Gradle入门系列（3）：依赖管理

2015-07-06 安卓应用频道

(点击上方蓝字，可快速关注我们)

在现实生活中，要创造一个没有任何外部依赖的应用程序并非不可能，但也是极具挑战的。这也是为什么依赖管理对于每个软件项目都是至关重要的一部分。

这篇教程主要讲述如何使用Gradle管理我们项目的依赖，我们会学习配置应用仓库以及所需的依赖，我们也会理论联系实际，实现一个简单的演示程序。

让我们开始吧。

仓库管理简介

本质上说，仓库是一种存放依赖的容器，每一个项目都具备一个或多个仓库。

Gradle支持以下仓库格式：

- Ivy仓库
- Maven仓库
- Flat directory仓库

我们来看一下，对于每一种仓库类型，我们在构建中应该如何配置。

在构建中加入Ivy仓库

我们可以通过URL地址或本地文件系统地址，将Ivy仓库加入到我们的构建中。

如果想通过URL地址添加一个Ivy仓库，我们可以将以下代码片段加入到build.gradle文件中：

```
repositories {
    ivy {
        url "http://ivy.petrikainulainen.net/repo"
    }
}
```

如果想通过本地文件系统地址添加一个Ivy仓库，我们可以将以下代码片段加入到build.gradle文件中：

```
repositories {
    ivy {
        url "../ivy-repo"
    }
}
```

小贴士：如果你想要获得更多关于Ivy仓库配置的信息，你可以参考以下资源：

- Section 50.6.6 Ivy Repositories of the Gradle User Guide
- The API documentation of the IvyArtifactRepository

我们继续，下面是如何在构建中加入Maven仓库。

在构建中加入Maven仓库

与Ivy仓库很类似，我们可以通过URL地址或本地文件系统地址，将Maven仓库加入到我们的构建中。

如果想通过URL地址添加一个Maven仓库，我们可以将以下代码片段加入到build.gradle文件中：

```
repositories {
    maven {
        url "http://maven.petrikainulainen.net/repo"
    }
}
```

如果想通过本地文件系统地址添加一个Maven仓库，我们可以将以下代码片段加入到build.gradle文件中：

```
repositories {
    maven {
        url "../maven-repo"
    }
}
```

在加入Maven仓库时，Gradle提供了三种“别名”供我们使用，它们分别是：

1. mavenCentral()别名，表示依赖是从Central Maven 2 仓库中获取的。
2. jcenter()别名，表示依赖是从Bintary’s JCenter Maven 仓库中获取的。
3. mavenLocal()别名，表示依赖是从本地的Maven仓库中获取的。

如果我们想要将Central Maven 2 仓库加入到构建中，我们必须在build.gradle文件中加入以下代码片段：

```
repositories {
    mavenCentral()
}
```

小贴士：如果你想要获取更多关于Maven仓库配置的信息，你可以参考这篇文章：

section 50.6.4 Maven Repositories of the Gradle User Guide

我们继续，下面是如何在构建中加入Flat Directory仓库。

在构建中加入Flat Directory仓库

如果我们想要使用Flat Directory仓库，我们需要将以下代码片段加入到build.gradle文件中：

```
repositories {
    flatDir {
        dirs 'lib'
    }
}
```

这意味着系统将在lib目录下搜索依赖，同样的，如果你愿意的话可以加入多个目录，代码片段如下：

```
repositories {
    flatDir {
        dirs 'libA', 'libB'
    }
}
```

小贴士：如果你想要获得更多关于Flat Directory仓库配置的信息，你可以参考以下资源：

- Section 50.6.5 Flat directory repository of the Gradle User Guide
- Flat Dir Repository post to the gradle-user mailing list

我们继续，下面要讲的是，如何使用Gradle管理项目中的依赖。

依赖管理简介

在配置完项目仓库后，我们可以声明其中的依赖，如果我们想要声明一个新的依赖，可以采用如下步骤：

1. 指定依赖的配置。
2. 声明所需的依赖。

让我们看一下详细步骤：

配置中的依赖分类

在Gradle中，依赖是按照指定名称进行分类的，这些分类被称为配置项，我们可以使用配置项声明项目的外部依赖。

Java插件指定了若干依赖配置项，其描述如下：

- 当项目的源代码被编译时，compile配置项中的依赖是必须的。
- runtime配置项中包含的依赖在运行时是必须的。
- testCompile配置项中包含的依赖在编译项目的测试代码时是必须的。
- testRuntime配置项中包含的依赖在运行测试代码时是必须的。
- archives配置项中包含项目生成的文件（如Jar文件）。
- default配置项中包含运行时必须的依赖。

我们继续，下面是如何在项目中声明依赖。

声明项目依赖

最普遍的依赖称为外部依赖，这些依赖存放在外部仓库中。一个外部依赖可以由以下属性指定：

- group属性指定依赖的分组（在Maven中，就是groupId）。

- name属性指定依赖的名称（在Maven中，就是artifactId）。
- vertion属性指定外部依赖的版本（在Maven中，就是version）。

小贴士：这些属性在Maven仓库中是必须的，如果你使用其他仓库，一些属性可能是可选的。打个比方，如果你使用Flat directory仓库，你可能只需要指定名称和版本。

我们假设我们需要指定以下依赖：

- 依赖的分组是foo。
- 依赖的名称是foo。
- 依赖的版本是0.1。
- 在项目编译时需要这些依赖。

我们可以将以下代码片段加入到build.gradle中，进行依赖声明：

```
dependencies {
    compile group: 'foo', name: 'foo', version: '0.1'
}
```

我们也可以采用一种快捷方式声明依赖：[group]:[name]:[version]。如果我们想用这种方式，我们可以将以下代码段加入到build.gradle中：

```
dependencies {
    compile 'foo:foo:0.1'
}
```

我们也可以在同一个配置项中加入多个依赖，传统的方式如下：

```
dependencies {
    compile (
        [group: 'foo', name: 'foo', version: '0.1'],
        [group: 'bar', name: 'bar', version: '0.1']
    )
}
```

如果采用快捷方式，那可以是这样：

```
dependencies {
    compile 'foo:foo:0.1', 'bar:bar:0.1'
```

```
}
```

自然地，声明属于不同配置项的依赖也是可以的。比如说，如果我们想要声明属于compile和testCompile配置项的依赖，可以这么做：

```
dependencies {
    compile group: 'foo', name: 'foo', version: '0.1'
    testCompile group: 'test', name: 'test', version: '0.1'
}
```

同样的，给力的快捷方式又来了(￣﹀￣)

```
dependencies {
    compile 'foo:foo:0.1'
    testCompile 'test:test:0.1'
}
```

小贴士：你可以从这篇文章中获得更多关于依赖声明的信息。

我们已经学习了依赖管理的基础知识，下面我们来实现一个演示程序。

创建演示程序

演示程序的需求是这样的：

- 演示程序的构建脚本必须使用Maven central仓库。
- 演示程序必须使用Log4j写入日志。
- 演示程序必须包含包含单元测试，保证正确的信息返回，单元测试必须使用JUnit编写。
- 演示程序必须创建一个可执行的Jar文件。

我们来看一下怎样实现这些需求。

配置仓库

我们的演示程序的一个需求是构建脚本必须使用Maven central仓库，在我们使用Maven central仓库配置构建脚本后，源代码如下：

```
apply plugin: 'java'
```

```
repositories {
    mavenCentral()
}

jar {
    manifest {
        attributes 'Main-Class': 'net.petrikainulainen.gradle.HelloWorld'
    }
}
```

再来看一下如何对我们的演示程序进行依赖声明。

依赖声明

在build.gradle文件中，我们声明了两个依赖：

- Log4j（版本1.2.17）用来记录日志。
- JUnit（版本4.11）用来编写单元测试。

在我们声明了这些依赖后，build.gradle文件是这样的：

```
apply plugin: 'java'

repositories {
    mavenCentral()
}

dependencies {
    compile 'log4j:log4j:1.2.17'
    testCompile 'junit:junit:4.11'
}

jar {
    manifest {
        attributes 'Main-Class': 'net.petrikainulainen.gradle.HelloWorld'
    }
}
```

我们继续，稍微加入一些代码。

编写代码

为了实现我们演示程序的需求，“我们不得不过度工程化一下”，我们会按照下列步骤创建程序：

1. 创建一个MessageService类，当其中的getMessage()方法被调用时，返回字符串“Hello World!”。
2. 创建一个MessageServiceTest类，确保MessageService类中的getMessage()方法返回字符串“Hello World!”。
3. 创建程序的主类，从MessageService对象获取信息，并使用Log4j写入日志。
4. 配置Log4j。

我们按部就班的操作一下。

首先，在src/main/java/net/petrikainulainen/gradle目录下新建一个MessageService类并加以实现，代码如下：

```
package net.petrikainulainen.gradle;

public class MessageService {

    public String getMessage() {
        return "Hello World!";
    }
}
```

其次，在src/main/test/net/petrikainulainen/gradle目录下新建一个MessageServiceTest类，编写单元测试，代码如下：

```
package net.petrikainulainen.gradle;

import org.junit.Before;
import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class MessageServiceTest {
```

```
    private MessageService messageService;

    @Before
    public void setUp() {
        messageService = new MessageService();
    }

    @Test
    public void getMessage_ShouldReturnMessage() {
        assertEquals("Hello World!", messageService.getMessage());
    }
}
```

第三，在src/main/java/net/petrikainulainen/gradle目录下新建一个HelloWorld类，这是程序的主类，从MessageService对象获取信息，并使用Log4j写入日志，代码如下：

```
package net.petrikainulainen.gradle;

import org.apache.log4j.Logger;

public class HelloWorld {

    private static final Logger LOGGER = Logger.getLogger(HelloWorld.class);

    public static void main(String[] args) {
        MessageService messageService = new MessageService();

        String message = messageService.getMessage();
        LOGGER.info("Received message: " + message);
    }
}
```

第四，在src/main/resources目录中，使用log4j.properties配置log4j，log4j.properties文件如下：

```
log4j.appender.Stdout=org.apache.log4j.ConsoleAppender
log4j.appender.Stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.Stdout.layout.conversionPattern=%-5p - %-26.26c{1} - %m\n
```

```
log4j.rootLogger=DEBUG,Stdout
```

这样就好了，我们看看如何执行测试。

执行测试

我们可以通过以下命令执行测试。

```
gradle test
```

当测试通过时，我们能看到如下输出：

```
> gradle test
:compileJava
:processResources
:classes
:compileTestJava
:processTestResources
:testClasses
:test

BUILD SUCCESSFUL

Total time: 4.678 secs
```

然而，如果测试失败，我们将看到如下输出：

```
> gradle test
:compileJava
:processResources
:classes
:compileTestJava
:processTestResources
:testClasses
:test

net.petrikainulainen.gradle.MessageServiceTest >
getMessage_ShouldReturnMessageFAILED
    org.junit.ComparisonFailure at MessageServiceTest.java:22
```

```
1 test completed, 1 failed
:test FAILED

FAILURE: Build failed with an exception.

* What went wrong:
Execution failed for task ':test'.
> There were failing tests. See the report at: file:///Users/loke/Projects/Java/Blog/gradle-examples/dependency-management/build/reports/tests/index.html

* Try:
Run with --stacktrace option to get the stack trace. Run with --info or --debug option to get more log output.

BUILD FAILED

Total time: 4.461 secs
```

正如我们所看到的，如果单元测试失败了，输出信息中将描述以下信息：

- 哪一个测试失败了。
- 执行了几个测试，其中几个失败了。
- 测试报告的位置，测试报告提供了失败（或成功）的测试的额外信息。

当我们执行单元测试时，Gradle会在相应目录创建测试报告：

- build/test-results目录包含每次测试执行的原始数据。
- build/reports/tests目录包含一个HTML报告，描述了测试的结果。

HTML测试报告是一个非常有用的工具，因为它描述了测试失败的原因。比如说，如果我们的单元测试认为MessageService类中的getMessage()方法返回字符串“Hello Worl1d!”，那么HTML报告看上去就像下图一样：

我们继续，下面是如何打包和运行我们的演示程序。

打包和运行程序

我们能够可以使用以下任意一种命令打包程序：gradle assembly或gradle build，这两个命令都会在build/libs目录中创建dependency-management.jar文件。

当我们使用java -jar dependency-management.jar命令运行演示程序时，我们可以看到如下输出：

```
> java -jar dependency-management.jar

Exception in thread "main" java.lang.NoClassDefFoundError: org/apache/log4j/Logger
    at net.petrikainulainen.gradle.HelloWorld.<clinit>(HelloWorld.java:10)
Caused by: java.lang.ClassNotFoundException: org.apache.log4j.Logger
    at java.net.URLClassLoader$1.run(URLClassLoader.java:372)
    at java.net.URLClassLoader$1.run(URLClassLoader.java:361)
    at java.security.AccessController.doPrivileged(Native Method)
    at java.net.URLClassLoader.findClass(URLClassLoader.java:360)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:424)
    at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:308)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:357)
    ... 1 more
```

抛出异常的原因是，当我们运行程序时，Log4j的依赖在classpath中没有找到。

解决这个问题最简单的方式是创建一个所谓的“胖”Jar文件，即把所有程序运行所需的依赖都打包到Jar文件中去。

通过查阅Gradle Cookbook中的教程，可以修改构建脚本，如下：

```
apply plugin: 'java'

repositories {
    mavenCentral()
}

dependencies {
    compile 'log4j:log4j:1.2.17'
    testCompile 'junit:junit:4.11'
}

jar {
    from { configurations.compile.collect { it.isDirectory() ? it : zipTree(it) } }
    manifest {
        attributes 'Main-Class': 'net.petrikainulainen.gradle.HelloWorld'
    }
}
```

现在，我们可以运行演示程序了（打包后），一切正常：

```
> java -jar dependency-management.jar
INFO  - HelloWorld                - Received message: Hello World!
```

这些就是今天的内容了，我们总结一下学到了什么。

总结

这篇教程教会了我们四个方面的内容：

我们学会了配置构建所需的仓库。
我们学会了如何声明所需的依赖以及依赖的分类（分组）。
我们了解了Gradle会在测试执行后创建一个HTML测试报告。

我们学会了创建一个所谓的“胖”Jar文件。

来源：伯乐在线 - Justin Wu

链接：http://android.jobbole.com/72992/

「安卓应用频道」专注 Android 技术、设计和市场推广相关的内容分享。如果你期望了解Android应用的全流程知识，欢迎关注我们。

微信号： **AndroidPD**

（长按上图↑可自动识别二维码）

阅读原文

微信扫一扫
关注该公众号