[Android Gradle] 搞定Groovy闭包这一篇就够了

HitenDev 关注

2 2017.04.07 14:17:37 字数 1,869 阅读 10,853

> 努力的人，应该像好色那样好学

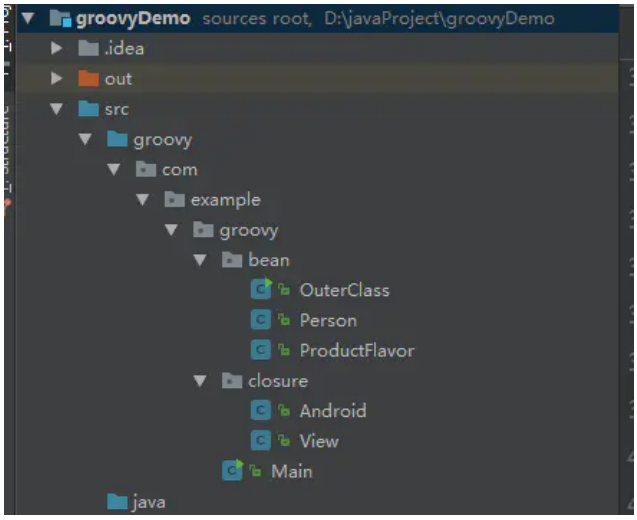
做Android开发的同学，对Gradle肯定不陌生，我们用它配置、构建工程，可能还会开发插件来促进我们的开发，我们必须了解Gradle，而不仅限于只会当配置构建工具，我想学习它，于是就有了这一系列的文章。

看到这篇博客，能可能会学到什么?

- 了解Groovy，我们并不需要精通，试着把它当java来写
- 理解Groovy闭包，闭包是必须要理解的，特别是delegate
- 利用闭包来实现自己的DSL

开发环境:

开发工具：Intellij Ideal
jdk版本：jdk1.8
sdk版本：groovy sdk 2.4.10
具体用Ideal创建groovy的教程，比较简单，自行摸索



工程如下

Groovy语法

其实，关于Groovy语法和DSL的介绍，网上的博客比较多，我也不想在这里做过多的介绍，我推荐阅读阿拉神农的这篇博客，建议大家看完邓老师的博客，再回来接着我的闭包来看，要不然，你可能还会不理解。

闭包的理解

闭包（Closure）是很多编程语言中很重要的概念，那么Groovy中闭包是什么，官方定义是"Groovy**中的闭包是一个开放，匿名的代码块，可以接受参数，返回值并分配给变量**"，简而言之，他说一个匿名的代码块，可以接受参数，有返回值，那么到底是怎么样的，我们来探究一下：

- 如何定义

> 定义闭的语意：{ [closureParameters ->] statements }

其中[closureParameters->]代表参数们，多参数用逗号分割，用->隔开参数与内容，没有参数可以不写->

- 闭包的写法

```
//执行一句话
{ printf 'Hello World' }

//闭包有默认参数it，且不用申明
{ println it }

//闭包有默认参数it，申明了也无所谓
{ it -> println it }

// name是自定义的参数名
{ name -> println name }

 //多个参数的闭包
{ String x, int y ->
    println "hey ${x} the value is ${y}"
}

```

- groovy.lang.Closure**对象**

其实，每定义的闭包是一个Closure对象，我们可以把一个闭包赋值给一个变量

```
def innerClosure = {
    printf("hello")
 }

def hello = { String x ->
    printf("hello ${x}")
 }

```

我们把闭包作为方法的参数类型

```
void setOnClickListener(Closure closure) {
        this.onClickListener = closure
    }
```

如何执行闭包对象呢，执行闭包对象有两种，一是直接用括号+参数，二是调用call方法

```
//执行innerClosure 闭包
innerClosure ()
//or
innerClosure.call()

//带参数的闭包
hello("world")
//or
hello.call("world")

```

- **理解闭包内**this，owner，delegate**对象**

在闭包内部，有三个内置对象this，owner，delegate，我们可以直接this，owner，delegate调用，或者用get方法：

- getThisObject() 等于 this
- getOwner() 等于 owner
- getDelegate() 等于delegate

那么这三个对象，分别指代的是哪个对象，是否和java的this关键字一样，我们先做文字解释：

- this 对应于定义闭包的那个类，如果在内部类中定义，指向的是内部类
- owenr 对应于定义闭包的那个类或者闭包，如果在闭包中定义，对应闭包，否则同this一致
- delegate 默认是和owner一致，或者自定义delegate指向
  我们来用代码验证一下

```
class OuterClass {
    class InnerClass {
        def outerClosure = {
            def innerClosure = {
            }
            printfMsg("innerClosure", innerClosure)
            println("------")
            printfMsg("outerClosure", outerClosure)
        }
        void printfMsg(String flag, Closure closure) {
            def thisObject = closure.getThisObject()
            def ownerObject = closure.getOwner()
            def delegate = closure.getDelegate()
            println("${flag} this: ${thisObject.toString()}")
            println("${flag} owner: ${ownerObject.toString()}")
            println("${flag} delegate: ${delegate.toString()}")
        }
    }

    def callInnerMethod() {
        def innerClass = new InnerClass()
        innerClass.outerClosure.call()
        println("------")
        println("outerClosure toString ${innerClass.outerClosure.toString()}")
    }

    static void main(String[] args) {
        new OuterClass().callInnerMethod()
    }
}
```

我们在OuterClass定义一个内部类InnerClass，在InnerClass中定义了一个outerClosure闭包，在outerClosure中定义了一个innerClosure闭包，现在我们分别打印innerClosure和outerClosure闭包对应的this，owner，delegate对象和outerClosure对象的toString方法

```
innerClosure this: com.example.groovy.bean.OuterClass$InnerClass@e874448
innerClosure owner: com.example.groovy.bean.OuterClass$InnerClass$_closure1@5bfbf16f
innerClosure delegate: com.example.groovy.bean.OuterClass$InnerClass$_closure1@5bfbf16f
------
outerClosure this: com.example.groovy.bean.OuterClass$InnerClass@e874448
outerClosure owner: com.example.groovy.bean.OuterClass$InnerClass@e874448
outerClosure delegate: com.example.groovy.bean.OuterClass$InnerClass@e874448
------
outerClosure toString com.example.groovy.bean.OuterClass$InnerClass$_closure1@5bfbf16f

```

分析结果：

- innerClosure
- this：结果是OuterClass$InnerClass对象
- owner：结果是OuterClass$InnerClass$_closure1对象 ， 即outerClosure
- delegate：同owenr
- **outerClosure **
- this：结果是OuterClass$InnerClass对象
- owner：结果是OuterClass$InnerClass对象
- delegate：同owenr

this，owner ， delegate指向总结：
this 永远是指定义该闭包类，如果存在内部类，则是最内层的类，但this不是指当前闭包对象
owenr 永远是指定义该闭包的类或者闭包，顾名思义，闭包只能定义在类中或者闭包中
** delegate** 默认是指owner，可以自己设置，自己设置的话又是什么情况

- delegate**才是重头戏**

前面已经说了，闭包可以设置delegate对象，**设置**delegate**的意义就是讲闭包和一个具体的对象关联起来**，这个如何理解，看代码：

```
# Person.groovy
class Person {
    String name
    int age

    void eat(String food) {
        println("你喂的${food}真难吃")
    }

    @Override
    String toString() {
        return "Person{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}'
    }
}

# Main.groovy
    def cc = {
        name = "hanmeimei"
        age = 26
     }
```

我们定义Person实体类，再定义一个名字叫cc的闭包，我们想在闭包里修改Person的name和age，还想调用eat方法，这个怎么关联起来？

```
cc.delegate = person
cc.call()
```

怎么，这样就ok了吗，是的，就是这么简单，完整代码

```
class Main {
    def cc = {
        name = "hanmeimei"
        age = 26
        eat("油条")
        eat "油条"
    }
    static void main(String... args) {
        Main main = new Main()
        Person person = new Person(name: "lilei", age: 14)
        println person.toString()

        main.cc.delegate = person
        main.cc.call()
        println person.toString()
}
}

#打印结果
Person{name='lilei', age=14}
你喂的油条真难吃
Person{name='hanmeimei', age=26}
```

上面我们知道了，在闭包中可以访问被代理对象的属性和方法，哦，那么我还有一个疑问，如果闭包所在的类或闭包中和被代理的类中有相同名称的方法，到底要调用哪个方法，其实这个问题groovy肯定考虑到了，为我们设定了几个代理的策略：

- Closure.OWNER_FIRST是默认策略。优先在owner寻找，owner没有再delegate
- Closure.DELEGATE_FIRST：优先在delegate寻找，delegate没有再owner
- Closure.OWNER_ONLY：只在owner中寻找
- Closure.DELEGATE_ONLY：只在delegate中寻找
- Closure.TO_SELF：暂时没有用到，哎不知道啥意思

为了验证，我们现在修改一下Main.groovy代码

```
class Main {
    void eat(String food){
        println "我根本不会吃，不要喂我${food}"
    }
    def cc = {
        name = "hanmeimei"
        age = 26
        eat("油条")
    }

    static void main(String... args) {
        Main main = new Main()
        Person person = new Person(name: "lilei", age: 14)
        println person.toString()

        main.cc.delegate = person
//        main.cc.setResolveStrategy(Closure.DELEGATE_FIRST)
        main.cc.setResolveStrategy(Closure.OWNER_FIRST)
        main.cc.call()
```

```
        println person.toString()
}
}

```

我们在Main中也定义了同名的方法eat（food）,因为当前cc闭包的owner正是Main对象，我们通过调用setResolveStrategy方法，修改策略，发现结果和预期的一致

闭包delegate**的基本概念已经讲完，看完这些，相信你能进一步理解**android**开发中**build.gradle**中的写法，已经到达本次学习的目的，下面我们练习一下：**

闭包练习

实现一个回调接口

做Android开发的同学对回调接口肯定不陌生，特别是事件的监听，现在，我们仿View.setOnClickListener用来闭包来实现一个回调接口

```
class View {
    private Closure onClickListener
    Timer timer

    View() {
        timer = new Timer()
        timer.schedule(new TimerTask() {
            @Override
            void run() {
                perOnClick()
            }
        }, 1000, 3000)
    }
    void setOnClickListener(Closure closure) {
        this.onClickListener = closure
    }
    private void perOnClick() {
        if (onClickListener != null) {
            onClickListener(this)
        }
    }
    @Override
    String toString() {
        return "this is view"
    }
}
```

定义一个View类，用Timer计时模拟事件的触发，暴露setOnClickListener方法，用于介绍闭包对象，那么调用者该如何写呢?

```
 View view = new View()
 view.setOnClickListener { View v ->
       println v.toString()
  }
```

其实调用者也很简单，只需定义一个闭包，v是传递过来的参数，打印出toString方法，结果如下

```
this is view
this is view
this is view
...
```

仿照Android DSL 定义闭包

在Android中我们熟悉的build.gradle配置，其实也是闭包，这下面肯定是你熟悉的代码

```
android {
    compileSdkVersion 25
    buildToolsVersion "25.0.2"

    defaultConfig {
        minSdkVersion 15
        targetSdkVersion 25
        versionCode 1
        versionName "1.0"
    }
}
```

我们要实现自己的闭包，我们要定义两个实体类Android.groovy和ProductFlavor.groovy，代码如下

```
# Android.groovy
class Android {
    private int mCompileSdkVersion
    private String mBuildToolsVersion
    private ProductFlavor mProductFlavor

    Android() {
        this.mProductFlavor = new ProductFlavor()
    }

    void compileSdkVersion(int compileSdkVersion) {
        this.mCompileSdkVersion = compileSdkVersion
    }

    void buildToolsVersion(String buildToolsVersion) {
        this.mBuildToolsVersion = buildToolsVersion
    }

    void defaultConfig(Closure closure) {
        closure.setDelegate(mProductFlavor)
        closure.setResolveStrategy(Closure.DELEGATE_FIRST)
        closure.call()
    }

    @Override
     String toString() {
        return "Android{" +
                "mCompileSdkVersion=" + mCompileSdkVersion +
                ", mBuildToolsVersion='" + mBuildToolsVersion + '\'' +
                ", mProductFlavor=" + mProductFlavor +
                '}'
    }
}

# ProductFlavor.groovy
class ProductFlavor {
    private int mVersionCode
    private String mVersionName
    private int mMinSdkVersion
    private int mTargetSdkVersion

    def versionCode(int versionCode) {
        mVersionCode = versionCode
    }

    def versionName(String versionName) {
        mVersionName = versionName
    }
```

```
    def minSdkVersion(int minSdkVersion) {
        mMinSdkVersion = minSdkVersion
    }


    def targetSdkVersion(int targetSdkVersion) {
        mTargetSdkVersion = targetSdkVersion
    }

    @Override
    String toString() {
        return "ProductFlavor{" +
                "mVersionCode=" + mVersionCode +
                ", mVersionName='" + mVersionName + '\'' +
                ", mMinSdkVersion=" + mMinSdkVersion +
                ", mTargetSdkVersion=" + mTargetSdkVersion +
                '}'
    }
}

```

这两个实体，相当于闭包的被代理对象，那么我们闭包怎么写呢

```
//闭包定义
def android = {
        compileSdkVersion 25
        buildToolsVersion "25.0.2"
        defaultConfig {
            minSdkVersion 15
            targetSdkVersion 25
            versionCode 1
            versionName "1.0"
        }
    }

//调用
Android bean = new Android()
android.delegate = bean
android.call()
println bean.toString()

//打印结果
Android{mCompileSdkVersion=25, mBuildToolsVersion='25.0.2', mProductFlavor=ProductFlavor{mVers
```

结果很明显，闭包申明的值，赋给了两个实体对象Android和ProductFlavor，这种从闭包到具体类的代理过程，才是闭包最魅力的地方所在。

闭包语义解析

在闭包中，访问代理对象的属性，用"="符合，访问代理对象的方法，用"()"或者空格，如果方法参数类型是Closure类型，可以直接用大括号申明闭包，就像android下的defaultConfig 一样。。。

关于代码

本次所有代码均上传在github上可供参考，地址[https://github.com/final-ly/groovyDemo]{https://github.com/final-ly/groovyDemo}，谢谢阅读！