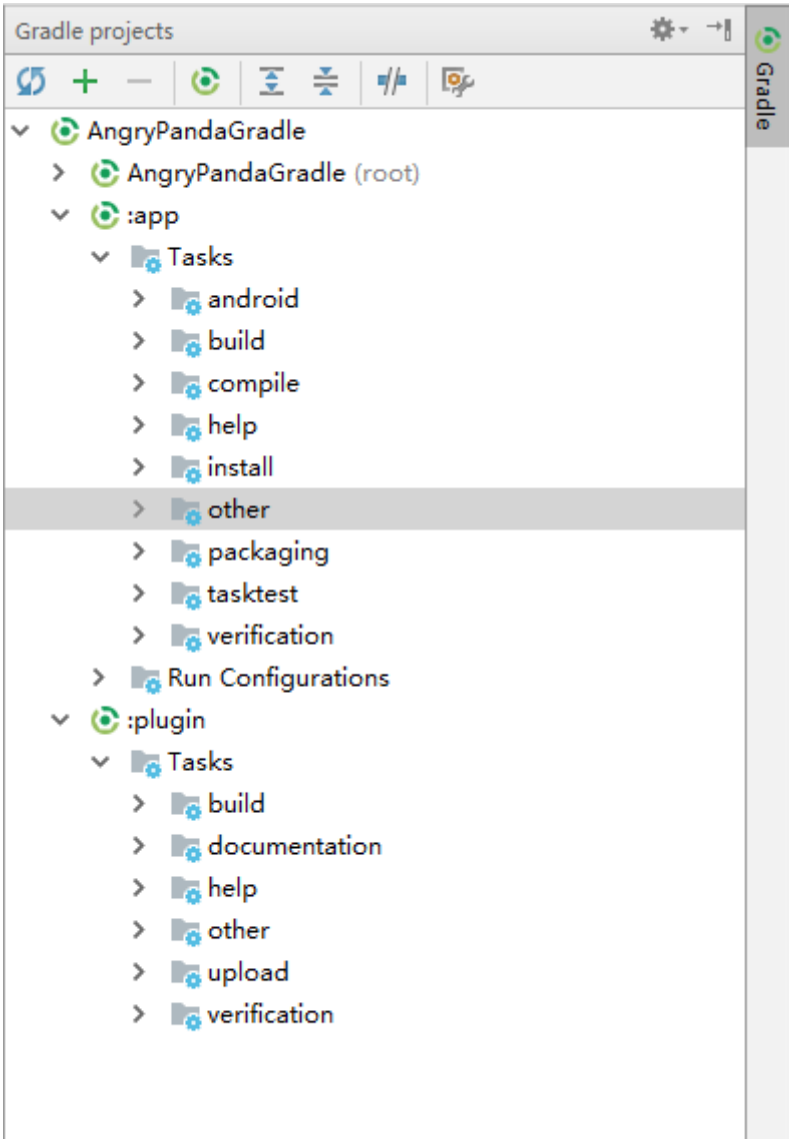
版本号	描述	日期(更新日期)/更新人
1.0	Android Studio Gradle技术	2019/05/24 **刘志保**
Github地址	https://github.com/MMLoveMeMM/ AngryPandaGradle	
	https://github.com/davenkin/gradle-learning	

<一> : 首先先看一下Android Studio 任务Task组织管理架构,如下图:



文件目录说明:

:app 是项目组(也可以说成是google提供的默认开发插件,用于完成工程的编译和生成工作的)

Tasks是所有任务组列表

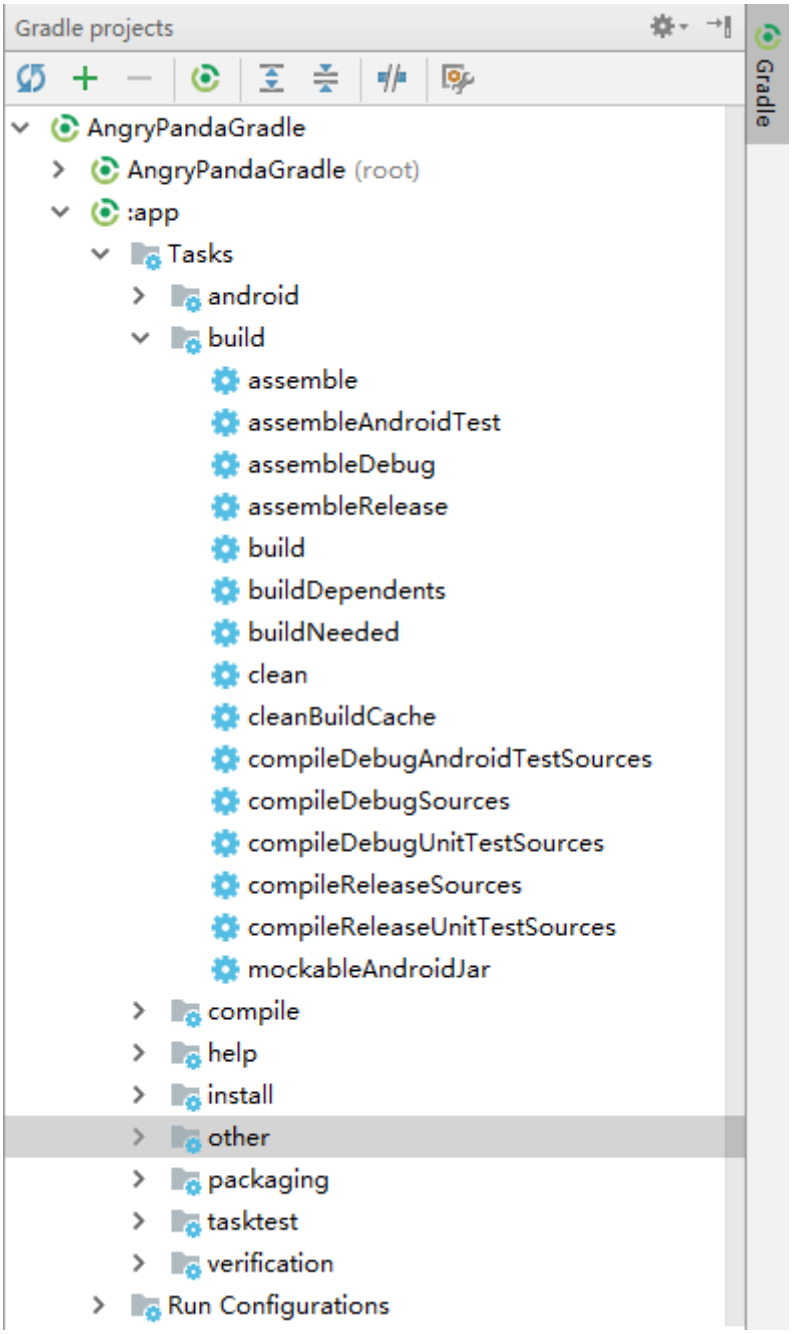
android

... 是任务组,任务组可以在创建任务Task时指定任务组,属性是group,group是标记当前Task的其中一个标识.

比如: 下面的分组就在tasktest下,那么Tasks下面就会多出一个tasktest,如上图中tasktest.

```
tasks.create(name: 'test') {
 group 'tasktest'
 description 'test task'
 println "[phase:configuration] test"
 doLast {
 println "[phase:execution] test:doLast()"
 }
}
```

在展开任务组看看,如下:



这个是group 'build' 任务组展开图,列出了所有build组下面的任务Task,用鼠标双击其中任意一个任务,该任务都会启动执行任务.

<二> 生命周期:

然后在看看Task的任务周期执行,先新建一个cycle.gradle文件,并且将该文件引入build.gradle文件中:

```
apply plugin: 'com.android.application'
apply from: this.file('cycle.gradle')
```

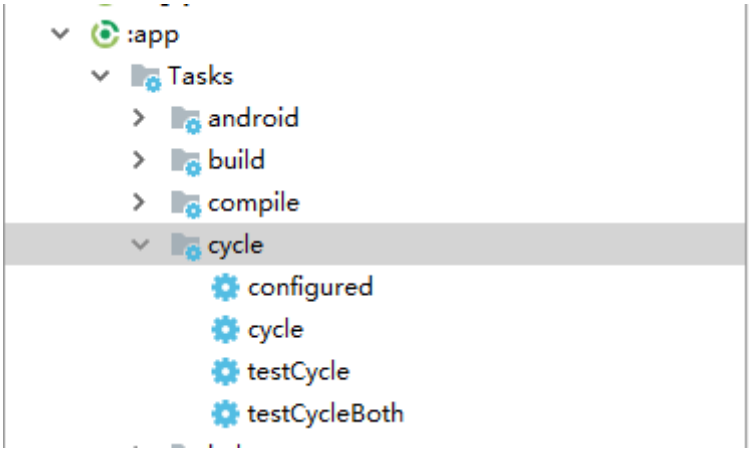
cycle.gradle 写入Task周期运行测试程序,如下:

```
println 'This is executed during the configuration phase.'

```

```
task configured {
 group 'cycle'
 println 'This is also executed during the configuration phase.'
}

task testCycle {
 group 'cycle'
  doLast {
  println 'This is executed during the execution phase.'
  }
 }

 task testCycleBoth {
  group 'cycle'

  doFirst {
  println 'This is executed first during the execution phase.'
  }
  doLast {
  println 'This is executed last during the execution phase.'
  }
  println 'This is executed during the configuration phase as well.'
 }

 class CycleTask extends DefaultTask{
  String installObjectName

  @TaskAction
  void checkObject() {
  println "[phase:execution] install:checkObject (${installObjectName})"
  }

  @TaskAction
  void installObject() {
  println "[phase:execution] install:installObject (${installObjectNam
e})"
  }
 }

 task cycle(type: CycleTask) {
  group 'cycle'
```

```
43  description 'install task'
44  installObjectName 'liuzhibao.jar'
45
46  println "[phase:configuration] install"
47  doFirst {
48  println "[phase:execution] install:doFirst()"
49  }
50  doLast {
51  println "[phase:execution] install:doLast()"
52  }
53 }
```

同时这里新建了一个group 'cycle'的分组:



双击cycle下面任意其中一个,即可以启动Task任务.

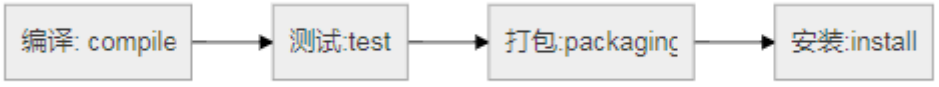
结合Task.class源码,可以看出每个任务都可以实现doFirst和doLast方法,如果需要,可以"重写"它们来完成一些任务.

doFirst是在Task最开始之前执行的,可以在任务执行之前初始化一些事物

doLast是当前Task所有工作都完成了,收尾的时候调用doLast.

这是Task自身执行过程,但是如果Task置身于整个工作组中,要完成一项Task的流程是如何的呢?

参照下面的:



一个Task开始执行,基本上要经历上面四个过程,才算是一个Task的完整执行过程.

举例如下:

```
1  println "[phase:configuration] build.gradle ..."
2
3  task compile {
4   group 'compile'
```

```
 description 'compile task'
 println "[phase:configuration] compile"
 doFirst {
 println "[phase:execution] compile :doFirst()"
 }
}

tasks.create(name: 'test') {
 group 'test'
 description 'test task'
 println "[phase:configuration] test"
 doLast {
 println "[phase:execution] test:doLast()"
 }
}

tasks.create("packaging") {
 group 'packaging'
 description 'packaging task'
 println "[phase:configuration] packaging"
 doLast {
 println "[phase:execution] packaging:doLast()"
 }
}

class Install extends DefaultTask{
 String installObjectName

 @TaskAction
 void checkObject() {
 println "[phase:execution] install:checkObject (${installObjectName})"
 }

 @TaskAction
 void installObject() {
 println "[phase:execution] install:installObject (${installObjectNam
e})"
 }
}

task install(type: Install) {
```

```
 group 'install'
 description 'install task'
 installObjectName 'test.jar'

 println "[phase:configuration] install"
 doFirst {
 println "[phase:execution] install:doFirst()"
 }
 doLast {
 println "[phase:execution] install:doLast()"
 }
}
```

执行上面的任务,日志如下:[gradle compile test packaging install]

```
[Phase: initialization] : settings executed...

> Configure project :
[phase:configuration] build.gradle ...
[phase:configuration] compile
[phase:configuration] test
[phase:configuration] packaging
[phase:configuration] install

> Task :compile
[phase:execution] compile :doFirst()

> Task :test
[phase:execution] test:doLast()

> Task :packaging
[phase:execution] packaging:doLast()

> Task :install
[phase:execution] install:doFirst()
[phase:execution] install:installObject (test.jar)
[phase:execution] install:checkObject (test.jar)
[phase:execution] install:doLast()

BUILD SUCCESSFUL in 0s
4 actionable tasks: 4 executed
liumiaocn:hello liumiao$
```

```
28
```

会发现,所有的任务都先执行配置compile过程,再执行test过程,再执行打包packaging和install过程.

那么知道Task生命周期了,我们可以做什么呢?
工作中常用的,编译完APK以后需要重写定义输出路径和APK名,以适应不同环境的需要,那么如何操作呢?首先要在编译任务完成后才能够开始,那么编译任务是由build任务完成的,那么build任务执行完成后,就应该"重写"doLast方法,即在build.gradle中添加:

```
build.doLast {
 def fileName = "app-debug.apk"
 def fromFile = "build/outputs/apk/debug/" + fileName
 def intoFile = "build/outputs/"

 def applicationId = android.defaultConfig.applicationId
 def versionName = android.defaultConfig.versionName
 def time = new java.text.SimpleDateFormat("yyyyMMddHHmmss").format(new D
ate())
 def buildType = "realse"
  def channel = "site"
  def appName = "${applicationId}_v${versionName}_${time}_${buildType}_
${channel}.apk"
  // copy --> rename
  copy {
  from fromFile
  into intoFile
  rename {
  appName
  }
  }
  println("====================build.doLast success.====================
======")
}
```

那么编译完成的时候,就会开始执行build.doLast.
当然一个Task还可以提供其他方方法,比如:

clean任务除了提供默认的doFirst和doLast,还提供的其他的如delete等"方法",比如我自定义个如下:

```
// 自定义任务
class HelloWorldTask extends DefaultTask {
 @Optional
 String message = 'I am davenkin'

 @TaskAction
 def hello(){
 println("hello world ${message}")
 }

 @Override
 Task doFirst(Action<? super Task> action) {
 println("hello world \n")
 return this
 }

 @Override
 Task doLast(Action<? super Task> action) {
 println("goodbye \n")
 return super.doLast(action)
 }

 Task doHello(Action<? super Task> action) {
 println("do hello ! \n")
 return this
```

```
 }

 def doHelloWork(){
 println("lets do hello work! \n")
 }
}
```

上面的doHello和doHelloWork即是我自己配置一个给其他人调用的.那如何调用呢?

```
task hello1(type:HelloWorldTask){
 dependsOn hello
 message ="I am a programmer"
}
```

然后假如放到build.doLast中执行:

```
hello1.doHelloWork()
```

doHello和doLast/doFirst有区别的,doHello只能够被调用,不能够被重写,基本上和doHelloWork方法调用没什么区别了.

<三> Task和Task调用以及依赖关系,依赖关系其实很简单,就是一个任务可能需要另外一个任务完成后才启动.

举例如下:

```

task deTask << {
 group 'depends'

 println("doTask hello ...")
}

task depTask{
 group 'depends'
 // dependsOn deTask

 println("depTask hello ...")
}

tasks.create("depends"){
 group 'depends'
 dependsOn depTask
 doLast {
 // 通过直接调用execute来完成任务Task
 deTask.execute()
```

```
 println("tasks create")
 }
}

depTask.dependsOn deTask
```

可以通过依赖去执行依赖的任务,也可以直接调用Task的execute执行.

<四> 任务Task的属性:
我们前面介绍周期的时候提到需要执行packaging过程.现在可以通过属性忽略之.

```
tasks.create("packaging") {
 group 'packaging'
 description 'packaging task'
 dependsOn test
 enabled false
 println "[phase:configuration] packaging"
 doLast {
 println "[phase:execution] packaging:doLast()"
 }
}
```

上面的属性group,dependsOn已经介绍了, enabled 这个关键属性即表示该任务是否需要忽略不执行,如果enabled false,即该任务将不会被执行,默认该属性为true.

另外我们还有其他方式可以屏蔽某个任务不执行,比如关闭lint检查:
命令行:

```
gradlew check -x lint
```

或者修改:

```
tasks.whenTaskAdded { task ->
 if (task.name.equals("lint")) {
 task.enabled = false
 }
}
```

不过上面经测试,通过命令行的方式效果最好,可以彻底禁止lint各种任务,不会有lint报告出来.

<五> 内置的一些任务可以调用,比如上一篇文章的拷贝,移动,创建,删除等,参考<Gradle 工作中常用点 亲测一> 第<五>节部分.

-注意-
其实看完,自己在操作一番,就会发现,无论是google工程师写得编译apk的build.gradle中的任务,还是自己写的任务,都可以再次挂在操作任务,比如google的build任务,我们只需要"重写"build.doLast就可以在编译后添加自己需要的变更,也可以在编译之前doFirst去变更一些处理,比如埋点操作,就是在编译的时候将"桩"打入进去,不仅仅是build任务其他的google工程师开发的任务都是可以的,也可以自己开发自定义任务去完成自己想要的工作任务,减轻自己的一些繁琐操作.