链滴

学习Gradle 3 Java 快速入门

作者：Hassan

原文链接：https://ld246.com/article/1471569886393

来源网站：链滴

<h2>1、插件介绍</h2>
插件是对Gradle功能的扩展，Gradle有着丰富的插件，你可以在这里搜索相关插件（传送门）。本章将简要介绍Gradle的Java插件（Java plugin），个插件会给你的构建项目添加一些任务，比如编译java类、执行单元测试和将编译的class文件打包成ja文件等。
Java插件是基于约定的（约定优于配置），它在项目的很多方面定义了默认值，例如，Java源文应该位于什么位置。我们只要遵循插件的约定，就不需要在Gradle配置脚本进行额外的相关配置。当，在某些情况下，你的项目不想或不能遵循这个约定也是可以的，这样你就需要额外的配置你的构建本。
Gradle Java插件对于项目文件存放的默认位置与maven类似。
Java源码存放在目录：src/main/java
Java测试代码存放目录：src/test/java
资源文件存放目录：src/main/resources
测试相关资源文件存放目录：src/test/resources
所有输出文件位于目录：build
输出的jar文件位于目录：build/libs
<h2>2、一个简单的Java项目</h2>
新建一个文件build.gradle，添加代码：

<pre>apply plugin: 'java'</pre>

以上代码即配置java插件到构建脚本中。当执行构建脚本时，它将给项目添加一系列任务。我们行：gradle build，来看看输出的结果：

<pre>:compileJava UP-TO-DATE
:processResources UP-TO-DATE
:classes UP-TO-DATE
:jar UP-TO-DATE
:assemble UP-TO-DATE
:compileTestJava UP-TO-DATE
:processTestResources UP-TO-DATE
:testClasses UP-TO-DATE
:test UP-TO-DATE
:check UP-TO-DATE
:build UP-TO-DATE

BUILD SUCCESSFUL</pre>

根据输出结果可以看出，我们执行的build这个任务依赖其他任务，比如compileJava等，这就是java插件预先定义好的一系列任务。
你还可以执行一些其他的任务，比如执行：gradle clean，gradle assemble，gradle check等
gradle clean：删除构建目录以及已经构建完成的文件；
gradle assemble（装配）：编译和打包java代码，但是不会执行单元测试（从上面的任务依赖果也可以看出来）。如果你应用了其他插件，那么还会完成一下其他动作。例如，如果你应用了War这个插件，那么这个务将会为你的项目生成war文件。
gradle check：编译且执行测试。与assemble类似，如果你应用了其他包含check任务的插件例如，Checkstyl

插件，那么这个任务将会检查你的项目代码的质量，并且生成检测报告。
如果想知道Gradle当前配置下哪些任务可执行，可以执行：gradle tasks，例如应用了java插件配置，执行该命令，输出：

<pre>: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 '/span>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 <pan class="pl-c1">source code.

Help tasks

components - Displays the components produced by root project 'learn-gradle' . [incubting]

dependencies - Displays all dependencies declared in root project 'learn-gradle' .

dependencyInsight - Displays the insight into a specific dependency in</pan> root project 'learn-gradle' .

help - Displays a help message.

model - Displays the configuration model of root project <span class="pl
pds">'learn-gradle' . [incubating]

projects - Displays the sub-projects of root project
/span>learn-gradle' .

properties - Displays the properties of root project
/span>learn-gradle' .

tasks - Displays the tasks runnable from root project <span class="pl-pds
>'learn-gradle' .

Verification tasks

check - Runs all checks.

test - Runs the unit tests.

Rules

Pattern: clean<TaskName>: Cl
ans the output files of a task.

Pattern: build<ConfigurationName></
pan>: Assembles the artifacts of a configuration.

Pattern: upload<ConfigurationName>
/span>: Assembles and uploads the artifacts belonging to a configuration.

To see all tasks and more detail, run gradle tasks --all

To see more detail about a task, run gradle help --task <span c
ass="pl-k"><task>

BUILD SUCCESSFUL</pre>

小伙伴们看到这里会不会有疑问，如果在构建脚本中定义了名为tasks的任务，执行会是如何？
奇的小伙伴可以自己试一试噢。事实上，是会覆盖原有的任务的。
<h2><a id="user-content-3外部依赖" class="anchor" href="https://github.com/HassanChian
/Blog/blob/master/source/_posts/gradle/learn-gradle-ch3-java-tutorial.md#3外部依赖" aria-h
dden="true">3、外部依赖</h2>
通常一个Java项目会依赖多个其他项目或jar文件，我们可以通过配置gradle仓库（repository）
诉gradle从哪里获取需要的依赖，并且gradle还可以配置使用maven仓库。例如，我们配置gradle使
maven中央仓库，在build.gradle中添加代码：

<pre>repositories {
    mavenCentral()
}</pre>

接下来，我们来添加一些依赖。代码示例：

<pre>dependencies {
    compile group: 'commons-collections<
pan class="pl-pds">', name: '</s
an>commons-collections', version: <span class="pl-s

>'3.2'
    testCompile group: 'junit<span class="
l-pds">', name: 'junit<s
an class="pl-pds">', version: '</
pan>4.+'
}</pre>

关于依赖，暂时就点这么多。详细可以参考<a href="https://docs.gradle.org/current/usergui
e/artifact_dependencies_tutorial.html">gradle依赖管理基础，也可以关注后续文章。
<h2><a id="user-content-4定义项目属性" class="anchor" href="https://github.com/HassanCh
ang/Blog/blob/master/source/_posts/gradle/learn-gradle-ch3-java-tutorial.md#4定义项目属性
 aria-hidden="true">4、定义项目属性</h2>
Java插件会为项目添加一系列的属性，通常情况下，初始的Java项目使用这些默认配置就足够了
我们不需要进行额外的配置。但是如果默认属性不满足于你的项目，你也可以进行自定义项目的一些
息。例如我们为项目指定版本号和一些jar manifest信息。

<pre>sourceCompatibility = 1.5
version = '1.0'</sp
n>
jar {
    manifest {
        attributes 'Implementation-Title<sp
n class="pl-pds">': 'Gra
le Quickstart', <span class="pl-p
s">'Implementation-Version': version
    }
}</pre>

事实上，Java插件添加的一系列任务与我们之前在脚本中自定义的任务没什么区别，都是很常规
任务。我们可以随意定制和修改这些任务。例如，设置任务的属性、为任务添加行为、改变任务的依
，甚至替换已有的任务。例如我们可以配置Test类型的test任务，当test任务执行的时候，添加一个
统属性。配置脚本如下：

<pre>test {
    systemProperties 'property<span class
"pl-pds">': 'value<span
lass="pl-pds">'
}</pre>

另外，与之前提到的“gradle tasks”命令类型，我们可以通过“gradle pro
erties”来查看当前配置所支持的可配置属性有哪些。
<h2><a id="user-content-5将jar文件发布到仓库" class="anchor" href="https://github.com/Ha
sanChiang/Blog/blob/master/source/_posts/gradle/learn-gradle-ch3-java-tutorial.md#5将jar
件发布到仓库" aria-hidden="true">5、将Jar文件发布到仓库</h2>

<pre>uploadArchives {
    repositories {
        flatDir {
            dirs '
epos'
        }
    }
}</pre>

执行gradle uploadArchives，将会把相关jar文件发布到reops仓库中。更多参考：Publishing artifacts
<h2>6、构建多个Java项目</h2>
假设我们的项目结构如下所示：

<pre>multiproject/
--api/
--services/webservice/
--shared/
--services/shared/</pre>

项目api生成jar文件，Java客户端通过jar提供的接口访问web服务；项目services/webservice一个webapp，提供web服务；项目shared 包含api和webservice公共代码；项目services/shared赖shared项目，包含webservice公共代码。
接下来，我们开始定义多项目构建。
1）首先，我们需要添加一个配置文件：settings.gradle文件。settings.gradle位于项目的根目，也就是multiproject目录。编辑settings.gradle，输入配置信息：

<pre>include 'shared', 'api', 'services:webservice', 'services:shared'</pre>

include是Gradle DSL定义的核心类型ettings的方法，用于构建指定项目。配置中指定的参数“shared”、“api”等值默认是当前配置目录的目录名称，而“services:webservice”将根据默认约定映系统物理路径"services/webservice"（相对于根目录）。关于include更详细的信息可以参考：构树。
2）定义所有子项目公用配置。在根目录创建文件：build.gradle，输入配置信息：

<pre>subprojects {
    apply plugin: 'java'
    apply plugin: 'eclipse-wtp'

repositories {
    mavenCentral()
}

dependencies {
    testCompile 'junit:junit:4.12'
}

version = '1.0'</pre>

```
jar {
    manifest.attributes provider: <span class="pl-s"><span class="pl-pds">'</span>gradle<s
an class="pl-pds">'</span></span>
}

}</pre>

</div>
<p>subprojects 是<a href="https://docs.gradle.org/current/dsl/">Gradle DSL</a>定义的构
脚本模块之一，用于定义所有子项目的配置信息。在以上配置中，我们给所有子项目定义了使用&ldq
o;java&rdquo;和&ldquo;<a href="https://docs.gradle.org/current/dsl/org.gradle.plugins.ide.e
lipse.model.EclipseWtp.html">eclipse-wtp</a>&rdquo;插件，还定义了仓库、依赖、版本号以及
ar（jar是Gradle的任务类型之一，任务是装配jar包，jar任务包含属性manifest，用于描述jar的信息
具体参考：<a href="https://docs.gradle.org/current/dsl/org.gradle.api.tasks.bundling.Jar.html
>Jar</a>）。</p>
<p>我们在根目录执行gradle build命令时，这些配置会应用到所有子项目中。</p>
<p>3）给项目添加依赖</p>
<p>新建文件：api/build.gradle，添加配置：</p>
<div class="highlight highlight-source-shell">
<pre>dependencies {
    compile project(<span class="pl-s"><span class="pl-pds">'</span>:shared<span class="p
-pds">'</span></span>)
}</pre>
</div>
<p>以上，我们定义了api项目依赖shared项目，当我们在根目录执行gradle build命令时，gradle
确保在编译api之前，先完成shared项目编译，然后才会编译api项目。</p>
<p>同样，添加services/webservice/build.gradle，添加配置：</p>
<div class="highlight highlight-source-shell">
<pre>dependencies {
    compile project(<span class="pl-s"><span class="pl-pds">'</span>:services:shared<span
lass="pl-pds">'</span></span>)
}</pre>
</div>
<p>在根目录执行：gradle compileJava，输出：</p>
<div class="highlight highlight-source-shell">
<pre>:shared:compileJava UP-TO-DATE
:shared:processResources UP-TO-DATE
:shared:classes UP-TO-DATE
:shared:jar UP-TO-DATE
:api:compileJava UP-TO-DATE
:services:compileJava UP-TO-DATE
:services:shared:compileJava UP-TO-DATE
:services:shared:processResources UP-TO-DATE
:services:shared:classes UP-TO-DATE
:services:shared:jar UP-TO-DATE
:services:webservice:compileJava UP-TO-DATE

BUILD SUCCESSFUL</pre>

</div>
<p>通过输出信息我们就可以清楚看出依赖配置是否正确啦。</p>
```