Gradle入门系列（1）：简介

Gradle是一种构建工具，它抛弃了基于XML的构建脚本，取而代之的是采用一种基于Groovy的内部领域特定语言。近期，Gradle获得了极大的关注，这也是我决定去研究Gradle的原因。

**这篇文章是Gradle教程的第一篇，我们有两个目标：**

* 1. 帮助我们学会安装Gradle。
* 2. 介绍一些基本概念，这有助于我们进一步理解本教程的后面章节。

我们开始吧，先看一下如何安装Gradle。

**安装Gradle**

如果我们使用的操作系统是Windows或Linux，我们可以根据以下步骤安装Gradle：  
1. 从[这个页面](http://www.gradle.org/downloads)下载二进制文件。  
2. 解压Zip文件，加入环境变量（在PATH中加入GRADLE\_HOME/bin目录）。

如果在安装过程中遇到问题，可以进一步查看官方的[安装指南](http://www.gradle.org/docs/current/userguide/installation.html)。  
如果我们使用的操作系统是OS X，我们可以使用Homebrew安装Gradle，在命令提示符中输入以下命令：

|  |  |
| --- | --- |
| 1 | brew install gradle |

我们可以验证一下Gradle是否工作正常，在命令提示符中执行命令*gradle -v*即可，如果Gradle工作正常，我们应该能看到以下输出结果（当然，Windows和Linux用户看到的结果会有细微差异）。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | > gradle -v    ------------------------------------------------------------  Gradle 1.12  ------------------------------------------------------------    Build time: 2014-04-29 09:24:31 UTC  Build number: none  Revision: a831fa866d46cbee94e61a09af15f9dd95987421    Groovy: 1.8.6  Ant: Apache Ant(TM) version 1.9.3 compiled on December 23 2013  Ivy: 2.2.0  JVM: 1.8.0 (Oracle Corporation 25.0-b70)  OS: Mac OS X 10.9.3 x86\_64 |

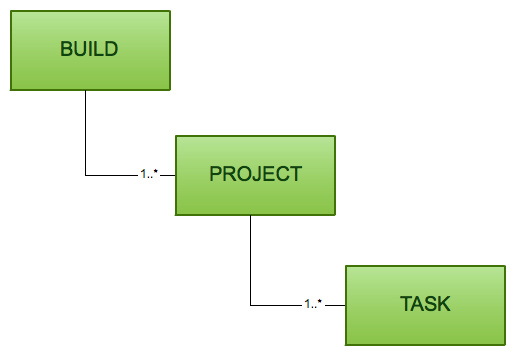
下面我们来快速浏览一下Gradle构建的基本概念。

**Gradle构建简介**

在Gradle中，有两个基本概念：**项目**和**任务**。请看以下详解：

* **项目**是指我们的构建产物（比如Jar包）或实施产物（将应用程序部署到生产环境）。**一个项目包含一个或多个任务。**
* **任务**是指不可分的最小工作单元，执行构建工作（比如编译项目或执行测试）。

那么，这些概念和Gradle的构建又有什么联系呢？好，**每一次Gradle的构建都包含一个或多个项目**。

下面这张图展示了上面所谈到的这些概念的关系。  


我们能够使用以下配置文件对Gradle的构建进行配置：

* Gradle构建脚本（build.gradle）指定了一个项目和它的任务。
* [Gradle属性文件（gradle.properties）](http://www.gradle.org/docs/current/userguide/build_environment.html#sec:gradle_configuration_properties)用来配置构建属性。
* [Gradle设置文件（gradle.settings）](http://www.gradle.org/docs/current/userguide/build_lifecycle.html#sec:settings_file)对于只有一个项目的构建而言是可选的，如果我们的构建中包含多于一个项目，那么它就是必须的，因为它描述了哪一个项目参与构建。每一个多项目的构建都必须在项目结构的根目录中加入一个设置文件。

你可以在[这篇文章](http://www.gradle.org/docs/current/userguide/tutorial_using_tasks.html)中获得更多关于Gradle构建脚本的信息。

我们继续，下面我们看一下如果使用Gradle插件为构建工作加入新功能。

**更简短的Gradle插件简介**

Gradle的设计理念是，所有有用的特性都由[Gradle插件](http://www.gradle.org/docs/current/userguide/plugins.html)提供，一个Gradle插件能够：

* 在项目中添加新任务
* 为新加入的任务提供默认配置，这个默认配置会在项目中注入新的约定（如源文件位置）。
* 加入新的属性，可以覆盖插件的默认配置属性。
* 为项目加入新的依赖。

Gradle用户手册提供了[一系列标准Gradle插件](http://www.gradle.org/docs/current/userguide/standard_plugins.html)。

在我们为项目加入Gradle插件时，我们可以根据名称或类型来指定Gradle插件。

我们可以将下面这行代码加入到build.gradle文件中，它通过名称指定Gradle插件（这里的名称是foo）

|  |  |
| --- | --- |
| 1 | apply plugin: 'foo' |

另一方面，我们也可以通过类型指定Gradle插件，将下面这行代码加入到build.gradle文件中（这里的类型是com.bar.foo）：

|  |  |
| --- | --- |
| 1 | apply plugin: 'com.bar.foo' |

你可以阅读[这篇文章](http://www.gradle.org/docs/current/userguide/plugins.html#sec:using_plugins)，掌握应用插件的更多信息。

今天就到这里，我们来总结一下我们所学的内容。

**总结**

这篇教程讲授了三部分内容：

* 我们学会了如何安装Gradle
* 我们理解了Gradle构建的一些基本知识
* 我们了解了如何使用Gradle插件为我们的构建工作增加功能。

Gradle入门系列（2）：第一个Java项目

这篇教程的主要内容是讲解如何用Gradle编译和打包一个简单的Java项目。

该Java项目只有一个需求：我们的构建脚本必须创建一个可执行的Jar文件，换句话说，我们必须能够使用命令*java -jar jarfile.jar*来运行我们的程序。我们来看一下如何满足这个需求。

**创建一个Java项目**

我们可以使用Java插件（译注：关于Gradle插件的定义，请查看[第一篇教程](http://blog.jobbole.com/71999/)）来创建一个Java项目，为了做到这点，我们需要把下面这段语句加入到*build.gradle*文件中：

|  |  |
| --- | --- |
| 1 | apply plugin: 'java' |

就是这样，现在我们已经创建了一个Java项目。Java插件会在我们的构建中添加一些新的约定（如默认的项目结构），新的任务，和新的属性。  
让我们来快速地看一下默认的项目结构。

**Java项目结构**

[默认的项目结构](http://www.gradle.org/docs/current/userguide/java_plugin.html#N12119)如下：

* *src/main/java*目录包含了项目的源代码。
* *src/main/resources*目录包含了项目的资源（如属性文件）。
* *src/test/java*目录包含了测试类。
* *src/test/resources*目录包含了测试资源。所有我们构建生成的文件都会在*build*目录下被创建，这个目录涵盖了以下的子目录，这些子目录我们会在这篇教程中提到，另外还有一些子目录我们会放在以后讲解。
* *classes*目录包含编译过的*.class*文件。
* *libs*目录包含构建生成的*jar*或*war*文件。

**为构建加入一个主类（main class）**

让我们创建一个简单的主类，在这个类中会打印一个“Hello world”然后*System.out*出来。这个HelloWorld类的源代码如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | package net.petrikainulainen.gradle;    public class HelloWorld {        public static void main(String[] args) {          System.out.println("Hello World!");      }  } |

*HelloWorld*类存放在*src/main/java/net/petrikainulainen/gradle*目录

这很好，然而，我们还需要编译和打包我们的项目，不是吗？我们先看一下这个Java工程中的任务。

**Java工程中的任务**

[Java插件在我们的构建中加入了很多任务](http://www.gradle.org/docs/current/userguide/java_plugin.html#N11EA7)，我们这篇教程涉及到的任务如下：

* *assemble*任务会编译程序中的源代码，并打包生成Jar文件，这个任务不执行单元测试。
* *build*任务会执行一个完整的项目构建。
* *clean*任务会删除构建目录。
* *compileJava*任务会编译程序中的源代码。

我们还可以执行以下命令得到一个可运行任务及其描述的完整列表

|  |  |
| --- | --- |
| 1 | gradle tasks |

这是一个很好的方式，不需要阅读构建脚本，就能对你的项目进行大致的浏览，如果我们在项目根目录下运行这个命令，我们可以看到以下输出：

|  |  |
| --- | --- |
|  | > gradle tasks  :tasks    ------------------------------------------------------------  All tasks runnable from root project  ------------------------------------------------------------    Build tasks  -----------  assemble - Assembles the outputs of this project.  build - Assembles and tests this project.  buildDependents - Assembles and tests this project and all projects that depend on it.  buildNeeded - Assembles and tests this project and all projects it depends on.  classes - Assembles classes 'main'.  clean - Deletes the build directory.  jar - Assembles a jar archive containing the main classes.  testClasses - Assembles classes 'test'.    Build Setup tasks  -----------------  init - Initializes a new Gradle build. [incubating]  wrapper - Generates Gradle wrapper files. [incubating]    Documentation tasks  -------------------  javadoc - Generates Javadoc API documentation for the main source code.    Help tasks  ----------  dependencies - Displays all dependencies declared in root project 'first-java-project'.  dependencyInsight - Displays the insight into a specific dependency in root project 'first-java-project'.  help - Displays a help message  projects - Displays the sub-projects of root project 'first-java-project'.  properties - Displays the properties of root project 'first-java-project'.  tasks - Displays the tasks runnable from root project 'first-java-project'.    Verification tasks  ------------------  check - Runs all checks.  test - Runs the unit tests.    Rules  -----  Pattern: build<ConfigurationName>: Assembles the artifacts of a configuration.  Pattern: upload<ConfigurationName>: Assembles and uploads the artifacts belonging to a configuration.  Pattern: clean<TaskName>: Cleans the output files of a task.    To see all tasks and more detail, run with --all.    BUILD SUCCESSFUL    Total time: 2.792 secs |

我们继续，下面要讲怎样打包我们的项目。

**打包我们的项目**

我们可以通过使用两个不同的任务来打包项目。  
如果我们在命令提示符中执行命令*gradle assemble*，我们可以看到以下输出：

|  |  |
| --- | --- |
|  | > gradle assemble  :compileJava  :processResources  :classes  :jar  :assemble    BUILD SUCCESSFUL    Total time: 3.163 secs |

如果我们在命令提示符中执行命令*gradle build*，我们可以看到以下输出：

|  |  |
| --- | --- |
|  | > gradle build  :compileJava  :processResources  :classes  :jar  :assemble  :compileTestJava  :processTestResources  :testClasses  :test  :check  :build    BUILD SUCCESSFUL    Total time: 3.01 secs |

这些命令的输出表明了它们的区别：

* *assemble*任务仅仅执行项目打包所必须的任务集。
* *build*任务执行项目打包所必须的任务集，以及执行自动化测试。这两个命令都会在*build/libs*目录中创建一个*file-java-project.jar*文件。默认创建的Jar文件名称是由这个模版决定的：*[projectname].jar*，此外，项目的默认名称和其所处的目录名称是一致的。因此如果你的项目目录名称是*first-java-project*，那么创建的Jar文件名称就是*first-java-project.jar。*

现在，我们尝试使用以下命令运行我们的程序：

|  |  |
| --- | --- |
| 1 | java -jar first-java-project.jar |

我们可以看到以下输出：

|  |  |
| --- | --- |
| 1  2 | > java -jar first-java.project.jar  No main manifest attribute, in first-java-project.jar |

问题出在，我们没有在*manifest*文件中配置Jar文件的主类，让我们继续看看怎样解决这个问题。

**配置Jar文件的主类**

Java插件在我们的项目中加入了一个Jar任务，每一个Jar对象都一个*manifest*属性，这个属性是[*Manifest*](http://www.gradle.org/docs/current/javadoc/org/gradle/api/java/archives/Manifest.html)的一个实例。

我们可以对生成的Jar文件的主类进行配置，[使用*Manifest*接口的*attributes()*方法](http://www.gradle.org/docs/current/javadoc/org/gradle/api/java/archives/Manifest.html#attributes%28java.util.Map%29)。换句话说，我们可以使用一个包含键值对的map结构指定加入到*manifest*文件的属性集。

我们能够通过设置*Main-Class*属性的值，指定我们程序的入口点。在我们对*build.gradle*文件进行必要的改动后，代码如下

|  |  |
| --- | --- |
|  | apply plugin: 'java'    jar {      manifest {          attributes 'Main-Class': 'net.petrikainulainen.gradle.HelloWorld'      }  } |

（[JavaSE教程](http://docs.oracle.com/javase/tutorial/deployment/jar/manifestindex.html)提供了关于*manifest*文件的更多信息。）  
在我们执行*gradle assemble*或*gradle build*命令生成一个新的jar文件之后，我们可以执行以下命令运行jar文件：

|  |  |
| --- | --- |
| 1 | java -jar first-java-project.jar |

当我们运行程序时，*System.out*会打印出以下信息：

|  |  |
| --- | --- |
| 1  2 | &gt; java -jar first-java-project.jar  Hello World! |

这就是我们今天所有的内容，我们看一下我们学到了什么。

**总结**

我们已经通过Gradle创建了一个简单的Java项目，这篇教程教会了我们四点：

* 我们了解了可以使用Gradle的Java插件创建一个Java项目。
* 我们知道了Java项目的默认结构和Maven项目的默认结构是一样的。
* 我们知道了构建所生成的所有文件都能在*build*目录下找到。
* 我们知道了我们可以自定义加入到*manifest*文件中的属性。

P.S. 这篇教程的示例代码可以在[Github](https://github.com/pkainulainen/gradle-examples/tree/master/first-java-project)找到。