Maven的作用

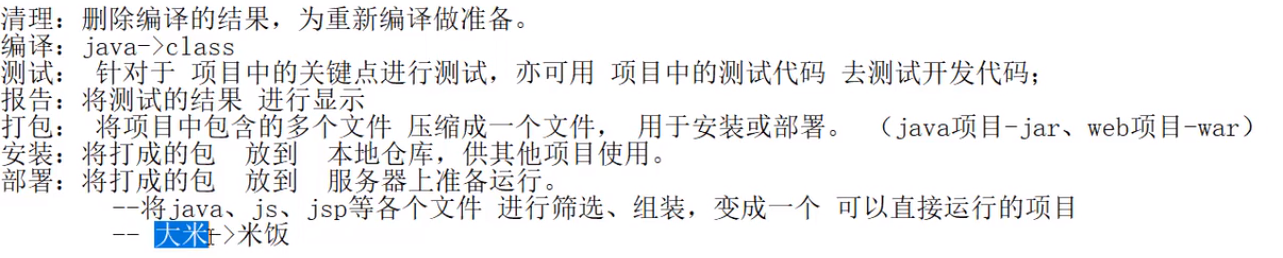
1. 管理jar包

* 添加第三方jar包（自动下载）
* 管理jar包之间的依赖关系，可自动关联下载一个jar包所依赖的jar包
* 避免jar因版本不同产生冲突

1. 将一个项目拆分成多个模块（子项目）

Maven的概念

一个基于Java平台的自动化构建工具（make 🡪 ant 🡪 maven 🡪 gradle）,包括清理、编译、测试、报告、打包、安装和部署等功能。



安装配置Maven

1. 下载Maven（https://maven.apache.org/）

binary为二进制文件，source为源代码；tar.gz是Linux系统下的压缩包，zip是Windows系统下的压缩包。



1. 配置环境变量，MAVEN\_HOME或M2\_HOME变量，path变量
2. 测试配置是否成功，命令行输入 mvn -v
3. **配置本地仓库**

在maven文件夹的conf\settings.xml文件中有仓库的配置信息：

<!-- localRepository

| The path to the local repository maven will use to store artifacts.

|

| Default: ${user.home}/.m2/repository

<localRepository>/path/to/local/repo</localRepository>

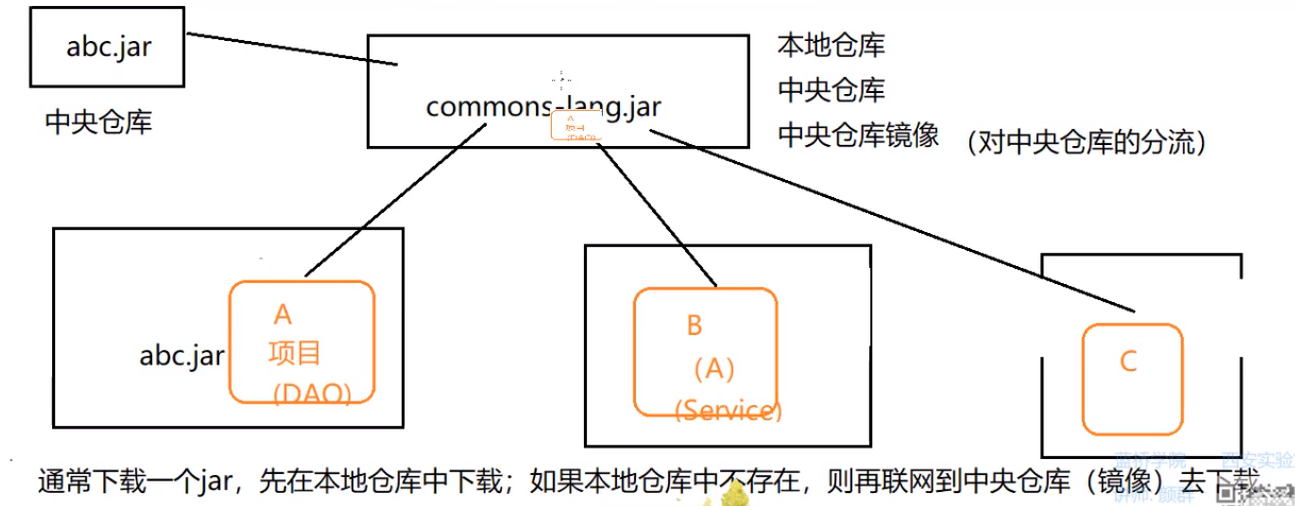
-->

修改本地仓库：添加“<localRepository>仓库文件夹路径</localRepository>”

仓库

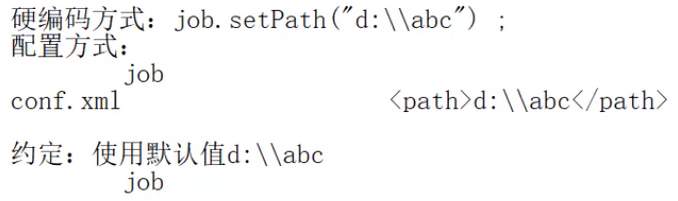
存放外部jar包和项目jar包的目录，用于各个模块从中应用相应的jar包，避免不同的项目重复添加同一jar包。

* 本地仓库：搭建在本机上的项目jar包仓库。
* 远程仓库：包括局域的私服和maven的中央仓库。
* 私服（nexus）：在局部建立的仓库，仅供数台计算机工作完成项目所需。
* 中央仓库：搭建在Maven服务器上的仓库。



使用Maven

**约定 优于 配置**



Maven约定的项目目录

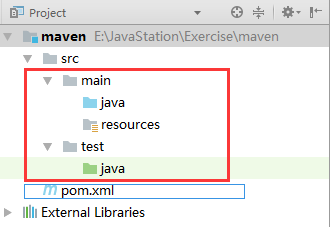
pom.xml：项目对象模型

-main目录：存放项目主要文件，包括java和resources两个子目录。

-test目录：存放测试文件的目录，包括java子目录。

--java目录：存放java源文件的目录，包括测试文件。

--resources目录：存放项目的配置文件等资源文件。



pom.xml

配置定位一个项目（模块）在参考中的位置。

groupId、artifactId、version，三个参数构成一个jar包的唯一标识，简称“GAV”。

<groupId>域名翻转.大项目名</groupId>

<artifactId>模块名称</artifactId>

<version>1.0-SNAPSHOT（版本）</version>

项目的依赖对象

使用<dependencies>标签先添加依赖群，再使用<dependency>标签逐个添加依赖的jar包。

<dependencies>

<dependency>

<groupId>……</groupId>

<artifactId>……</artifactId>

<version>……</version>

<scope>……</scope>

</dependency>

</dependencies>

添加依赖

到Maven的官方仓库（<http://mvnrepository.com/>）搜索jar包，选择版本后可以直接复制依赖配置。



Maven命令

必须在pom.xml所在目录下执行（在资源管理器的地址栏输入cmd可在当前目录下打开命令行窗口）

mvn compile：编译当前项目，只对main目录下的Java源文件进行编译，编译生成的class文件放置在一个target文件夹（与src同级）中。同时会读取pom.xml中配置好的依赖对象，从本地仓库中搜索，若没有发现目标jar包，则会到互联网上Maven的中央仓库去下载。

mvn test：对test目录下的测试类进行编译并执行。

mvn package：对项目进行打包，打包后的jar或war包存放在target目录下。

mvn install：将项目放入本地仓库，供其他模块使用。路径为“仓库/groupId/artifactId/version/jar包”。

mvn clean：清除编译产物，即删除target文件夹。

Maven的生命周期

三个阶段

1. **清理**：clean lifecycle，包括清理相关的几个过程

pre-clean、clean、post-clean

1. **默认**：default lifecycle，最常用的阶段，包括编译、测试、打包等等常用过程

compile、test-compile（编译测试代码）、test、package、install

1. **站点**：site lifecycle，包括以下过程

pre-site、site、post-site、site-deploy

Maven生命周期的各个过程是有一定顺序的，当执行后面的过程时，前面的过程会被自动执行。

Maven配置JDK的版本

<profile>

<id>自定义JDK的名称</id>

<activation>

<activeByDefault>是否激活（true/false）</activeByDefault>

<jdk>JDK版本</jdk>

</activation>

<properties>

<maven.compiler.source>JDK版本</maven.compiler.source>

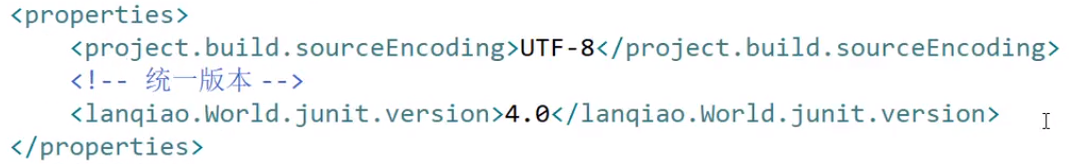
<maven.compiler.target>JDK版本</maven.compiler.target>

<maven.compiler.compilerVersion>JDK版本</maven.compiler.compilerVersion>

</properties>

</profile>

配置pom.xml属性



在pom.xml中可以使用类似EL表达式的形式引用这些属性：${lanqiao.World.junit.version}

部署web应用

通常使用Maven将web项目打包成war包，放到Tomcat的webapps目录下即可（Tomcat会自动加压）。

依赖

依赖范围

在pom.xml文件中使用<scope>指明jar包的依赖范围，即在何种情形下获取引用的jar包是有效的。

scope的三个值：

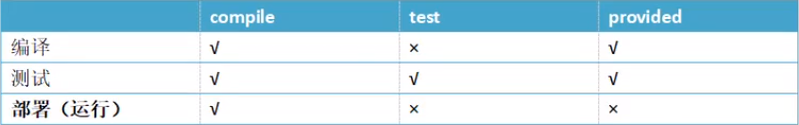
runtime：

compile：默认值，只要需要编译则依赖生效。

test：执行测试时才生效。

provided：项目开发阶段才有效，如对于servlet-api.jar，开发阶段需要从外部引用，当当项目部署到服务器上时，由于服务器自带该jar包，因而无需依赖仓库。

**scope的三个值在不同开发阶段依赖的有效性**



maven在编译、测试和执行时各使用一套classpath，均在target目录下。

依赖排除

当A.jar只有部分类依赖于B.jar，而在项目中使用的A.jar的功能不需要依赖B.jar时，可以使用依赖排除。

在<dependency>标签中添加<exclusions>标签，指定要排除的jar包的G和A即可。

<exclusions>

<exclusion>

<groupId>……</groupId>

<artifactId>……</artifactId>

</exclusion>

</exclusions>

依赖的传递性

若有A.jar 🡪 B.jar 🡪 C.jar

则要是A.jar 🡪 C.jar，在依赖A时能自动引入C的条件是：

当且仅当B依赖于C的范围是compile是才成立。

依赖原则

* 路径最短优先原则
* 同一pom.xml文件中配置有同一个依赖的不同版本是，后配置的版本会覆盖前面配置的版本
* 当依赖的同一个jar的不同版本路径相同，但在不同pom.xml中，先配置的优先

依赖的继承

一个项目继承另一个项目的所有依赖关系，被继承的项目成为父项目。

打包方式

* Java程序——jar
* web项目——war
* 父项目或聚合项目——pom

继承的步骤

1. 创建父项目时，选择打包方式为pom

* 父项目的src目录会与一般的Maven项目不同，补分main和test目录，一般只用于存放依赖信息。
* 父目录的pom.xml文件会自动添加一个<packaging>pom</packaging>标签。

1. 在父目录的<dependencies>外需要再添加一级<dependencyManagement>标签。
2. 在子项目的pom.xml中添加<parent>标签指示继承的父项目，指定父目录的GAV即可

* 若子项目和父项目具有相同的G和V，则在子项目的pom.xml中可省略对G和V的声明，直接从父项目继承

1. 在<parent>中添加<relativePath>标签，指明从子项目的pom.xml到父项目的pom.xml的相对路径。

* 返回上一级目录用“../”表示

1. 子项目中声明使用哪些父项目的依赖，添加一个<dependenciey>指明使用哪个依赖，但只需指定G和A。

依赖聚合

* 一个项目依赖于多个项目时，需要将被依赖项目install到仓库才能被依赖使用。
* 依赖聚合就是在总项目中对所有依赖进行配置，使项目在运行时被依赖的项目可以自动install。
* 依赖聚合只能在打包方式为pom的Maven项目中配置。

<modules>

<!-- 当前项目的pom.xml到被依赖项目的根目录（pom.xml所在目录）的相对路径 -->

<module>../根目录 </module>

<module>../根目录 </module>

</modules>