**1 groovy相关知识**

**1.1 数据结构**

**1.1.1 基本数据类型int long float等，没啥可说的**

**1.1.2 字符串**

既可以用双引号，也可以用单引号。区别是，用**双引号定义的字符串可以**插值,还可以三引号，三引号中的字符串可任意换行****

1：单引号：

def str = 'java'

def str =' "this is a string with 双引号" '

//注意这个单引号中间的双引号不会进行转义，也就是说这个里面的双引号也是字符串里面的部分

2：双引号: def pi = 3.14

def str = "pi is ${pi}"  //结果值是”pi is 3.14 ”

3: 三引号：d

ef str=''' begin

     line  1

     line  2

     end '''

**1.1.3 数组**

Int[] arr={1,2,3,4,5}

**1.1.4 Map类型**

def langs = ['C++':'Stroustrup', 'Java':'Gosling', 'Lisp':'McCarthy']

def langs = ['C++':'Stroustrup', Java:'Gosling', Lisp:'McCarthy']

Langs.addkeyname=’addvalue’//添加一个[addkeyname:’addvalue’]对象

apply plugin: 'java';//调用apply函数传入参数是一个Map对象

特性：key和value用冒号分隔，pair之间用逗号分隔。另外，用这种语法创建出来的是java.util.LinkedHashMap实例。如果key是普通的字符串，还可以进一步省略掉引号.

易混淆之处：

def aconfusekey=”strkey”

def langs=[aconfusekey:”strValue”]

此时langs里面那个元素的key值是“aconfusekey”如果想使用外部变量的值作为key则要Def langs=[(aconfusekey):”strValue”]这么定义。

读取：形如 println langs[‘Java’] 或者

println langs.Java

**1.1.5 List类型**

def aList = [5,'string',**true**]

aList[100]=1000;

特性：List由[]定义，其元素可以是任何对象 ，变量存取可以直接通过索引存取，而且不用担心索引越界。如果索引超过当前链表长度，List会自动 往该索引添加元素

**1.2 函数相关**

**1.2.1 函数定义**

**（1）定义时可以不指定参数的类型**

String testFunction(arg1**,**arg2){

}

**（2）定义返回值也可以不指定类型，这种情况下要加def关键字定义**

def  nonReturnTypeFunc(){

    last\_line   //最后一行代码的执行结果就是本函数的返回值

}

**（3）函数返回值：Groovy的函数里，可以不使用return xxx来设置xxx为函数返回值。如果不使用return语句的话，则函数里最后一句代码的执行结果被设置成返回值。但是如果指定了返回参数类型，则必须返回对应类型参数**

1. def getSomething(){
2. "getSomething return value" //如果这是最后一行代码，则返回类型为String
3. 1000//如果这是最后一行代码，则返回类型为Integer
4. }

**1.2.1 函数调用**

**（1）方法调用可以省略引号**

**（2）如果传入参数是Map类型，可以省略中括号**

**（3）如果闭包是方法的最后一个参数，闭包可以置于圆括号外面**

apply(plugin: 'java')

apply plugin: 'java'

testCompile group: 'junit', name: 'junit', version: '4.10'

repositories() {

    mavenCentral()

}

task 'myTask' {

    doLast {

        println 'hello world!'

    }

}

分析：task()方法接受两个参数，一个字符串类型，一个闭包类型，传入的闭包调用了宿主的doLast方法，doLast方法接收一个闭包作为参数

**1.3 闭包相关**

**1.3.1 定义**

**形式1：def xxx = {paramters -> code}  //或者**

**形式2：def xxx = {无参数，纯code}  //这种case不需要->符号**

**形式3：def xxx = {->纯code}//这种情况下闭包不接受参数**

def aClosure = {//闭包是一段代码，所以需要用花括号括起来..

    String param1, int param2 ->  //这个箭头很关键。箭头前面是参数定义，箭头后面是代码

    println"this is code" //这是代码，最后一句是返回值，

   //也可以使用return，和Groovy中普通函数一样

}

def aClosure={

println"this is code"

}

**1.3.2 调用**

**（1）闭包对象.call(参数)  或者更像函数指针调用的方法：**

**（2）闭包对象(参数)**

**（3）如果闭包没定义参数的话，则隐含有一个参数，这个参数名字叫it，和this的作用类似。it代表闭包的参数。**

aClosure.call("this is string",100)  或者

aClosure("this is string", 100)

def greeting = { "Hello, $it!" }

def helloWorld = greeting(‘world’) //helloWorld ==”Hello,world!”

**注意：闭包的调用最复杂的在于其参数的确定，如果不看上下文，我们无法确定参数类型，参数**

**例如：List的each函数，如果不看定义，我们可能无法确定要传入一个什么类型的闭包，参数是什么，返回什么值，这些需要看源码进行确认**

1. def iamList = [1,2,3,4,5]  //定义一个List变量
2. iamList.each{ //调用它的each函数，只要传入一个Closure就可以了。
3. println it
4. }

**1.3.3 隐含变量**

it：默认的参数名，调用是如果没有传参数，it为null  
this : 跟Java一样，是定义闭包所在类的一个引用，不管有多少层闭包嵌套，this指向的都是最上层的类。  
owner : 封闭闭包的对象(如果只有一层闭包就是this，如果有多层闭包嵌套就是含有此闭包的上层闭包)  
delegate :缺省值是owner，但是可以改变，后面详说。

闭包可以设置方法和属性引用的模式(self是指 闭包内部定义属性)

使用函数setResolveStrategy(int type)

1 Closure.DELEGATE\_FIRST

//先引用self或者owner，没有则引用delegate的，如果没有则用this的，再没有就会报错

2 Closure.DELEGATE\_ONLY

//先引用self或者owner，没有则引用delegate的，再没有就会报错

3 Closure.OWNER\_FIRST（默认的模式）

//先引用self或者owner，没有则引用this的，再没有则引用delegate的，再没有就会报错

4 Closure.OWNER\_ONLY

//先引用self或者owner，没有则引用this的，再没有就报错

5 Closure.TO\_SELF //这种模式下，只能引用owner或者self的

Class class1{

def closure1{

def closure2{

def closure3{

}

}

}

}

对于闭包

closure1

this----->class1 owner----->class1 delegate------>class1

Closure2

this----->class1 owner----->closure1 delegate------>closure1

Closure3

this----->class1 owner----->closure2 delegate------>closure2

这里，delegate是默认owner一致的，除非被设置成其他的

Class C1{

def play(){

Println ‘in C1.play()’

}

}

Class C2{

def play(){

Println ‘in C2.play()’

}

def getPlayer(Closure closure){

closure.delegate=new C1();

//OWNER\_FIRST

//DELEGATE\_FIRST

//OWNER\_ONLY

//DELEGATE\_ONLY

//TO\_SELF

closure.setResolveStrategy(Closure.DELEGATE\_ONLY)

closure();

}

}

class C3{  
 def play={  
 println("in this.play()");  
 }  
 def outer={  
 def play={  
 println("in outer.play()");  
 }  
 C2 test=new C2();  
 def toPlay={  
// def play={  
// println("in self.play()");  
// }  
 //this c3 owner outer delegate c1  
 println("this "+this+" owner "+owner+" delegate "+delegate)  
 play();  
 }  
 toPlay.delegate=new C1()  
 test.getPlayer toPlay  
 }  
}

New C3().outer()

注意：outer.Play() outer.toplay.play()属于重定义，只能定义其中一个

那么按照以上的调用流程

**（1） 采用OWNER\_FIRST**

调用优先级(箭头代表上一个对象没有该属性或方法)：

1 如果定义的outer.toPlay.play()

Self > this > delegate > 报错 (self没有则引用this的在没有则引用delegate的)

2 定义的outer.play()

Owner > this > delegate > 报错

**（2） 采用DELEGATE\_FIRST**

调用优先级：

1 如果定义的outer.toPlay.play()

Self > delegate > this > 报错

2 定义的outer.play()

Owner > delegate > this > 报错

**（3） 采用OWNER\_ONLY**

调用优先级：

1 如果定义的outer.toPlay.play()

Self > this > 报错

2 定义的outer.play()

Owner > this > 报错

**（4） 采用DELEGATE\_ONLY**

调用优先级：

1 如果定义的outer.toPlay.play()

Self > delegate > 报错

2 定义的outer.play()

Owner > delegate > 报错

**（5） 采用TO\_SELF**

调用优先级：

1 如果定义的outer.toPlay.play()

Self > 报错

2 定义的outer.play()

Owner > 报错

**2 gradle相关知识**

2.1 gradle是什么

Gradle就是编译框架，跟CMake一样，他负责处理各种源文件文件之间的依赖关系，然后将源文件编译成各种输出文件，或许是APK或许是JAR或许是AAR。

**Gradle中，每一个待编译的工程都叫一个Project。每一个Project在构建的时候都包含一系列的Task。比**如一个Android APK的编译可能包含：

Java源码编译Task、资源编译Task、JNI编译Task、lint检查Task、打包生成APK的Task、签名Task等。

一个Project到底包含多少个Task，其实是由编译脚本指定的插件决定。插件是什么呢？插件就是用来定义Task，并具体执行这些Task的东西。

Gradle是一个框架，作为框架，它负责定义流程和规则。而具体的编译工作则是通过插件的方式来完成的。比如编译Java有Java插件，编译Groovy有Groovy插件，编译Android APP有Android APP插件，编译Android Library有Android Library插件

好了。到现在为止，你知道Gradle中每一个待编译的工程都是一个Project，一个具体的编译过程是由一个一个的Task来定义和执行的。

2.2 project task gradle

Gradle中每一个待编译的工程都是一个Project，拿一个android studio的项目做比喻，

可能有下面这样一个项目

-------------------------------------------------------

----testProj

------------build.gradle

------------setting.gradle

------------local.properties

------------app

------------------------build.gradle

------------app\_library

------------------------build.gradle

---------------------------------------------------

按照惯例，我们知道，一个android studio项目可能有多个moudle（app,app\_library）而这些moudle可能是可运行的APP项目(app)，可能是供其它moudle使用的一些库项目(app\_library)这些都是一个个独立的project,每一个project要求再其根目录下放置一个编译脚本，也就是build.gradle文件

因此，上面这个项目总共生成三个project(一个build.gradle生成一个project)

这是一个multiproject

testProj文件夹下的build.gradle的作用是配置其他子Project的。比如为子Project添加一些属性。这个build.gradle有没有都无所属。

testProj文件夹下的settings.gradle。这个文件很重要，名字必须是settings.gradle。它里边用来告诉Gradle，这个multiprojects包含多少个子Project

Setting.gradle的格式如下：

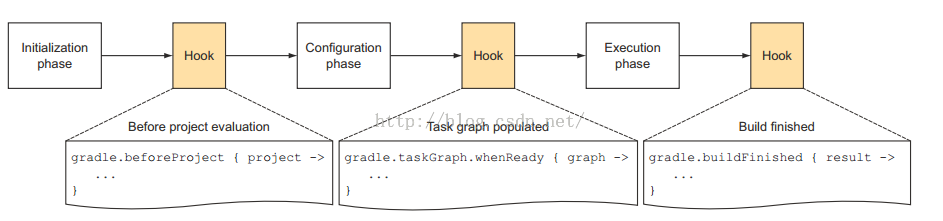
include ':app', ':app\_library'

查看project数目：在testProj目录下使用命令行 gradle projects

每个build.gradle里面会为project配置很多task

查看task数目：在对应项目根目录使用命令行gradle tasks

gradle工作流程



1 initialization阶段：解析setting.gradle文件

2 configuration阶段：解析各个项目的build.gradle文件，确认各个project以及project内的task之间的关系，**Configuration会建立一个有向图来描述Task之间的依赖关系**

3 execution阶段：执行编译任务