Maven学习文档

参考-

1）<https://blog.csdn.net/qq_33363618/article/details/79438044>

2）<https://blog.csdn.net/zhuralll112/article/details/83505487>

3）<http://c.biancheng.net/view/4715.html>

4）https://www.cnblogs.com/gongnol/p/9667010.html  
5）<https://www.cnblogs.com/baizihua/p/11519509.html>

6）

7）

1. Maven的安装   
    1先进入http://maven.apache.org/download.cgi ，下载对应版本的压缩包  
    2解压缩到一定目录，  
    bin目录：maven的运行文件。mvn.cmd正常运行mvn，mvnDebug.cmd以debug方式运行maven。  
    boot目录：maven运行需要类加载器。  
    conf目录：maven的配置文件目录，核心配置文件为其中的setting.xml。  
    lib目录：maven所需要的jar包。并配置环境变量

3配置环境变量，MAVEN\_HOME，压的目录，然后添加path%MAVEN\_HOME%/bin

4配置setting文件，可以更改仓库地址与镜像地址，不建议直接用中央仓库

5安装，在preference ->maven->选项usersetting，选择解压目录下的conf/setting文件

然后在maven->install选项，选择解压目录，安装到IDE里

1. Pom文件结构

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

<!-- 模型版本。maven2.0必须是这样写，现在是maven2唯一支持的版本 -->

<modelVersion>4.0.0</modelVersion>

<!-- 公司或者组织的唯一标志，并且配置时生成的路径也是由此生成， 如com.winner.trade，maven会将该项目打成的jar包放本地路径：/com/winner/trade -->

<groupId>com.winner.trade</groupId>

<!-- 本项目的唯一ID，一个groupId下面可能多个项目，就是靠artifactId来区分的 --> <artifactId>trade-core</artifactId>

<!-- 本项目目前所处的版本号 -->

<version>1.0.0-SNAPSHOT</version>

<!-- 打包的机制，如pom,jar, maven-plugin, ejb, war, ear, rar, par，默认为jar --> <packaging>jar</packaging>

<!-- 定义本项目的依赖关系 -->

<dependencies>

<!-- 每个dependency都对应这一个jar包 -->

<dependency>

<!--一般情况下，maven是通过groupId、artifactId、version这三个元素值（俗称坐标）来检索该构件， 然后引入你的工程。如果别人想引用你现在开发的这个项目（前提是已开发完毕并发布到了远程仓库），--> <!--就需要在他的pom文件中新建一个dependency节点，将本项目的groupId、artifactId、version写入， maven就会把你上传的jar包下载到他的本地 -->

<groupId>com.winner.trade</groupId>

<artifactId>trade-test</artifactId>

<version>1.0.0-SNAPSHOT</version>

<!-- maven认为，程序对外部的依赖会随着程序的所处阶段和应用场景而变化，所以maven中的依赖关系有作用域(scope) 的限制。 -->   
<!--scope包含如下的取值：compile（编译范围）、provided（已提供范围）、runtime（运行时范围）、test （测试范围）、system（系统范围） ，各值范围见此条最后-->

<scope>test</scope>

<!-- classifier的用途在于:

1使用maven-jar-plugin打包的时候，让名字带有指定值，暂不确定是多打还是只有加了

2引入依赖的时候，有时候仅凭groupId、artifactId、version无法唯一的确定某个构件，需要借助classifier来进一步明确目标。这种需要进一步指定的有多种情况，比如第一种1 JSON-lib，有时候会同一个版本会提供多个jar包，在JDK1.5环境下是一套，在JDK1.3环境下是一套,那便指定<classifier>jdk15</classifier>，或者2 org.apache.xxx.jar,这里的xxx同级目录下有多种可选，可以通过在<classifier>...</classifier>里书写对应名字来指定，如common-auth-0.0.1-SNAPSHOT.jar 与common-auth-0.0.1-SNAPSHOT-core.jarorg.apache.core.jar可以通过<classifier>core</classifier>指定添加后面那个-->

<classifier>...</classifier>

<!-- 设置指依赖是否可选，默认为false,即子项目默认都继承:为true,则子项目必需显示的引入，与 dependencyManagement里定义的依赖类似 -->

<optional>false</optional>

<!-- 屏蔽依赖关系。 比如项目中使用的libA依赖某个库的1.0版，libB依赖某个库的2.0版，现在想统一使用2.0版，就应该屏蔽掉对1.0版的依赖 -->

<exclusions>

<exclusion>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

</exclusion> </exclusions>

</dependency>

</dependencies>

<!-- 为pom定义一些常量，在pom中的其它地方可以直接引用 使用方式 如下 ：${file.encoding} -->   
<properties>

<file.encoding>UTF-8</file.encoding>

<java.source.version>1.5</java.source.version> <java.target.version>1.5</java.target.version>

</properties>

</project>

| **scope** | **编译** | **测试** | **运行** |
| --- | --- | --- | --- |
| compile | Y | Y | Y |
| test |  | Y |  |
| provided | Y | Y |  |
| runtime |  | Y | Y |
| system | Y | Y |  |

1. maven命令

cmd下执行 mvn +命令

1 clean clean是清除命令，执行clean会删除target下文件。

2 compile compile用来将src/main/java下的文件编译为class文件，并输出到target中。

3 test test 用来将src/main/test下的文件进行编译，同时执行一次。

4 package 打包,将项目进行打包，如果是jar打包为jar，war打包为war。

5 install 把项目打包成war包或jar包发布到本地仓库

6tomcat：run 一键启动

1. 插件