链滴

Hello,Maven

作者：liumapp

原文链接：https://ld246.com/article/1496220994707

来源网站：链滴

Hello,Maven

对于Maven的接触已经有很长的一段时间了，但是并没有做过系统的笔记，写这篇博文备忘一下。

Maven是什么

Maven与Composer一样，是一个资源依赖包的管理工具。

更官方一点，它是一个基于POM（项目对象模型），可以通过一小段描述信息来管理项目的构建、告和文档的软件项目管理工具。

Maven下载与配置

下载

请直接去Maven的官方站点下载源码

配置

这里我们假设Maven的目录位于/usr/local/maven（附：类Unix系统），并且您的系统已经具有Jav环境。

那么在执行如下操作：

```
cd ~

vim .bash_profile
```

.bash_profile下新增如下两行代码：

```
export M2_HOME=/usr/local/maven3.3.9/

export PATH=/usr/local/maven3.3.9/bin/:$PATH
```

再执行

```
source .bash_profile
```

测试

mvn -v

如果看见下列信息：

```
Apache Maven 3.3.9 (bb52d8502b132ec0a5a3f4c09453c07478323dc5; 2015-11-11T00:41:47+8:00)
```

```
Maven home: /usr/local/maven3.3.9

Java version: 1.8.0_112, vendor: Oracle Corporation

Java home: /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/jre

Default locale: zh_CN, platform encoding: UTF-8
```

那就说明您对Maven的配置已经完成了。

```
OS name: "mac os x", version: "10.11.6", arch: "x86_64", family: "mac"
```

案例

源码

源码请看我的项目java-core-learn

目录结构

```
--src
  --main
    --java（在这个目录下写java代码）
      --org
        --javacore
          --base
            --具体java文件
    --resources（资源文件）
  --test
    --base
      --具体java文件
```

maven配置文件pom.xml

代码

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/
MLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd
maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.javacore</groupId>
    <artifactId>java-core-learn</artifactId>
    <version>0.0.1-SNAPSHOT</version>

    <name>java core learning example</name>
```

```
    <url>http://www.liumapp.com<url>
    <description>java core learning example</description>
    <packaging>war</packaging>
    <developers>
        <developer>
            <name>liumapp</name>
        </developer>
    </developers>

    <properties>
        <google-collections.version>1.0</google-collections.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>com.google.collections</groupId>
            <artifactId>google-collections</artifactId>
            <version>${google-collections.version}</version>
        </dependency>

        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <version>1.16.8</version>
            <scope>provided</scope>
        </dependency>

        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

pom.xml分析

<xml>标签、<modelVersion>标签和<project>标签是pom的固定结构，我们不需要进行变化。

<groupId>的值就是项目的包名,一般而言，都是按照“组织名＋公司网址的反写＋项目名”这样的式来填写，比如说我的网址liumapp.com，我要创建一个helloworld项目，那么包名就应该是com.limapp.helloworld

<artifactId>的值是模块名，一般使用项目名称＋模块名进行标识

<version>表示版本号，比如0.0.1snapshot，第一个0表示大版本号，第二个0表示分支版本号，第个0表示小版本号，snapshot表示快照版本、alpha表示内侧版本、beta表示公测版本、Release表示定版本、GA表示正式发布版本。

<packaging>打包方式，默认是jar（不写这个标签的时候打包成jar），可以使用war、zip、pom等。

<dependencies>项目的依赖，经常使用到。<dependency>下面有一个<scope></scope>，它示依赖的范围。比如<scope>test</scope>，那么就说明这个依赖只在测试的范围内有用，超出这范围去使用就会报错。scope标签的值一共有六种，分别是：compile、provided、runtime、test、ystem、import。
compile是默认的节点，它表示编译测试和运行都有效。
provided是在测试和编译的时候有效。比如说Servlet的API，在测试和编译的时候有效，但如果在运的时候，就会和tomcat的servlet发生冲突。
runtime表示在测试和运行时有效。比如jdbc的驱动。
test表示只在测试的时候有效。比如junit
system表示编译和测试时有效，但有很大的不可移植性，只能本地使用。
import表示导入的范围，它只使用在dependencyManagement中，表示从其他的pom中导入depedency的配置。

<dependency>下面还有一个<optional>true/false<optional>默认为false，表示设置依赖是否可。如果为false，表示这个依赖是可以继承的，如果为true，则这个子项目必须显示引入该依赖。

<dependency>下面还有一个<exclusions>，它表示一个排除依赖传递的列表。比如说Ajar包依赖Bjr包，Bjar包依赖Cjar包，那么C对于A来说，就是一个传递依赖，如果A不想依赖C，就可以在这里面行声明。排除对C的依赖关系。
这里我很喜欢别人举的一个例子来解释依赖传递，在古惑仔里面，山鸡跟楠哥混，楠哥跟B哥混，所山鸡就间接的跟B哥混，所以山鸡跟B哥之间的关系，就是一个依赖传递的关系。那如果有一天，山鸡楠哥表忠心，说我只听你的，不听B哥的，那么山鸡这个项目里，在依赖楠哥的时候，还要再加一条对哥的排除依赖。

<dependencyManagement></dependencyManagement>表示依赖的管理。这个标签里面也是dependencies和dependency，但是这里面的依赖并不会被实际的引用。

<build>为构建行为提供支持，比如说使用maven的相关插件。

<parent>通常用于子模块中对父模块pom.xml的继承

<modules>定义多个模块然后一起编译。

<name>表示项目的描述名，一般用于在生成项目文档的时候使用。

<url>项目的地址。

<description>项目描述

<developers>项目开发人员列表

<license>

<organization>

运行

进入项目目录后，运行

mvn compile

maven会自动下载依赖的项目，并且完成编译。

再运行

mvn test

maven会自动的运行我们写在test目录下的测试用例。

再运行

mvn package

maven会自动对我们的项目进行编译后的打包

maven生成的文件

运行完上述说的命令后，项目根目录下应该会出现如下所示的目录

--target
--classes //编译后的文件
--maven-status
--surefire-reports//存放测试报告
--test-classes

Maven中的坐标和仓库

坐标与构件

构件：maven中，任何一个依赖，插件，项目构建的输出都可以被称之为构件。

构件通过坐标作为其唯一标识。

而坐标呢，则是由pom.xml文件中的groupId 、artifactId 和 version来组成。

仓库

仓库分为本地仓库和远程仓库。

maven在安装依赖之前，会先去本地的maven仓库中查找是否有相同的依赖，如果有，则不会去网下载，直接使用本地的依赖，如果没有，再去网上进行下载。这里的网上，则是指的maven的远程仓。

一般而言，maven默认的会给我们一个远程仓库地址：https://repo.maven.apache.org/maven2

这个地址我们可以在maven项目源码里面的org\apache\maven\model\pom-4.0.0.xml这个文件里找到。(被命名为：Central Repository，中央仓库)

镜像仓库

maven的中央仓库位置在国外，可能有些时候访问无法成功，这个时候我们就需要使用maven在国的镜像仓库。

使用方法

进入本地maven项目目录，进入conf目录，打开settings.xml文件

大概在146行左右的样子，找到mirrors标签。添加如下代码：

```
<mirror>
  <id>maven.net.cn</id>
  <mirrorOf>central</mirrorOf>
  <name>central mirror in china</name>
  <url>http://maven.net.cn/content/groups/public/</url>
</mirror>
```

保存后退出即可。

当然国内的话，还是阿里云的要快一点，所以我们可以使用这个地址：http://maven.aliyun.com/nexus/content/groups/public/

更改仓库位置

默认位置

默认情况下，maven安装的依赖都位于如下地址：

```
cd ~/.m2/repository
```

更改方法

依然在settings.xml文件中，我们找到第55行左右的<localRepository>标签，然后输入新的地址即。

```
<localRepository>/path/to/repository</localRepository>
```

Maven的生命周期

完整的项目构建过程包括：清理、编译、测试、打包、集成测试、验证、部署。

Maven生命周期：

clean 清理项目
    pre-clean 执行清理前的工作
    clean 清理上一次构建生成的所有文件
    post-clean 执行清理后的文件

default 构建项目（最核心的一部分）
    compile、test、package、install等等。

site 生成项目站点
    pre-site 在生成项目站点前要完成的工作
    site 生成项目的站点文档
    post-site 在生成项目站点后要完成的工作
    site-deploy 发布生成的站点到服务器上

在这几个clean、compile、test、package、install命令中，当我们运行package的时候，clean、compile、test将会在在package之前依次运行。

Maven的插件

官方插件列表：http://maven.apache.org/plugins/

对source插件的使用

source插件：将项目的源码进行打包

在pom.xml文件的project节点内添加如下代码：

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-source-plugin</artifactId>
      <version>2.4</version>
      <executions>
        <execution>
          <phase>package</phase>
          <goals>
            <goal>jar-no-fork</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

上面代码中的executions节点，用来绑定插件source到maven的package生命周期中。

goal目标节点的内容，可以在插件详情里面查看。

接下来运行package即可。

依赖冲突

比如说，我现在的项目为A，A依赖B项目，A也依赖C项目，B项目和C项目都依赖一个D项目，但是，B项目依赖D的1.0版本，而C项目依赖D的2.0版本，那这个时候怎么办呢。

短路优先

在maven中，首先要遵循短路优先原则。意思就是，优先解析路径短的版本。比如A依赖B，B依赖C C依赖X（jar，2.0版本），然后又有A依赖D，D依赖X（jar，1.0版本），那么最后maven会优先解析的1.0版本。

先声明先优先

如果路径长度相同，则原先声明，先解析谁。

聚合和继承

聚合

在maven中，如果想将多个项目进行install，将其安装到本地仓库中，必须对其依次执行install操作maven中有一种方式，可以将其放到一起运行，这种方式称之为聚合。

具体使用方法：

- 首先我们需要新建一个项目，然后将项目的打包方式修改为pom，也就是把packaging元素的值设为pom
- 然后在pom.xml文件中，添加如下代码：

```
<modules>
  <module>/path/to/your/moduleA</module>
  <module>/path/to/your/moduleB</module>
  <module>/path/to/your/moduleC</module>
</modules>
```

- 再在这个项目的目录下运行mvn install 即可。

继承

基本上不管哪一个maven项目，都会用到junit，而且每一个项目的pom.xml文件，都会对其进行配。这个时候就会出现很多重复的配置代码。

所以在maven中，我们可以像java一样，将共用的部分，写成一个父类。

具体使用方法：

- 新建一个maven项目（假设命名为parent），packaging改为pom，将要写入的依赖，写在depen

encyManagement中。如下所示：

```
<properties>
   <junit.version>4.12</junit.version>
</properties>

<dependencyManagement>
   <dependencies>
      <dependency>
         <groupId>junit</groupId>
         <artifactId>junit</artifactId>
         <scope>test</scope>
         <version>${junit.version}</version>
      </dependency>
   </dependencies>
</dependencyManagement>
```

- 在其他的maven项目中，去继承parent。代码如下所示：

在pom.xml文件中：

```
<parent>
 <groupId>parent的groupId</groupId>
 <artifactId>parent的artifactId</artifactId>=
 <version>parent的version</version>
</parent>

<dependencies>
   <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
   </dependency>
</dependencies>
```

如上所示，继承parent之后，再引用junit，已经不需要进行过多的配置了，只需要定义它的坐标即可。

常用命令

```
mvn -v 查看maven版本
   -compile 编译整个项目
   -test 执行test下面的测试代码
   -package 打包
   -clean 删除编译后的字节码文件和测试报告等等，也就是删除target
   -install 安装jar包到本地仓库中
```

在使用命令的时候，可以多个命令按照顺序使用，如：

```
mvn clean compile
```

就表示先clean，再compile

自动创建目录骨架

使用archetype插件，用于创建符合maven规定的目录骨架，具体代码如下：

```
mvn archetype:generate
```

如果是第一次运行的话，maven会自动的下载相关依赖，同时我们还要进行版本的选择。一般选择第个就可以了。

之后，会要求我们输入groupId，这里按照自己想要的项目包名输入。

然后是artifactId，输入项目名即可。

然后是version，版本号，一般第一次执行都是1.0.0SNAPSHOT

再是package,包名，输入项目包名再加一个-service即可。

最后进行确认整个项目的骨架就自动创建好了。

当然我们也可以直接使用一条命令完成整个过程：

```
 mvn archetype:generate -DgroupId=yourGroupId -DartifactId=yourArtifactId -Dversion=1.0.SNAPSHOT -Dpackage=yourPackage
```

利用maven发布web项目

相关源码可以看我的项目 jsp-basic

- 利用插件创建项目
- 添加对servlet的依赖，具体坐标请前往官网查找（很多依赖我们都需要在官网上去找）。这里推荐：ttp://mvnrepository.com/
- 设定servlet的依赖scope值为provided。
- packaging设定为war
- 添加maven的jetty插件。
- 执行mvn compiler，接下来就能够在浏览器成功访问到项目。
- 如果要使用tomcat作为web容器，那么就是用tomcat对maven的插件来取代jetty即可。

注意事项

请注意，maven在安装依赖之前，会先去本地的maven仓库中查找是否有相同的依赖，如果有，则会去网上下载，直接使用本地的依赖，如果没有，再去网上进行下载。

另外需要注意的是，如果我们有两个项目A和项目B，项目B需要引用项目A的类的话，通过maven，们只需要在项目A中执行命令mvn -install ，这样将会把A保存在本地仓库中，接下来再到项目B的po.xml文件中添加对A的dependency，再执行compaile，maven就能够自动的下载A的依赖。

在maven中，有三种classpath：编译、测试和运行。