**1 自定义task**

在[本系列](http://www.cnblogs.com/davenkin/p/gradle-learning-1.html)的[上篇文章](http://www.cnblogs.com/davenkin/p/gradle-learning-8.html)中，我们学习了多Project构建，在本篇文章中，我们将学到如何自定义Task类型。

　　请通过以下方式下载本系列文章的Github示例代码：

git clone https://github.com/davenkin/gradle-learning.git

　　在前面的文章中我们讲到，Gradle本身只是一个架子，真正起作用的是Task和Plugin。要真正了解Task和Plugin的工作机制并熟练运用，学会自定义Task类型和Plugin是大有裨益的。

　　Gradle中的Task要么是由不同的Plugin引入的，要么是我们自己在build.gradle文件中直接创建的。在默认情况下，我们所创建的Task是DefaultTask类型，该类型是一个非常通用的Task类型，而在有些时候，我们希望创建一些具有特定功能的Task，比如Copy和Jar等。还有时候，我们希望定义自己创建的Task类型，在本文中，我们以定义一个简单的HelloWorldTask为例，讲解如何自定义一个Task类型，并且如何对其进行配置。

　　在Gradle中，我们有3种方法可以自定义Task类型。

***（一）在build.gradle文件中直接定义***

　　我们知道，Gradle其实就是groovy代码，所以在build.gradle文件中，我们便可以定义Task类。



class HelloWorldTask extends DefaultTask {

@Optional

String message = 'I am davenkin'

@TaskAction

def hello(){

println "hello world $message"

}

}

task hello(type:HelloWorldTask)

task hello1(type:HelloWorldTask){

message ="I am a programmer"

}



　　在上例中，我们定义了一个名为HelloWorldTask的Task，它需要继承自DefaultTask，它的作用是向命令行输出一个字符串。@TaskAction表示该Task要执行的动作，即在调用该Task时，hello()方法将被执行。另外，message被标记为 @Optional，表示在配置该Task时，message是可选的。在定义好HelloWorldTask后，我们创建了两个Task实例，第一个hello使用了默认的message值，而第二个hello1在创建时重新设置了message的值。

　　在执行hello时，命令行输出如下：

:hello

hello world I am davenkin

BUILD SUCCESSFUL

Total time: 2.139 secs

　　在执行hello1时，命令行输出如下：

:hello1

hello world I am a programmer

BUILD SUCCESSFUL

***（二）在当前工程中定义Task类型***

　　在（一）中，我们在build.gradle中直接定义了Task的类型，这样将Task的定义和使用混在一起。在需要定义的Task类型不多时，我们可以采用这种方法，但是在项目中存在大量的自定义Task类型时，这就不见得是中好的做法了。一种改进方法是在另外的一个gradle文件中定义这些Task，然后再apply到build.gradle文件中。这里，我们将使用另一种方法：**在buildSrc目录下定义Task类型**，**Gradle在执行时，会自动地查找该目录下所定义的Task类型，并首先编译该目录下的groovy代码以供build.gradle文件使用。**

　　在当前工程的buildSrc/src/main/groovy/davenkin目录下创建HelloWorldTask.groovy文件，将（1）中对HelloWorldTask的定义转移到该文件中：



package davenkin

import org.gradle.api.\*

import org.gradle.api.tasks.\*

class HelloWorldTask extends DefaultTask {

@Optional

String message = 'I am davenkin'

@TaskAction

def hello(){

println "hello world $message"

}

}



　　这里，我们将HelloWorldTask定义在了davenkin包下，因此在build.gradle文件中引用该Task时，我们需要它的全名称：

task hello(type:davenkin.HelloWorldTask)

task hello1(type:davenkin.HelloWorldTask){

message ="I am a programmer"

}

　　以上的hello和hello1与（1）中的hello和hello1完成的功能相同。

***（三）在单独的项目中定义Task类型***

　　虽然（2）中的Task定义与build.gradle分离开了，但是它依然只能应用在当前工程中。如果我们希望所定义的Task能够用在另外的项目中，那么（2）中的方法便不可行的，此时我们可以将Task的定义放在单独的工程中，然后在所有使用Task的工程中通过声明依赖的方式引入这些Task。

**创建另外一个项目，将（2）中buildSrc目录下的内容考到新建项目中，由于该项目定义Task的文件是用groovy写的，因此我们需要在该项目的build.gradle文件中引入groovy Plugin。**另外，由于该项目的输出需要被其他项目所使用，因此我们**还需要将其上传到repository中**，在本例中，我们将该项目生成的包含了Task定义的jar文件上传到了本地的文件系统中。最终的build.gradle文件如下：



apply plugin: 'groovy'

apply plugin: 'maven'

version = '1.0'

group = 'davenkin'

archivesBaseName = 'hellotask'

repositories.mavenCentral()

dependencies {

compile gradleApi()

groovy localGroovy()

}

uploadArchives {

repositories.mavenDeployer {

repository(url: 'file:../lib')

}

}



　　执行“gradle uploadArchives”，所生成的jar文件将被上传到上级目录的lib(../lib)文件夹中。

　　在使用该HelloWorldTask时，客户端的build.gradle文件可以做以下配置：



buildscript {

repositories {

maven {

url 'file:../lib'

}

}//**需要告诉Gradle到何处去取得依赖**

dependencies {

classpath group: 'davenkin', name: 'hellotask', version: '1.0'

}//**声明对HelloWorldTask的依赖**

}

task hello(type: davenkin.HelloWorldTask)//**依赖用于当前build文件**



　　首先，**我们需要告诉Gradle到何处去取得依赖，即配置repository。另外，我们需要声明对HelloWorldTask的依赖，该依赖用于当前build文件**。之后，对hello的创建与（2）中一样。

　　在[下一篇](http://www.cnblogs.com/davenkin/p/gradle-learning-10.html)文章中，我们将讲到如何自定义Plugin。

**1 自定义plugin**

在[本系列](http://www.cnblogs.com/davenkin/p/gradle-learning-1.html)的[上篇文章](http://www.cnblogs.com/davenkin/p/gradle-learning-9.html)中，我们讲到了如何自定义Task类型，在本篇文章中，我们将讲到如何自定义Plugin。

　　请通过以下方式下载本系列文章的Github示例代码：

git clone https://github.com/davenkin/gradle-learning.git

　　在Plugin中，我们可以向Project中加入新的Task，定义configurations和property等。我们3种方法可以自定义Plugin，这些方法和自定义Task类型的3种方法相似。在接下来的例子中，我们将分别通过这3种方法来创建一个DateAndTimePlugin，该Plugin定义了2个Task，分别用于输出系统当前的日期和时间，另外，我们可以配置日期和时间的输出格式。

***（一）在build.gradle文件中直接定义Plugin***

　　和在build.gradle文件中定义Task类型一样，我们可以将对Plugin的定义直接写在build.gradle中：



apply plugin: DateAndTimePlugin

dateAndTime {**//extension**

timeFormat = 'HH:mm:ss.SSS'

dateFormat = 'MM/dd/yyyy'

}

class DateAndTimePlugin implements Plugin<Project> {

void apply(Project project) {

project.extensions.create("dateAndTime", DateAndTimePluginExtension)

project.task('showTime') << {

println "Current time is " + new Date().format(project.dateAndTime.timeFormat)

}

project.tasks.create('showDate') << {

println "Current date is " + new Date().format(project.dateAndTime.dateFormat)

}

}

}

class DateAndTimePluginExtension {

String timeFormat = "MM/dd/yyyyHH:mm:ss.SSS"

String dateFormat = "yyyy-MM-dd"

}



　　每一个自定义的Plugin都需要实现Plugin<T>接口，事实上，除了给Project编写Plugin之外，我们还可以为其他Gradle类编写Plugin。该接口定义了一个apply()方法，在该方法中，我们可以操作Project，比如向其中加入Task，定义额外的Property等。

　　在上例中，我们在DateAndTimePlugin中向Project添加了2个Task，一个名为showTime，一个名为showDate。请注意创建这2个Task所使用的不同方法，更多的创建Task的方法，请参考本系列[这篇文章](http://www.cnblogs.com/davenkin/p/gradle-learning-2.html)。

**每个Gradle的Project都维护了一个ExtenionContainer，我们可以通过project.extentions进行访问，比如读取额外的Property和定义额外的Property等。在DateAndTimePlugin中，我们向Project中定义了一个名为dateAndTime的extension，并向其中加入了2个Property，分别为timeFormat和dateFormat，他们又分别用于showTime和showDate。在使用该Plugin时，我们可以通过以下方式对这两个Property进行重新配置：**

dateAndTime {

timeFormat = 'HH:mm:ss.SSS'

dateFormat = 'MM/dd/yyyy'

}

***（二）在当前工程中定义Plugin***

**在当前工程中的buildSrc/src/main/groovy/davenkin目录下创建**DateAndTimePlugin.groovy文件，将build.gradle中定义DateAndTimePlugin的代码提取到给文件中，但是除去对DateAndTimePluginExtension的定义，因为我们将在另外一个单独的文件中定义DateAndTimePluginExtension。



package davenkin

import org.gradle.api.Plugin

import org.gradle.api.Project

class DateAndTimePlugin implements Plugin<Project> {

void apply(Project project) {

project.extensions.create("dateAndTime", DateAndTimePluginExtension)

project.task('showTime') << {

println "Current time is " + new Date().format(project.dateAndTime.timeFormat)

}

project.tasks.create('showDate') << {

println "Current date is " + new Date().format(project.dateAndTime.dateFormat)

}

}

}



　　再创建DateAndTimePluginExtension.groovy：

package davenkin

class DateAndTimePluginExtension {

String timeFormat = "MM/dd/yyyyHH:mm:ss.SSS"

String dateFormat = "yyyy-MM-dd"

}

　　这里，我们将2个类文件都放在了davenkin包下。**Gradle在执行时，会自动扫描buildSrc目录，并会在执行Task之前构建该目录下的内容**。在build.gradle文件中，在apply该Plugin时，我们需要声明对该Plugin的全名称，即包含报名：

apply plugin: davenkin.DateAndTimePlugin

dateAndTime {

timeFormat = 'HH:mm:ss.SSS'

dateFormat = 'MM/dd/yyyy'

}

　　执行“gradle showTime”，命令行输出如下：



:buildSrc:compileJava UP-TO-DATE

:buildSrc:compileGroovy UP-TO-DATE

:buildSrc:processResources UP-TO-DATE

:buildSrc:classes UP-TO-DATE

:buildSrc:jar UP-TO-DATE

:buildSrc:assemble UP-TO-DATE

:buildSrc:compileTestJava UP-TO-DATE

:buildSrc:compileTestGroovy UP-TO-DATE

:buildSrc:processTestResources UP-TO-DATE

:buildSrc:testClasses UP-TO-DATE

:buildSrc:test UP-TO-DATE

:buildSrc:check UP-TO-DATE

:buildSrc:build UP-TO-DATE

:showTime

Current time is 19:08:35.489

BUILD SUCCESSFUL

Total time: 2.995 secs



　　可以看到，Gradle会首先构建buildSrc目录，然后才执行showTime（红色部分）。

***（三）在单独的项目中创建Plugin***

　　新建一个项目，**将（二）中buildSrc目录下的内容拷贝到该项目下(buildSrc/src/main/groovy/davenkin下的文件拷贝到app/src/main/groovy/davenkin下)**，定义该项目的build.gradle文件如下：



apply plugin: 'groovy'

apply plugin: 'maven'

version = 1.0

group = 'davenkin'

archivesBaseName = 'datetimeplugin'

repositories.mavenCentral()

dependencies {

compile gradleApi()

compile localGroovy()

}

uploadArchives {

repositories.mavenDeployer {

repository(url: 'file:../lib')

}

}



　　此外，我们还可以为该Plugin重新命名，**如果我们希望将该Plugin命名为time，那么我们需要在src/main/resources/META-INF/gradle-plugins目录下创建名为time.properties的文件，内容如下：**

**implementation-class = davenkin.DateAndTimePlugin**

**在执行“gradle uploadArchives”时，Gradle会将该Plugin打包成jar文件，然后将其上传到上级目录下的lib目录中（../lib）**。之后，在客户端的build.gradle文件中，我们需要做如下定义：



buildscript {

repositories {

maven {

url 'file:../lib'

} }

dependencies {

classpath group: 'davenkin', name: 'datetimeplugin',

version: '1.0'

或者

classpath ‘davenkin:datetimeplugin:1.0’

}

}

apply plugin: 'time'

dateAndTime {

timeFormat = 'HH:mm:ss.SSS'

dateFormat = 'MM/dd/yyyy'

}



　　首先我们配置repository以执行lib目录，然后声明对DateAndTimePlugin的依赖，再apply该Plugin，此时我们应该使用“time”作为该Plugin的名称，最后对该Plugin进行配置。

额外的

class DateAndTimePlugin implements Plugin<Project> {

void apply(Project project) {

project.extensions.create("dateAndTime", DateAndTimePluginExtension)

project.dateAndTime.extensions.create(“elseprop”,ElseProp);

**//为 dateAndTime extension加上一个extension**

project.task('showTime') << {

println "Current time is " + new Date().format(project.dateAndTime.timeFormat)

}

project.tasks.create('showDate') << {

println "Current date is " + new Date().format(project.dateAndTime.dateFormat)

}

}

}

class ElseProp{

String prop=”hello else”

}