1. **Maven环境搭建**

下载安装目录 🡪 配置环境变量:M2\_HOME、Path 🡪 cmd测试配置

1. **基本目录结构**

src

-main

-java

-package

-test(测试)

-java

-package

resources（资源文件）

POM.xml管理项目(根目录中)

* 运行后生成的target文件中:classes文件存的是.class字节码文件；surefire-reports生成的是测试报告.

1. **基本命令**
2. mvn –v 查看maven版本
3. mvn compile编译
4. mvn test 测试
5. mvn package 项目打包
6. mvn clean 删除target
7. mvn install 安装jar包到本地仓库中
8. **自动创建目录骨架**

archetype插件:mvn archetyoe

1. archetype:generate 按照提示进行选择
2. archetype:generate -DgroupId=组织名，公司地址犯些+项目名

-Darchetype=项目名-模块名

-Dversion=版本号

-Dpackage=代码所存在的包名

1. **坐标和仓库**

* 坐标：groupId + artifactId + version
* 仓库：本地仓库

远程仓库(默认提供全球中央仓库的地址lib/maven-model-builder-3.3.9.jar中pom-4.0.0.xml)

仓库资源搜索:http://mvnrepository.com/

* 镜像仓库：（地址修改）

conf/settings.xml 中mirrors

中提供的示例复制一份

进行配置将镜像映射到国内

* 更改仓库位置:

默认存放位置:c:用户/.m2/repository

仓库不要放在c盘中，以防重装系统

修改settings.xml 中lacalRepository

url要改成斜杠

复制settings.xml到repo文件中，这样以后更新maven就不用修改settings.xml

1. **eclipse中maven的搭建**
2. 修改eclipse默认运行jre到jdk

Window 🡪 preference 🡪 java 🡪install jre

1. 修改eclipse.ini将jdk配置进去
2. 运行pom.xml: run as 🡪 maven builder

进行compile 、package等指令

1. 设置jdk目录
2. **生命周期**

* 三套相互独立的生命周期:

clean 清理项目:pre-clean clean post-clean

default 构建项目(最核心):compile test package install

site 生成项目站点(根据pom.xml生成站点):pre-site site post-site site-deploy

不同生命周期的各个阶段是顺序的，且后面的阶段依赖于前面的阶段

,但是并不会触发其他生命周期的任何阶段

例: clean、compile、test、package、install

在执行package命令时会自动执行之前的:clean、compile和test

1. **maven插件引入**

* 项目在打包时对source插件源码同时进行打包:绑定source插件到default生命周期的package阶段.
* Pom.xml中配置引入插件:

<!-- 引入插件 -->

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-source-plugin</artifactId>

<version>2.4</version>

<!-- 绑定插件到package阶段，可随项目一并打包 -->

<executions>

<execution>

<phase>package</phase>

<!-- 具体使用哪种类型上官网查goal值配置 -->

<goals>

<goal>jar-no-fork</goal>

</goals>

</execution>

</executions>

</plugin>

</plugins>

</build>

1. **pom.xml文件讲解(各个标签)**

<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"*>

<!-- 指定pom版本 -->

<modelVersion>4.0.0</modelVersion>

<!-- 坐标 -->

<groupId>com.imooc.hi</groupId><!-- 反写公司名+项目名 -->

<artifactId>hi</artifactId><!-- 项目名 + 模块名 -->

<!--第一个0标识大版本号，第二个0表示分支版本号，第三个0表示小版本号

snapshot 快照

alpha 内部测试

beta 公测

GA 正式发布

-->

<version>0.0.1-SNAPSHOT</version>

<!-- 项目默认打包类型:jar war zip pom等 -->

<packaging>jar</packaging>

<!-- 项目描述名 -->

<name>hi</name>

<!-- 项目的地址 -->

<url>http://maven.apache.org</url>

<!-- 项目描述 -->

<description></description>

<!-- 项目开发人员列表 -->

<developers></developers>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<!-- 依赖列表，包含多个依赖项 -->

<dependencies>

<dependency>

<!-- 依赖项的坐标设置 -->

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope><!-- 依赖范围设置 -->

<!-- 设置依赖是否可选，默认为false -->

<optional></optional>

<!-- 排除依赖传递列表 -->

<exclusions></exclusions>

</dependency>

</dependencies>

<!-- 依赖的管理,里面可以定义多个依赖，但不会引入到实际应用中，定义父类被子类所引用 -->

<dependencyManagement>

<dependencies>

</dependencies>

</dependencyManagement>

<!-- 引入插件 -->

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-source-plugin</artifactId>

<version>2.4</version>

<!-- 绑定插件到package阶段，可随项目一并打包 -->

<executions>

<execution>

<phase>package</phase>

<!-- 具体使用哪种类型上官网查goal值配置 -->

<goals>

<goal>jar-no-fork</goal>

</goals>

</execution>

</executions>

</plugin>

</plugins>

</build>

<!-- 子模块中对父模块pom的继承 -->

<parent></parent>

<!-- 指定多个模块一起编译 -->

<modules>

<module></module>

</modules>

</project>

1. **依赖范围**

* Maven提供三种classpath:

编译、测试、运行

* Scope的6个属性:

1. compile:对于编译、测试、运行三个阶段都有效(默认)
2. provided:编译、测试时有效(例:servletAPI)
3. runtime:在测试、运行时有效(例:jdbc驱动)
4. test:只在测试时有效(例:junit)
5. system:也在编译、测试时有效(项目依赖本机JAVA\_HOME,导致可移植性差)
6. import:只可在dependencyManagement标签中使用，表示将依赖导入到项目中

**十一、依赖传递**

* A依赖B,B又依赖C,所以A依赖C.
* 配置依赖关系步骤: c的pom.xml获取c的坐标 🡪 b的pom.xml中根据坐标配置依赖<dependency></dependency> 🡪 先将c进行打包并安装(clean install) 🡪 compile b 🡪

Clean install b 🡪 compile a.

* 这样搭建好之后，a中会有b和c两个依赖.
* 但是，如果a只需要b的依赖，不需要c中的依赖时:（排除依赖）

将对c依赖排除出去:<exclusions></exclusions>

**十二、依赖冲突**

* 当a依赖b，b依赖c(jar);a又依赖d，d也依赖c(jar);像这种依赖于相同构件，应该选择哪个构件的问题.

解决办法:短路优先,先声明先优先

**十三、聚合和继承**

* 聚合:

将项目聚合起来统一进行install,新项目中使用<module></module>

<modules>

<module>../hx.bg</module>

<module>../hx.ng</module>

<module>../hx.sj</module>

</modules>

* 继承:

将项目中共有的依赖项提取出来放入<dependencyManagement></dependencyManagement>中，作为父类 🡪 然后在要调用的子类使用<parent></parent>输入父类坐标进行引用

* 父类:

<dependencyManagement>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>$(junit-version)</version>

<scope>test</scope>

</dependency>

</dependencies>

</dependencyManagement>

* 子类:

<parent>

<groupId>com.hx</groupId>

<artifactId>hx.parent</artifactId>

<version>0.0.1-SNAPSHOT</version>

</parent>

**十四、创建基于maven的web项目**

* 流程:

新建时选择webapp 🡪 pom.xml中引入servlet依赖(从中央仓库中找) 🡪 手动创建基本目录结构 🡪 检查classpath的输出路径:右键项目buildpath /source/Output folder(确保此次输出路径为target/classes中!) 🡪 将项目转化成web项目:项目右键属性/project Facets/选择Dynamic Web Module动态web工程 🡪 修改部署时的默认配置:项目右键属性/Deployment Assembly(将测试代码文件删除，发布时不需要测试代码remove test相关代码) 🡪 web项目搭建成功!

* package命令导出为war包，部署到web容器(jetty)中

中央仓库中找到jetty插件的依赖(将坐标导入<plugin></plugin>中)

**备注：**

更新项目:maven/uapdate project