**maven的dependency参数**

# 一、type

有时候我们引入某一个依赖时，必须指定type，这是因为用于匹配dependency引用和dependencyManagement部分的最小信息集实际上是{groupId，artifactId，type，classifier}。在很多情况下，这些依赖关系将引用没有classifier的jar依赖。这允许我们将标识设置为{groupId，artifactId}，因为type的默认值是jar，并且默认classifier为null。

type的值一般有jar、war、pom等，声明引入的依赖的类型。

# 二、classifier

Classifier可能是最容易被忽略的Maven特性，但它确实非常重要，我们也需要它来帮助规划坐标。设想这样一个情况，有一个jar项目，就说是 dog-cli-1.0.jar 吧，运行它用户就能在命令行上画一只小狗出来。现在用户的要求是希望你能提供一个zip包，里面不仅包含这个可运行的jar，还得包含源代码和文档，换句话说，这是比较正式的分发包。这个文件名应该是怎样的呢？dog-cli-1.0.zip？不够清楚，仅仅从扩展名很难分辨什么是Maven默认生成的构件，什么是额外配置生成分发包。如果能是dog-cli-1.0-dist.zip就最好了。这里的dist就是classifier，默认Maven只生成一个构件，我们称之为主构件，那当我们希望Maven生成其他附属构件的时候，就能用上classifier。常见的classifier还有如dog-cli-1.0-sources.jar表示源码包，dog-cli-1.0-javadoc.jar表示JavaDoc包等等。

classifier它表示在相同版本下针对不同的环境或者jdk使用的jar,如果配置了这个元素，则会将这个元素名在加在最后来查找相应的jar，例如：

```
<code><code><code>给相同的version，构建不同的环境使用依赖
<classifier>jdk17</classifier>
<classifier>jdk18</classifier></code></code></code>
```

# 三、optional

当project-A 依赖project-B，project-B 依赖project-D时

```
<code><code><code><dependency>
  <groupid>sample.ProjectD</groupid>
  ProjectD</artifactid>
  <version>1.0-SNAPSHOT</version>
  <optional>true</optional>
</dependency></code></code></code>
```

所以当project-B的true时，project-A中如果没有显式的引入project-D，则project-A不依赖project-D，即project-A可以自己选择是否依赖project-D

默认的值为false，及子项目必须依赖。

# 四、scope

scope的分类

compile

默认就是compile，什么都不配置也就是意味着compile。compile表示被依赖项目需要参与当前项目的编译，当然后续的测试，运行周期也参与其中，是一个比较强的依赖。打包的时候通常需要包含进去。

test

scope为test表示依赖项目仅仅参与测试相关的工作，包括测试代码的编译，执行。比较典型的如junit。

runntime

runntime表示被依赖项目无需参与项目的编译，不过后期的测试和运行周期需要其参与。与compile相比，跳过编译而已，说实话在终端的项目（非开源，企业内部系统）中，和compile区别不是很大。比较常见的如JSR×××的实现，对应的API jar是compile的，具体实现是runtime的，compile只需要知道接口就足够了。oracle jdbc驱动架包就是一个很好的例子，一般scope为runntime。另外runntime的依赖通常和optional搭配使用，optional为true。我可以用A实现，也可以用B实现。

provided

provided意味着打包的时候可以不用包进去，别的设施(Web Container)会提供。事实上该依赖理论上可以参与编译，测试，运行等周期。相当于compile，但是在打包阶段做了exclude的动作。

system

从参与度来说，也provided相同，不过被依赖项不会从maven仓库抓，而是从本地文件系统拿，一定需要配合systemPath属性使用。

import(only available in Maven 2.0.9 or later)

这个是maven2.0.9版本后出的属性，import只能在dependencyManagement的中使用，能解决maven单继承问题，import依赖关系实际上并不参与限制依赖关系的传递性。

使用import scope解决maven继承（单）问题

scope的依赖传递

A->B->C。当前项目为A，A依赖于B，B依赖于C。知道B在A项目中的scope，那么怎么知道C在A中的scope呢？答案是：
当C是test或者provided时，C直接被丢弃，A不依赖C；
否则A依赖C，C的scope继承于B的scope。

# 五、systemPath

当maven依赖本地而非repository中的jar包，sytemPath指明本地jar包路径,例如：

```
<code><code><code><code><code><code><dependency>
    <groupid>org.hamcrest</groupid>
    hamcrest-core</artifactid>
    <version>1.5</version>
    <scope>system</scope>
    <systempath>${basedir}/WebContent/WEB-INF/lib/hamcrest-core-
1.3.jar</systempath>
</dependency></code></code></code></code></code></code>
```

# 六、exclusions

依赖排除，就是有时候引入某一个依赖的时候，该依赖下有jar包冲突，可以排除掉，不引用该jar，例如：

```
<code><code><code><code><code><code>        <dependency>
            <groupid>test</groupid>
            test</artifactid>
            <version>1.0.2-SNAPSHOT</version>
            <exclusions>
                <exclusion>
                    <groupid>org.springframework</groupid>
                    spring</artifactid>
                </exclusion>
                <exclusion>
                    slf4j-log4j12</artifactid>
                    <groupid>org.slf4j</groupid>
                </exclusion>
            </exclusions>
        </dependency></code></code></code></code></code></code>
```