Gradle入门系列（5）：创建多项目构建

原创 2015-11-23 伯乐在线 安卓应用频道

本系列：
Gradle入门系列（1）：简介
Gradle入门系列（2）：第一个Java项目
Gradle入门系列（3）：依赖管理
Gradle入门系列（4）：创建二进制发布版本

尽管我们可以仅使用单个组件来创建可工作的应用程序，但有时候更广泛的做法是将应用程序划分为多个更小的模块。

由于这是一个非常普通的案例，因此每个成熟的构建工具都必须支持这项功能，Gradle也不例外。倘若Gradle项目拥有多于一个组件，我们就将其称之为多项目构建（multi-project build）。

这篇教程描述了如何使用Gradle创建一个多项目构建。

我们先来看一看Gradle构建的一些需求。
扩展阅读：如果你对Gradle不太熟悉，在阅读这篇教程前你应该先阅读以下文章。

- 《Gradle入门系列（1）：简介》帮助你安装Gradle，描述Gradle构建的基本概念，以及如何使用Gradle插件在构建中添加功能。
- 《Gradle入门系列（2）：第一个Java项目》描述如何使用Gradle创建Java项目，并将应用程序打包为可执行jar文件。
- 《Gradle入门系列（3）：依赖管理》描述如何在Gradle项目中管理依赖。

Gradle Build 的需求

我们的示例程序拥有两个模块：

core模块包含一些通用的组件，它们能够被程序的其他模块使用。在我们的例子上，只包含一个类：MessageService类返回‘Hello World!’字符串。该模块只有一个依赖：它包含一个单元测试，使用Junit 4.11。

app模块包含HelloWorld类，是程序的开端，它从MessageService对象中获取信息，并将接收到的信息写入一个日志文件中。该模块拥有两个依赖：它需要core模块，还使用Log4j 1.2.17作为日志库。

我们的Gradle构建还有其他两个需求：

- 我们必须要使用Gradle运行程序。
- 我们必须要创建一个可运行的二进制发布，而且不能使用所谓的“fat jar”方式。

如果你还不清楚怎样使用Gradle运行程序，以及创建可运行的二进制发布，在阅读这篇教程前，你应该先阅读以下文章。

《Gradle入门系列（4）：创建二进制发布版本》

我们继续来探讨一下如何创建一个多项目构建来满足我们的需求。

创建一个多项目构建

下一步，我们将创建一个多项目的Gradle构建，包括两个子项目：app 和 core。初始阶段，先要建立Gradle构建的目录结构。

建立目录结构

由于core和app模块都使用Java语言，而且它们都使用Java项目的默认项目布局，我们根据以下步骤建立正确的目录结构：

1. 建立core模块的根目录(core)，并建立以下子目录：
   - src/main/java目录包含core模块的源代码。
   - src/test/java目录包含core模块的单元测试。

2. 建立app模块的根目录(app)，并建立以下子目录：
   - src/main/java目录包含app模块的源代码。
   - src/main/resources目录包含app模块的资源文件。

现在，我们已经创建了所需的目录，下一步是配置Gradle构建，先对包含在多项目构建中的项目进行配置。

对包含在多项目构建中的项目进行配置

我们可以通过以下步骤，对包含在多项目构建中的项目进行配置：

1. 在根项目的根目录下创建settings.gradle文件，一个多项目Gradle构建必须含有这个文件，因为它指明了那些包含在多项目构建中的项目。

2. 确保app和core项目包含在我们的多项目构建中。

我们的settings.gradle文件如下：

```
include 'app'
include 'core'
```

扩展阅读：

- Gradle User Guide: 56.2 Settings file
- Gradle DSL Reference: Settings

配置 core 项目

我们可以通过以下步骤对core项目进行配置：

1. 在core项目的根目录下创建build.gradle文件。
2. 使用Java插件创建一个Java项目。
3. 确保core项目从Maven2中央仓库(central Maven2 repository)中获取依赖。
4. 声明JUnit依赖(版本4.11)，并使用testCompile配置项，该配置项表明：core项目在它的单元测试被编译前，需要JUnit库。

core项目的build.gradle文件如下：

```
apply plugin: 'java'

repositories {
  mavenCentral()
}

dependencies {
  testCompile 'junit:junit:4.11'
}
```

扩展阅读：

- 《Gradle入门系列（2）：第一个Java项目》
- 《Gradle入门系列（3）：依赖管理》

我们继续对app项目进行配置。

配置App项目

在配置app项目之前，我们先来快速浏览一下对一些特殊依赖的依赖管理，这些依赖都是同一个多项目构建的一部分，我们将其称之为”项目依赖“。

如果多项目构建拥有项目A和B，同时，项目B的编译需要项目A，我们可以通过在项目B的build.gradle文件中添加以下依赖声明来进行依赖配置。

```
dependencies {
  compile project(':A')
}
```

扩展阅读

- Gradle User Guide: 51.4.3. Project dependencies
- Gradle User Guide: 57.7. Project lib dependencies

现在，我们可以按照以下步骤配置app项目：

1. 在app项目的根目录下创建build.gradle文件。
2. 用Java插件创建一个Java项目。
3. 确保app项目从Maven2中央仓库(central Maven2 repository)中获取依赖。
4. 配置所需的依赖，app项目在编译时需要两个依赖：
   - Log4j (version 1.2.17)
   - core 模块
5. 创建二进制发布版本

app项目的build.gradle文件如下：

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

```
repositories {
  mavenCentral()
}

dependencies {
  compile 'log4j:log4j:1.2.17'
  compile project(':core')
}

mainClassName = 'net.petrikainulainen.gradle.client.HelloWorld'

task copyLicense {
  outputs.file new File("$buildDir/LICENSE")
  doLast {
     copy {
        from "LICENSE"
        into "$buildDir"
     }
  }
}

applicationDistribution.from(copyLicense) {
  into ""
}
```

我们继续，下面是移除core和app项目的构建脚本中的重复配置。

移除重复配置

当我们对多项目构建中的子项目进行配置时，我们在core和app项目的构建脚本中添加了重复的配置。

由于两个项目都是Java项目，因此它们都使用Java插件。
两个项目都使用Maven2中央仓库(central Maven2 repository)。
换句话说，两个构建脚本都包含以下配置：

```
apply plugin: 'java'

repositories {
```

```
  mavenCentral()
}
```

让我们将这项配置转移到根项目的build.gradle文件中，在此之前，我们必须先学习一下如何在根项目的build.gradle文件中配置子项目。

如果我们想要在一个称为core的子项目中添加配置，那么就必须在根项目的build.gradle文件中添加以下片段：

```
project(':core') {
  //Add core specific configuration here
}
```

换句话说，如果想要将重复的配置转移到根项目的构建脚本中，就必须将以下配置添加到build.gradle文件中：

```
project(':app') {
  apply plugin: 'java'

  repositories {
     mavenCentral()
  }
}

project(':core') {
  apply plugin: 'java'

  repositories {
     mavenCentral()
  }
}
```

不过这种做法在实质上并没有改变什么，在构建脚本中依然还存在重复配置，唯一的区别是重复配置现在转移到了根项目的build.gradle文件中。让我们来消灭这些重复配置。

如果我们想要在根项目的子项目中添加通用的配置，需要将以下片段添加到根项目的build.gradle文件中：

```
subprojects {
```

```
  //Add common configuration here
}
```

在根项目的build.gradle文件中移除了重复配置后，代码如下：

```
subprojects {
  apply plugin: 'java'

  repositories {
     mavenCentral()
  }
}
```

如果我们的配置项是被多项目构建中的所有项目所共享的，那么需要在根项目的build.gradle文件中添加以下片段：

```
allprojects {
  //Add configuration here
}
```

Additional Reading:

- Gradle User Guide: 57.1 Cross project configuration
- Gradle User Guide: 57.2 Subproject configuration

现在，我们可以从子项目的构建脚本中移除重复配置，子项目新的构建脚本如下：

core/build.gradle文件如下：

```
dependencies {
    testCompile 'junit:junit:4.11'
}
```

app/build.gradle文件如下：

```
apply plugin: 'application'

dependencies {
  compile 'log4j:log4j:1.2.17'
```

```
  compile project(':core')
}

mainClassName = 'net.petrikainulainen.gradle.client.HelloWorld'

task copyLicense {
  outputs.file new File("$buildDir/LICENSE")
  doLast {
     copy {
        from "LICENSE"
        into "$buildDir"
     }
  }
}

applicationDistribution.from(copyLicense) {
  into ""
}
```

现在，我们已经创建了一个多项目Gradle构建，下面让我们来直观的感受一下我们做了些什么。

我们刚刚做了什么？

在我们的多项目构建的根目录下执行命令gradle projects，可以看到如下输出：

```
> gradle projects
:projects

------------------------------------------------------------
Root project
------------------------------------------------------------

Root project 'multi-project-build'
+--- Project ':app'
--- Project ':core'

To see a list of the tasks of a project, run gradle <project-path>:tasks
For example, try running gradle :app:tasks
```

```
BUILD SUCCESSFUL
```

正如我们所看到的，这条命令列出了根项目的子项目(app和core)，这意味着我们刚才已经创建了一个多项目Gradle构建，它拥有两个子项目。

在我们的多项目构建的根目录下执行命令gradle tasks，可以看到如下输出（仅列出相关部分）：

```
> gradle tasks
:tasks

------------------------------------------------------------
All tasks runnable from root project
------------------------------------------------------------

Application tasks
-----------------
distTar - Bundles the project as a JVM application with libs and OS specific scripts.
distZip - Bundles the project as a JVM application with libs and OS specific scripts.
installApp -Installs the project as a JVM application along with libs and OS specific scripts
run - Runs this project as a JVM application
```

正如我们所看到的，我们可以使用Gradle运行程序，并创建一个可运行的二进制发布，它没有使用所谓的“fat jar”方式。这表明我们已经满足了本文中要求实现的Gradle构建的所有需求。

我们继续来看一下，从这篇教程中我们学到了什么。

总结

这篇教程教会了我们三部分内容：

1. 一个多项目构建必须在根项目的根目录下包含settings.gradle文件，因为它指明了那些包含在多项目构建中的项目。
2. 如果需要在多项目构建的所有项目中加入公用的配置或行为，我们可以将这项配置加入到根项目的build.gradle文件中(使用allprojects)。
3. 如果需要在根项目的子项目中加入公用的配置或行为，我们可以将这项配置加入到根项目的build.gradle文件中(使用subprojects)。

P.S. 你可以从Github上获取这篇教程的演示程序。

阅读原文