InfoQ Maven专栏（六）——Gradle，构建工具的未来？

April 7th, 2011

Maven面临的挑战

软件行业新旧交替的速度之快往往令人咂舌，不用多少时间，你就会发现曾经大红大紫的技术已经成为了明日黄花，当然，Maven也不会例外。虽然目前它基本上是Java构建的事实标准，但我们也能看到新兴的工具在涌现，比如基于Goovy的Gradle，而去年Hibernate宣布从Maven迁移至Gradle这一事件更是吸引了不少眼球。在此之前，我也听到了不少对Maven的抱怨，包括XML的繁冗，不够灵活，学习曲线陡峭等等。那Gradle是否能够在继承 Maven优点的基础上，克服这些缺点呢？带着这个疑问，我开始阅读Gradle的文档并尝试着将一个基于Maven的项目转成用Gradle构建，本文所要讲述大概就是这样的一个体验。需要注意的是，本文完全是基于Maven的角度来看Gradle的，因此对于Ant用户来说，视角肯定会大有不同。

Gradle初体验

Gradle的安装非常方便，下载ZIP包，解压到本地目录，设置 GRADLE_HOME 环境变量并将 GRADLE_HOME/bin 加到 PATH 环境变量中，安装就完成了。用户可以运行**gradle -v**命令验证安装，这些初始的步骤和Maven没什么两样。Gradle目前的版本是1.0-milestone-1，根据其Wiki上的Roadmap，在1.0正式版发布之前，还至少会有3个里程碑版本，而1.0的发布日期最快也不会早于6月份。而正是这样一个看起来似乎还不怎么成熟的项目，却有着让很多成熟项目都汗颜的文档，其包括了安装指南、基本教程、以及一份近300页的全面用户指南。这对于用户来说是非常友好的，同时也说明了Gradle的开发者对这个项目非常有信心，要知道编写并维护文档可不是件轻松的工作，对于Gradle这样未来仍可能发生很大变动的项目来说尤为如此。

类似于Maven的pom.xml文件，每个Gradle项目都需要有一个对应的build.gradle文件，该文件定义一些任务（task）来完成构建工作，当然，每个任务是可配置的，任务之间也可以依赖，用户亦能配置缺省任务，就像这样：

```
defaultTasks 'taskB'

task taskA << {
    println "i'm task A"
}

task taskB << {
    println "i'm task B, and I depend on " + taskA.name
}

taskB.dependsOn taskA
```

运行命令**$ gradle -q**之后（参数q让Gradle不要打印错误之外的日志），就能看到如下的预期输出：

```
i'm task A
i'm task B, and I depend on taskA
```

这不是和Ant如出一辙么？的确是这样，这种“任务”的概念与用法与Ant及其相似。Ant任务是Gradle世界的第一公民，Gradle对Ant做了很好的集成。除此之外，由于Gradle使用的Groovy脚本较XML更为灵活，因此，即使我自己不是Ant用户，我也仍然觉得Ant用户会喜欢上Gradle。

依赖管理和集成Maven仓库

我们知道依赖管理、仓库、约定优于配置等概念是Maven的核心内容，抛开其实现是否最优不谈，概念本身没什么问题，并且已经被广泛学习和接受。那Gradle实现了这些优秀概念了么？答案是肯定的。

先看依赖管理，我有一个简单的项目依赖于一些第三方类库包括SpringFramework、JUnit、Kaptcha等等。原来的Maven POM配置大概是这样的（篇幅关系，省略了部分父POM配置）：

```
    <properties>
        <kaptcha.version>2.3</kaptcha.version>
    </properties>

    <dependencies>
```

```
        <dependency>
            <groupId>com.google.code.kaptcha</groupId>
            <artifactId>kaptcha</artifactId>
            <version>${kaptcha.version}</version>
            <classifier>jdk15</classifier>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-beans</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
        </dependency>
    </dependencies>
```

然后我将其转换成Gradle脚本，结果是惊人的：

```
dependencies {
    compile('org.springframework:spring-core:2.5.6')
    compile('org.springframework:spring-beans:2.5.6')
    compile('org.springframework:spring-context:2.5.6')
    compile('com.google.code.kaptcha:kaptcha:2.3:jdk15')
    testCompile('junit:junit:4.7')
}
```

注意配置从原来的28行缩减至7行！这还不算我省略的一些父POM配置。依赖的groupId、artifactId、version，scope甚至是classfier，一点都不少。较之于Maven或者Ant的XML配置脚本，Gradle使用的Groovy脚本杀伤力太大了，爱美之心，人皆有之，相比于七旬老妇松松垮垮的皱纹，大家肯定都喜欢少女紧致的脸蛋，XML就是那老妇的皱纹。

关于Gradle的依赖管理起初我有一点担心，就是它是否有传递性依赖的机制呢？经过文档阅读和实际试验后，这个疑虑打消了，Gradle能够解析现有的Maven POM或者Ivy的XML配置，从而得到传递性依赖的信息，并且引入到当前项目中，这实在是一个聪明的做法。在此基础上，它也支持排除传递性依赖或者干脆关闭传递性依赖，其中第二点是Maven所不具备的特性。

自动化依赖管理的基石是仓库，Maven中央仓库已经成为了Java开发者不可或缺的资源，Gradle既然有依赖管理，那必然也得用到仓库，这当然也包括了Maven中央仓库，就像这样：

```
repositories {
    mavenLocal()
    mavenCentral()
    mavenRepo urls: "http://repository.sonatype.org/content/groups/forge/"
}
```

这段代码几乎不用解释，就是在Gradle中配置使用Maven本地仓库、中央仓库、以及自定义地址仓库。在我实际构建项目的时候，能看到终端打印的下载信息，下载后的文件被存储在 USER_HOME/.gradle/cache/ 目录下供项目使用，这种实现的方法与Maven又是及其类似了，可以说Gradle不仅最大限度的继承Maven的很多理念，仓库资源也是直接拿来用。

Gradle项目使用Maven项目生成的资源已经不是个问题了，接着需要反过来考虑，Maven用户是否能够使用Gradle生成的资源呢？或者更简单点问，Gradle项目生成的构件是否可以发布到Maven仓库中供人使用呢？这一点非常重要，因为如果做不到这一点，你可能就会丢失大量的用户。幸运的是Gradle再次给出了令人满意的答案。使用Gradle的Maven Plugin，用户就可以轻松地将项目构件上传到Maven仓库中：

```
apply plugin: 'maven'
```

```
...
uploadArchives {
    repositories.mavenDeployer {
        repository(url: "http://localhost:8088/nexus/content/repositories/snapshots/") {
            authentication(userName: "admin", password: "admin123")
            pom.groupId = "com.juvenxu"
            pom.artifactId = "account-captcha"
        }
    }
}
```

在上传的过程中，Gradle能够基于build.gradle生成对应的Maven POM文件，用户可以自行配置POM信息，比如这里的groupId和artifactId，而诸如依赖配置这样的内容，Gradle是会自动帮你进行转换的。由于Maven项目之间依赖交互的直接途径就是仓库，而Gradle既能够使用Maven仓库，也能以Maven的格式将自己的内容发布到仓库中，因此从技术角度来说，即使在一个基于Maven的大环境中，局部使用Gradle也几乎不会是一个问题。

约定优于配置

如同Ant一般，Gradle给了用户足够的自由去定义自己的任务，不过同时Gradle也提供了类似Maven的约定由于配置方式，这是通过Gradle的Java Plugin实现的，从文档上看，Gradle是推荐这种方式的。Java Plugin定义了与Maven完全一致的项目布局：

- src/main/java
- src/main/resources
- src/test/java
- src/test/resources

区别在于，使用Groovy自定义项目布局更加的方便：

```
sourceSets {
    main {
        java {
            srcDir 'src/java'
        }
        resources {
            srcDir 'src/resources'
        }
    }
}
```

Gradle Java Plugin也定义了构建生命周期，包括编译主代码、处理资源、编译测试代码、执行测试、上传归档等等任务：

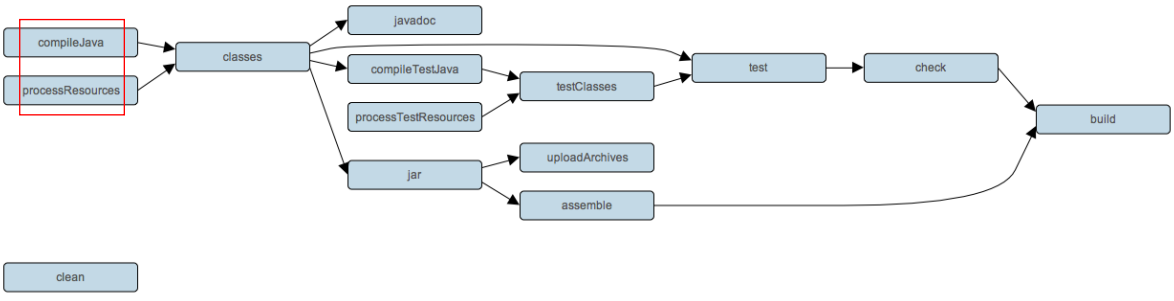

Figure 1. Gradle的构建生命周期

相对于Maven完全线性的生命周期，Gradle的构建生命周期略微复杂，不过也更为灵活，例如jar这个任务是用来打包的，它不像Maven那样依赖于执行测试的test任务，类似的，从图中可以看到，一个最终的build任务也没有依赖于uploadArchives任务。这个生命周期并没有将用户限制得很死，举个例子，我希望每次build都发布SNAPSHOT版本到Maven仓库中，而且我只想使用最简单的**$ gradle clean build**命令，那只需要添加一行任务依赖配置即可：

```
build.dependsOn 'uploadArchives'
```

由于Gradle完全是基于灵活的任务模型，因此很多事情包括覆盖现有任务，跳过任务都非常易于实现。而这些事情，在Maven的世界中，实现起来就比较的麻烦，或者说Maven压根就不希望用户这么做。

小结

一番体验下来，Gradle给我最大的感觉是两点。其一是简洁，基于Groovy的紧凑脚本实在让人爱不释手，在表述意图方面也没有什么不清晰的地方。其二是灵活，各种在Maven中难以下手的事情，在Gradle就是小菜一碟。比如修改现有的构建生命周期，几行配置就完成了，同样的事情，在Maven中你必须编写一个插件，那对于一个刚入门的用户来说，没个一两天几乎是不可能完成的任务。

不过即使如此，Gradle在未来能否取代Maven，在我看来也还是个未知数。它的一大障碍就是Grovvy，几乎所有Java开发者都熟悉XML，可又有几个人了解Groovy呢？学习成本这道坎是很难跨越的，很多人抵制Maven就是因为学起来不容易，你现在让因为一个构建工具学习一门新语言（即使这门语言和Java非常接近），那得到冷淡的回复几乎是必然的事情。Gradle的另外一个问题就是它太灵活了，虽然它支持约定优于配置，不过从本文你也看到了，破坏约定是多么容易的事情。人都喜欢自由，爱自定义，觉得自己的需求是多么的特别，可事实上，从Maven的流行来看，几乎95%以上的情况你不需要自行扩展，如果你这么做了，只会让构建变得难以理解。从这个角度来看，自由是把双刃剑，Gradle给了你足够的自由，约定优于配置只是它的一个选项而已，这初看起来很诱人，却也可能使其重蹈Ant的覆辙。Maven在Ant的基础上引入了依赖管理、仓库以及约定优于配置等概念，是一个很大的进步，不过在我现在看来，Gradle并没有引入新的概念，给我感觉它是一个结合Ant和Maven理念的优秀实现。

如果你了解Groovy，也理解Maven的约定优于配置，那试试Gradle倒也不错，尤其是它几乎能和现有的Maven系统无缝集成，而且你也能享受到简洁带来的极大乐趣。其实说到简洁，也许在不久的将来Maven用户也能直接享受到，Polyglot Maven在这方面已经做了不少工作。本文完全基于Maven的视角介绍Gradle这一构建工具的新秀，不过限于篇幅原因，无法深入Gradle的方方面面，例如Gradle也支持多模块构建，它提供了GUI操作界面，支持Grovvy（理所当然）和Scala项目等等。有兴趣的读者可以自行进一步了解。