链滴

学习Gradle 2 基本的构建脚本介绍

作者：Hassan

原文链接：https://ld246.com/article/1471568865440

来源网站：链滴

<h2>1、项目和任务</h2>
Gradle 构建脚本包括两个最基本的概念，就是项目（projects）和任务（tasks）。
项目是指我们的构建产物（比如jar包）或实施产物（比如web application等）。Gradl
构建脚本包含一个或多个项目。
任务是指不可分的最小工作单元，执行构建工作（比如编译一些类文件、创建jar文件、生成java
oc以及发布架构文档到仓库等）。一个项目包含一个或多个任务。
<h2><a id="user-content-2hello-world" class="anchor" href="https://github.com/HassanChi
ng/Blog/blob/master/source/_posts/gradle/learn-gradle-ch2-basic-scripts.md#2hello-world"
ria-hidden="true">2、Hello World！！</h2>
下面我们学习一个简单的hello world例子来简单认识一下Gradle构建脚本。
新建文件：build.gradle
添加内容：

<pre>task hello {
    doLast {
        println 'Hello world!<span class="pl
pds">'
    }
}</pre>

使用命令行进入build.gradle所在目录，执行：gradle hello ，输出：

<pre>:hello
Hello world!

BUILD SUCCESSFUL</pre>

在这个例子中build.gradle 文件就是一个构建脚本（严格的说，这是一个构建配置脚本），这个
本定义了一个项目以及项目包含的任务。
Gradle是领域驱动设计的构建工具，在它的实现当中，Project接口对应上面的project概念，Tas
接口对应上面的task概念，实际上除此之外还有一个重要的领域对象，即Action，对应的是task里面
体的某一个操作。一个project由多个task组成，一个task也是由多个action组成。
当执行gradle hello的时候，Gradle就会去调用这个hello task来执行给定操作(Action)。这个操
其实就是一个用Groovy代码写的闭包，代码中的task是Project类里的一个方法，通过调用这里的tas
方法创建了一个Task对象，并在对象的doLast方法中传入 println 'Hello world!' 这个闭包。这个闭
就是一个Action。
Task是Gradle里定义的一个接口，表示上述概念中的task。它定义了一系列的诸如doLast, doFirs
等抽象方法，具体可以看gradle api里org.gradle.api.Task的文档。
<h2><a id="user-content-3使用快捷键定义任务" class="anchor" href="https://github.com/Ha
sanChiang/Blog/blob/master/source/_posts/gradle/learn-gradle-ch2-basic-scripts.md#3使用
捷键定义任务" aria-hidden="true">3、使用快捷键定义任务</h2>
doLast还有另外一种简单的写法：

<pre>task hello << {
    println 'Hello world!<span class="pl-pd
">'
}</pre>

执行gradle hello 命令，输出结果与之前的相同。也就是我们把像doLast这样的代码，直接简化
<<这个符号了。这其实是Gradle利用了Groovy的操作符重载的特性，把左位移操作符实现为将a
tion加到task的最后，相当于调用doLast方法。看Gradle的api文档里对doLast()和leftShift()这两个
法的介绍，可知它们的作用是一样的，所以在这里，<<左移操作符即doLast的简写方式。

<h2>4、脚本即代码</h2>
在Gradle的构建脚本中，可以使用Groovy代码以实现更强大的功能。
例1：

<pre>task upper << {
    String someString = 'mY_nAmE'
    println "Original: " + someString
    println "Upper case: " + someString.toUpperCase()
}</pre>

执行gradle upper，输出：

<pre>:upper
Original: mY_nAmE
Upper case: MY_NAME

BUILD SUCCESSFUL</pre>

例2：

<pre>task count << {
    4.times { print "$it " }
}</pre>

执行gradle count，输出：

<pre>:count
0 1 2 3
BUILD SUCCESSFUL</pre>

<h2>5、任务依赖</h2>
1、在脚本中定义任务依赖：

<pre>task hello << {
    println 'Hello world!'
}
task intro(dependsOn: hello) << {
    println "I'm Gradle"
}</pre>

执行gradle intro，输出：

<pre>:hello

```
Hello world<span class="pl-k">!</span>
:intro
I<span class="pl-s"><span class="pl-pds">'</span>m Gradle</span>

<span class="pl-s">BUILD SUCCESSFUL</span></pre>

</div>
<p>2、任务可以依赖尚未出现的任务：</p>
<div class="highlight highlight-source-shell">
<pre>task taskX(dependsOn: <span class="pl-s"><span class="pl-pds">'</span>taskY<span class="pl-pds">'</span></span>) <span class="pl-k">&amp;lt;&amp;lt;</span> {
    println <span class="pl-s"><span class="pl-pds">'</span>taskX<span class="pl-pds">'</span></span>
}
task taskY <span class="pl-k">&amp;lt;&amp;lt;</span> {
    println <span class="pl-s"><span class="pl-pds">'</span>taskY<span class="pl-pds">'</span></span>
}</pre>
</div>
<p>本例中任务taskX依赖taskY，但是taskY在taskX之后才定义。执行gradle taskX，输出：</p>
<div class="highlight highlight-source-shell">
<pre>:taskY
taskY
:taskX
taskX

BUILD SUCCESSFUL</pre>

</div>
<h2><a id="user-content-6动态任务" class="anchor" href="https://github.com/HassanChian/Blog/blob/master/source/_posts/gradle/learn-gradle-ch2-basic-scripts.md#6动态任务" aria-hidden="true"></a>6、动态任务</h2>
<p>我们可以使用Groovy的语法动态创建任务，例如：</p>
<div class="highlight highlight-source-shell">
<pre>4.<span class="pl-c1">times</span> { counter -<span class="pl-k">&amp;gt;</span>
    task <span class="pl-s"><span class="pl-pds">"</span>task<span class="pl-smi">$counter</span><span class="pl-pds">"</span></span> <span class="pl-k">&amp;lt;&amp;lt;</span> {
        println <span class="pl-s"><span class="pl-pds">"</span>I'm task number <span class="pl-smi">$counter</span><span class="pl-pds">"</span></span>
    }
}</pre>
</div>
<p>执行gradle -q task1，输出：</p>
<div class="highlight highlight-source-shell">
<pre>:task1
I<span class="pl-s"><span class="pl-pds">'</span>m task number 1</span>

<span class="pl-s">BUILD SUCCESSFUL</span></pre>

</div>
<h2><a id="user-content-7操纵任务" class="anchor" href="https://github.com/HassanChian/Blog/blob/master/source/_posts/gradle/learn-gradle-ch2-basic-scripts.md#7操纵任务" aria-hidden="true"></a>7、操纵任务</h2>
```

已经创建的任务可以通过api（例1用到的api是dependsOn ）进行访问。
例1：给一个任务添加依赖

<pre>4.times { counter ->
    task "task$coun
er" << {
        println "I'm task number <span clas
="pl-smi">$counter"
    }
}
task0.dependsOn task2, task3</pre>

执行gradle task0 ，输出

<pre>:task2
I'm task number 2
:task3
I'm task number 3
:task0
I'm task number 0

BUILD SUCCESSFUL</pre>

例2：给一个任务添加行为

<pre>task hello << {
    println 'Hello Earth<span class="pl-pds
>'
}
hello.doFirst {
    println 'Hello Venus<span class="pl-pd
">'
}
hello.doLast {
    println 'Hello Mars<span class="pl-pds
>'
}
hello << {
    println 'Hello Jupiter<span class="pl-p
s">'
}</pre>

执行gradle hello，输出：

<pre>:hello
Hello Venus
Hello Earth
Hello Mars
Hello Jupiter

BUILD SUCCESSFUL</pre>

任务先执行了doFirst，再按顺序执行了doLast（"<<"可以理解为doLast的别名，所以相同api方法将按照配置文件顺序执行）
<h2>8、自定义属性</h2>

<pre>task myTask {
    ext.myProperty = "myValue"
}
task printTaskProperties << {
    println myTask.myProperty
}</pre>

执行gradle printTaskProperties，输出：

<pre>:printTaskProperties
myValue

BUILD SUCCESSFUL</pre>

<h2>9、默认任务</h2>
gradle允许在构建过程中配置一个或多个任务作为默认任务来执行，例如：

<pre>defaultTasks 'clean', 'run'

task clean << {
println 'Default Cleaning!'
}

task run << {
println 'Default Running!'
}

task other << {
println "I'm not a default task!"
}</pre>

执行 gradle ，输出：

<pre>:clean
Default Cleaning!

原文链接：学习Gradle 2 基本的构建脚本介绍

```
:run
Default Running<span class="pl-k">!</span>

BUILD SUCCESSFUL</pre>

</div>
<h2><a id="user-content-10dagdirected-acyclic-graph有向非循环图配置" class="anchor" hre
="https://github.com/HassanChiang/Blog/blob/master/source/_posts/gradle/learn-gradle-ch
-basic-scripts.md#10dagdirected-acyclic-graph有向非循环图配置" aria-hidden="true"></a>1
、DAG（Directed acyclic graph，有向非循环图）配置</h2>
<p>Gradle构建的生命周期包含初始化阶段、配置阶段和执行阶段。Gradle使用DAG来记录任务执
的顺序。配置阶段完成后，Gradle就明确了所有需要被执行的任务，这些任务将被存储到taskGraph
Gradle提供了一个钩子来使用这个（taskGraph）信息。下面这个例子我们将判断taskGraph是否包含
elease任务来确定项目发布的版本号。</p>
<div class="highlight highlight-source-shell">
<pre>task distribution <span class="pl-k">&amp;lt;&amp;lt;</span> {
    println <span class="pl-s"><span class="pl-pds">"</span>We build the zip with version=
span class="pl-smi">$version</span><span class="pl-pds">"</span></span>}

task release(dependsOn: <span class="pl-s"><span class="pl-pds">                    '</s
an>distribution<span class="pl-pds">'</span></span>          ) <span class="pl-k">&amp;lt;&amp;lt;
/span> {
println <span class="pl-s"><span class="pl-pds">                    '</span>We release n
w<span class="pl-pds">'</span></span>          }

gradle.taskGraph.whenReady {taskGraph -<span class="pl-k">&amp;gt;</span>
<span class="pl-k">if</span> (taskGraph.hasTask(release)) {
version = <span class="pl-s"><span class="pl-pds">                    '</span>1.0<span c
ass="pl-pds">'</span></span>
} <span class="pl-k">else</span> {
version = <span class="pl-s"><span class="pl-pds">                    '</span>1.0-SNAP
HOT<span class="pl-pds">'</span></span>
}
}</pre>

</div>
<p>执行 gradle distribution，输出：</p>
<div class="highlight highlight-source-shell">
<pre>:distribution
We build the zip with version=1.0-SNAPSHOT

BUILD SUCCESSFUL</pre>

</div>
<p>执行gradle release，输出：</p>
<div class="highlight highlight-source-shell">
<pre>:distribution
We build the zip with version=1.0
:release
We release now
```

BUILD SUCCESSFUL</pre>

原文链接：学习Gradle 2 基本的构建脚本介绍