. 什么是构建?

[1]概念:以”Java源文件”,”框架配置文件”,”JSP”,”HTML”,”图片”等资源为”原材料”,去生产一个可以运行的项目的过程.

->编译

->部署

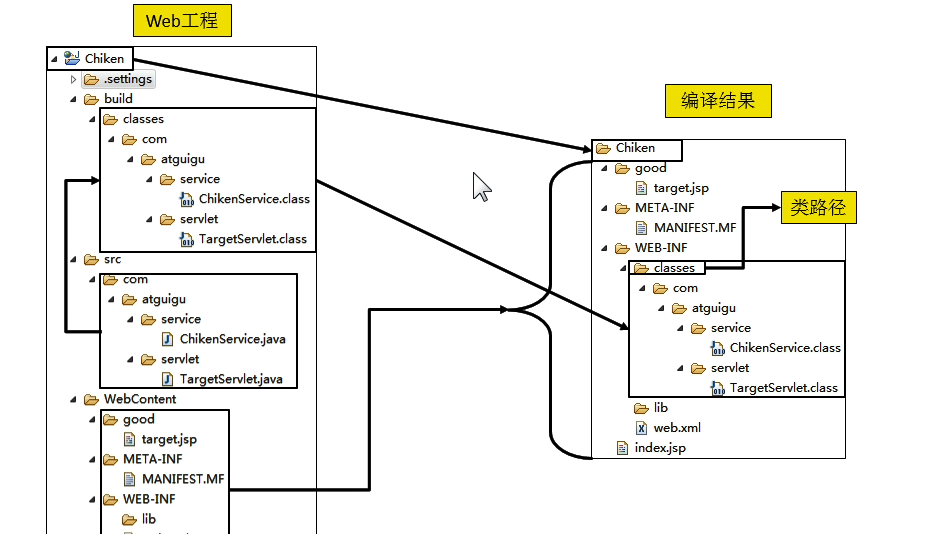
->搭建

[2]编译:Java源文件[User.java]->编译->Class字节码文件[User.class]->交给JVM去执行

[3]部署:一个BS项目最终运行的并不是动态WEB工程本身,而是这个动态WEB工程”编译的结果”.

生的鸡->处理->熟的鸡

动态web工程->编译,部署->编译结果



开发过程中,所有的路径或者配置文件中配置的类路径等都是以编译结果的目录结构为标准的.

## 4/35运行时环境



其实是一组jar包的引用,并没有把jar包本身复制到工程中,所以并不是目录.

## 5/35一只煮熟的鸡

TC\_server

一个插件,整个目录复制到eclipse解压安装目录下的dropins目录下即可

## 6/35自动化构建和构建环节

构建过程中的各个环节

[1]清理:将以前编译得到的旧的class字节码文件删除,为下一次编译做准备;

[2]编译:将java源程序编程成class字节码文件

[3]测试:自动测试,自动调用 junit程序

[4]报告:测试程序执行的结果

[5]打包:动态web工程打war包,java工程打Jar包

[6]安装:Maven特定的概念---将打包得到的文件复制带”仓库”中指定位置

[7]部署:将动态web工程生产的war包复制到Servlet容器的指定目录下,使其可运行.

自动化构建?

## 7/35部署Maven核心程序

1. 检查JAVA\_HOME环境变量
2. 解压Mavne核心程序的压缩包;放在一个非中文,无空格的目录下!
3. 配置maven的环境变量
   1. MAVEN\_HOME或者M2\_HOME
   2. path

## 8/35约定的目录结构说明

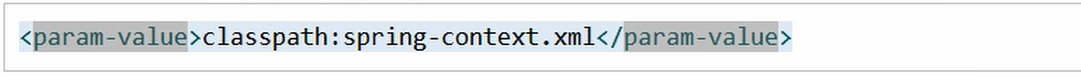
Maven的核心概念

1. 约定的目录结构
2. POM
3. 坐标
4. 依赖
5. 仓库
6. 生命周期/插件/目标
7. 继承
8. 聚合

第一个Maven工程

1. 创建约定的目录结构
2. 根目录:工程名
3. Src目录:源码
4. Pom.xml文件:Maven工程的核心配置文件
5. Main目录:存放主程序
6. Test目录:存放测试程序
7. Java目录:存放Java源文件
8. Resource目录:存放框架或其他工具的配置文件
9. 为什么要遵守约定的目录结构?

* Maven要负责我们这个项目的自动化构建,以编译为例,Maven要想自动进行编译,那么它必须知道Java源文件保存在哪里.
* 如果我们自己定义的东西想让框架或工具知道,有两种办法
  + 以配置的方式明确告诉框架



* + 遵守框架内部已经存在的约定

Log4j.properties

Log4j.xml

* 约定>配置>编码

## 9/35第一个Maven工程的目录结构和文件内容

## 10/35Maven联网问题

常用Maven命令

1. 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.

与构建过程相关:编译,测试,打包...

1. 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. Mvc test:执行测试
   5. Mvn package:打包
   6. Mvn install:安装
   7. Mvn site:生成站点

关于联网问题:

1. 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由特定的插件来完成,而插件本身并不包含在Maven的核心程序中.
2. 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
3. 本地仓库默认位置:[系统中当前用户的家目录]\.m2\repository
4. Maven核心程序如果在本地仓库中找不到需要的插件,那么他会自动连接网络,到中央仓库下载;
5. 如果此时无法连接网络,则构建失败;
6. 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\setting.xml
   2. 在settings.xml文件中找到localRepository标签
   3. 将<localRepository>/path/to/local/repo</localRepository>从注释中取出
   4. 将标签内容修改为已经准备好的Maven仓库目录

## 11/35几个常用的maven命令

## 12/35Maven\_POM

POM

1. 含义:Project Object Model 项目对象模型

DOM Document Object Model文档对象模型

1. 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.

## 13/35Maven\_坐标

坐标

1. 数学中的坐标:
   1. 在平面上,使用X,Y两个向量可以唯一的定位平面中的任何一个点.
   2. 在空间中,使用X,Y,Z三个向量可以唯一的定位空间中的任何一个点;
2. 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

[1]**g**roupid:公司或者组织域名倒叙+项目名

<groupid>com.epfresh.maven</groupid>

[2]**a**rtifactid:模块名

<artifactid>Hello</artifactid>

[3]**v**ersion:版本

<version>1.0.0</version>

1. Maven工程的坐标与仓库中的路径的对应关系

|  |
| --- |
| <groupId>org.springframework</groupId>  <artifactId>spring-core</artifactId>  <version>4.0.2.RELEASE</version> |
| Org/springframework/spring-core/4.0.2.RELEASE/spring-core-4.0.2.RELEASE.jar |

## 14/35Maven\_仓库

仓库

1. 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:为了分担中央仓库的流量,提升用户访问速度;

②仓库中保存的内容:Maven工程

[1]Maven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

## 15/35第二个Mavne工程的目录结构和文件内容

## 16/35依赖的基本操作

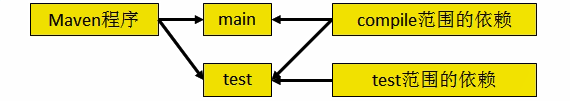
依赖[初步]

1. Maven解析依赖信息时会到本地仓库中查找依赖的jar包

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

## 17/35依赖的范围初步介绍

1. 依赖的范围



[1]compile范围的依赖

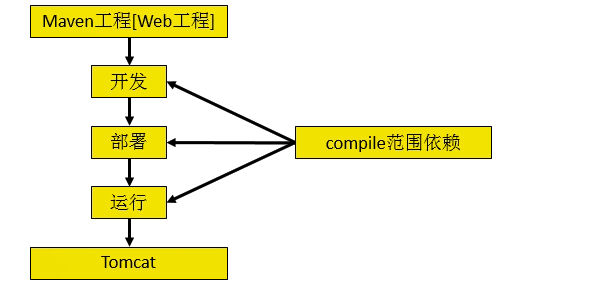
* + 对主程序是否有效:有效
  + 对测试程序是否有效:有效
  + 是否参与打包:**参与**
  + 是否参与部署:**参与**
  + 典型的例子:spring-core

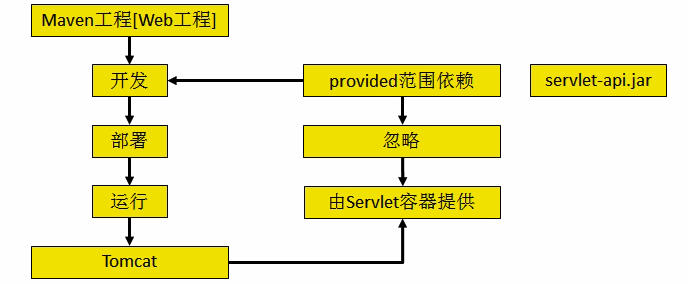
[2]test范围依赖

* + 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
  + 典型的例子:junit

[3]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**
  + 典型的例子:servlet-api.jar





## 18/35生命周期插件和目标

生命周期

1. 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.
2. 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;
3. 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阶段:不论现在要执行生命周期中的哪一个阶段,都是从这个生命周期最初的位置开始执行!







1. 插件和目标

[1]生命周期的各个阶段仅仅定义了要执行的任务是什么

[2]各个阶段和插件的目标是对应的

[3]相似的目标由特定的插件来完成



[4]可以将目标看作”调用插件功能的命令”

## 19/35Maven插件的设置

在eclipse中使用Maven

1. Maven插件:eclipse内置
2. Maven插件的设置

[1]installations:指定Maven核心程序的位置.不建议使用插件自带的

[2]user settings:指定conf/settings.xml的位置,从而获得本地仓库的位置.

## 20/35创建Maven版Java工程

1. 基本操作

[1]穿件Maven版的java工程

[2]创建Maven版的Web工程

参见: 尚硅谷\_Maven03操作步骤截图.pdf

## 21/35在eclipse中执行java命令

[3]执行maven命令

## 22/35在eclipse中创建Maven版的Web工程

修改project facets中Dynamic web module

让maven项目给你创建web.xml文件

## 23/35通过servlet-api进一步理解provided依赖范围

## 24/35通过jsp-api体验jar包之间的冲突

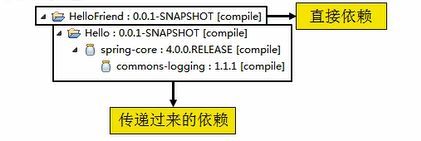
## 25/35导入maven工程

## 26/35第三个Mavne工程

## 27/35依赖的传递性

依赖[高级]

1. 依赖的传递性



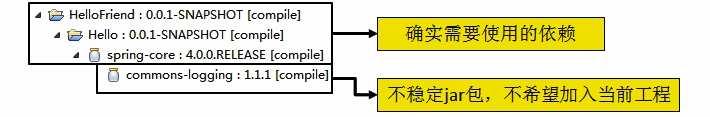
[1]好处:可以传递的依赖不必在每个模块工程中都重复生命,在”最下面”的工程中依赖一次即可.

[2]注意:非compile范围的依赖不能传递.所以在各个工程模块,如果有需要就得重复声明依赖;

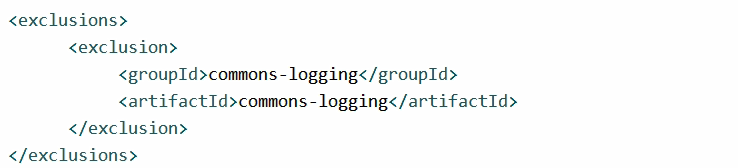
## 28/35依赖的排除

②依赖的排除

[1]需要设置依赖排除的场合



[2]依赖排除的设置方式

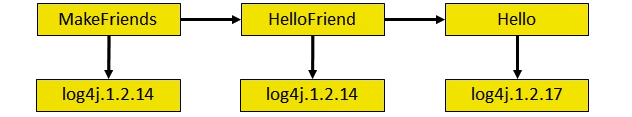


## 29/35依赖的原则说明

1. 依赖的原则

[1]作用:解决jar包冲突问题

[2]情景设定1:验证路径最短者优先原则



[3]情景设定2:验证路径相同时先声明者优先



先声明指的是dependency标签的声明顺序

## 30/35依赖原则测试

## 31/35统一管理依赖的版本号

[1]情景举例



这里对spring各个jar包的依赖版本都是4.0.0

如果需要统一升级为4.1.1,怎么办?手动逐一修改不可靠.

[2]建议配置方式

(1)使用properties标签内使用自定义标签统一声明版本号



(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



[3]其实properties标签配合自定义标签声明数据的配置并不是只能用于声明依赖的版本号.凡是需要统一声明后再引用的场合都可以使用.



## 32/35Maven继承

继承

1. 现状

Hello依赖junit:4.0

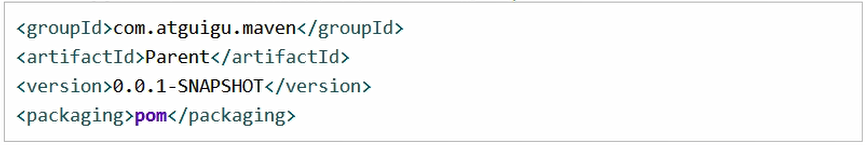
HelloFriend依赖junit:4.0

MakeFriends依赖junit:4.9

由于test范围的依赖不能传递,所以必然会分散在各个模块工程中,很容易造成版本不一致.

1. 需求:统一管理各个模块工程中对junit依赖的版本
2. 解决思路:将junit依赖统一提取到”父”工程中,在子工程中声明依赖时不指定版本,以父工程中统一设定的为准,同时也便于修改.
3. 操作步骤

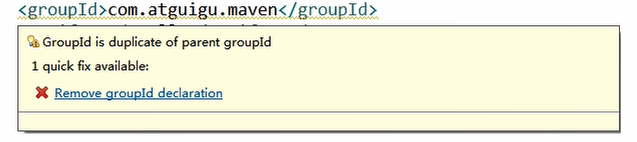
[1]**创建一个Maven工程作为父工程.注意:打包的方式pom**



[2]在子工程中声明对父工程的引用



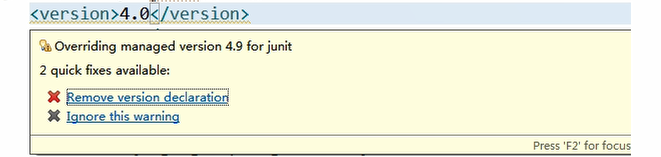
[3]将自工程的坐标中与父工程坐标中重复的内容删除



[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

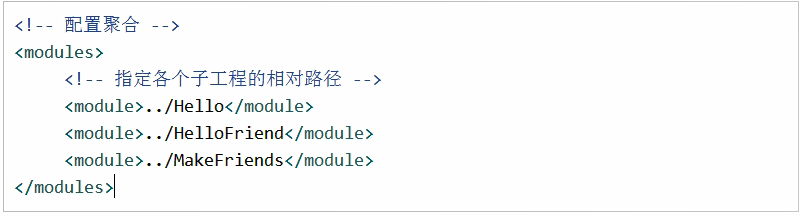


## 33/35Maven聚合

1. 注意:配置继承后,执行安装命令时,要先安装父工程.

聚合

1. 作用:一键安装各个模块工程;
2. 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.



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

## 34/35Maven\_web工程的自动化部署

## 35/35Maven\_查找依赖信息的网站

<https://mvnrepository.com/>