InfoQ Maven专栏（一）——坐标规划

December 13th, 2010

坐标是什么？为什么要规划？

坐标是Maven最基本的概念，它就像每个构件的身份证号码，有了它我们就可以在数以千万计的构件中定位任何一个我们感兴趣的构件。举个最简单的例子，如果没有坐标，使用JUnit的时候，用户就需要去下载依赖jar包，用依赖的方式，简单配置使用如junit:junit:4.8.2就可以了。这里第一个junit是groupId，第二个junit是artifactId，4.8.2是version。

Maven的很多其他核心机制都依赖于坐标，其中最显著的就是仓库和依赖管理。对于仓库来说，有了坐标就知道在什么位置存储构件的内容，例如junit:junit:4.8.2就对应仓库中的路径/junit/junit/4.8.2/junit-4.8.2.pom和/junit/junit/4.8.2/junit-4.8.2.jar这样的文件，读者可以直接访问中央仓库地址看到这样的仓库布局，或者浏览本地仓库目录~/.m2/repository/以获得直观的体验。

依赖的配置也是完全基于坐标的，例如：

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.8.2</version>
  <scope>test</scope>
</dependency>
```

有了正确的坐标，Maven才能够在正确的位置找到依赖文件并使用，这里值为test的scope是用来控制该依赖只在测试时可用，与坐标无关。

正因为坐标是Maven核心的核心，因此规划正确的坐标至关重要。如果你使用了模糊不清的坐标，那么你的用户就很难找到你的构件，或者即使找到了，也容易写错。错误的使用坐标，还会造成冲突，如果你也使用junit这样的groupId，那会发生什么？下面先看一些不是很规范的坐标使用方式。

坐标规划的原则

滥用坐标、错用坐标的样例比比皆是，在中央仓库中我们能看到SpringFramework有两种坐标，其一是直接使用springframework作为groupId，如springframework:spring-beans:1.2.6，另一种是用org.springframework作为groupId，如org.springframework:spring-beans:2.5。细心看看， 前一种方式显得比较随意，后一种方式则是基于域名衍生出来的，显然后者更合理，因为用户能一眼根据域名联想到其Maven坐标，方便寻找。因此新版本的 SpringFramework构件都使用org.springframework作为groupId。由这个例子我们可以看到坐标规划一个原则是*基于项目域名衍生*。其实很多流行的开源项目都破坏了这个原则，例如JUnit，这是因为Maven社区在最开始接受构件并部署到中央仓库的时候，没有很严格的限制。而对于这些流行的项目来说，一时间更改坐标会影响大量用户，因此也算是个历史遗留问题了。

还有一个常见的问题是将groupId直接匹配到公司或者组织名称，因为乍一看这是显而易见的。例如组织是zoo.com，有个项目是dog，那有 些人就直接使用groupId com.zoo了。如果项目只有一个模块，这是没有什么问题的，但现实世界的项目往往会有很多模块，Maven的一大长处就是通过多模块的方式管理项目。 那dog项目可能会有很多模块，我们用坐标的哪个部分来定义模块呢？groupId显然不对，version也不可能是，那只有artifactId。因 此要这里有了另外一个原则，*用artifactId来定义模块*，而不是定义项目。接下来，很显然的，*项目就必须用groupId来定义*。因此对于dog项目来说，应该使用groupId com.zoo.dog，不仅体现出这是zoo.com下的一个项目，而且可以与该组织下的其他项目如com.zoo.cat区分开来。

除此之外，artifactId的定义也有最佳实践，我们常常可以看到一个项目有很多的模块，例如api，dao，service，web等等。 Maven项目在默认情况下生成的构件，其名称不会是基于artifactId，version和packaging生成的，例如api-1.0.jar，dao-1.0.jar等 等，他们不会带有groupId的信息，这会造成一个问题，例如当我们把所有这些构件放到Web容器下的时候，你会发现项目dog有api- 1.0.jar，项目cat也有api-1.0.jar，这就造成了冲突。更坏的情况是，dog项目有api-1.0.jar，cat项目有api- 2.0.jar，其实两者没什么关系，可当放在一起的时候，却很容易让人混淆。为了让坐标更加清晰，又出现了一个原则，即*在定义artiafctId时也加入项目的信息*， 例如dog项目的api模块，那就使用artifactId dog-api，其他就是dog-dao，dao-service

等等。虽然连字号是不允许出现在Java的包名中的，但Maven没这个限制。现在 dog-api-1.0.jar，cat-2.0.jar 被放在一起时，就不容易混淆了。

关于坐标，我们还没谈到version，这里不再详述因为读者可以从Maven: The Complete Guide中找到详细的解释，简言之就是使用这样一个格式：

```
1 <主版本>.<次版本>.<增量版本>-<限定符>
```

其中主版本主要表示大型架构变更，次版本主要表示特性的增加，增量版本主要服务于bug修复，而限定符如alpha、beta等等是用来表示里程 碑。当然不是每个项目的版本都要用到这些4个部分，根据需要选择性的使用即可。在此基础上Maven还引入了SNAPSHOT的概念，用来表示活动的开发 状态，由于不涉及坐标规划，这里不进行详述。不过有点要提醒的是，由于SNAPSHOT的存在，自己显式地在version中使用时间戳字符串其实没有必要。

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。制作classifier的方式多种多样，其中最重要的一种是使用Maven Assembly Plugin，感兴趣的读者可以进一步研究。

小结

本文是InfoQ Maven专栏的第一篇，讨论的是Maven坐标的规划，包括如何正确的使用groupId、artifactId、version，以及 classfier。笔者在维护Maven中央仓库的工作过程中遇到过各种各样模糊的甚至是错误的坐标，它们的存在给广大Maven用户带来极大的不 便。本文抛出一些较好的实践，帮助大家更好的使用Maven。如果读者有相关的经验总结，也请不吝分享。