# BuildConfig文件

* 在app/build/generated/source/buildConfig文件夹下面
* 使用： buildConfigField "String" , "key" , "\"value\""  
  配置buildConfig中的属性参数 String ： 参数类型（int,boolean...）, key : 属性的名字， value ： 属性的值， \ 为转义字符

# resValue 动态修改工程资源

* 和buildConfig 类似， resValue("string","key","value")
* 其中string 表示 会在 app/build/generated/res/resValue/.../generated.xml中生成对应的String 的 key 和Value， 代码中可以直接getResources().getString(R.string.key);获取到value
* **注意：** 因为Gradle编译的时候会将脚本配置和string.xml文件中配置进行merge，所以string.xml中已经存在的key在编译前要删除，否则会报错

//下面模拟在不同渠道下修改资源参数

productFlavors{

baidu{

buildConfigField "String" , "productCode" , "\"baidu 1.0\"" resValue("string","productName","baidu")

}

}

# 全局属性配置

## 在project : build.gradle 的buildscript中声明ext和自定义属性，然后在其他module中就可以直接使用这个属性了

ext{  
 version\_code =1  
 version\_name =**"1.0"**}

modul中调用方式如下：

defaultConfig {  
 applicationId **"com.scsowing.decoration"** minSdkVersion 16  
 targetSdkVersion 28  
 versionCode version\_code  
 versionName version\_name  
*// flavorDimensions "1"* testInstrumentationRunner **"android.support.test.runner.AndroidJUnitRunner"** }

## 新建自己的Gradle属性文件 text.gradle ， 然后在Project中引入，最后就可以在其他的module中直接使用

**text.gradle 编写如下：**

ext {

kotlinVersion = "1.1.0"

rxjavaLibVersion = "1.2.0"

}

**Project : build.gradle** 中引入，然后才可以在moduel中使用 //就是引入他的相对根目录路径 apply from: 'text.gradle'

**Module : build.gradle** 中使用 rootProject.ext.XXXXX

applicationId rootProject.ext.applicationId  
minSdkVersion rootProject.ext.min\_version

# 4.defaultConfig 基本配置信息

*//默认配置*defaultConfig {  
 *//包名* applicationId **"com.rayhahah.gradledemo"** *//最低版本* minSdkVersion 19  
 *//目标版本* targetSdkVersion 25  
 *//版本代码* versionCode getVersinCode()  
 *//版本* versionName **"1.0"** *//自动化测试* testInstrumentationRunner **"android.support.test.runner.AndroidJUnitRunner"** resValue **"int"**,**"test"**,**"1"**}

# 5.signConfigs签名配置信息

signingConfigs {  
 *//debug模式签名文件* debug {}  
 *//签名打包* release {  
 *//签名文件所在路径* storeFile file(**"ray.jks"**)  
 *//签名密码* storePassword **"111111"** *//别名* keyAlias **"rayhahah"** keyPassword **"111111"** }  
 *//自定义签名配置* ray{  
 *//和上面的属性一致，根据个人需求实现不同配置* }  
}

# 6.buildTypes

编译类型 ： 指定编译不同类型情况下的不同配置信息，这里是列举了部分属性和方法，全部的方法和属性请看

applicationVariants.all { variant ->  
 variant.outputs.all { output -> *// each 改为 all* **def** fileName = System.*currentTimeMillis*() + **"\_"** + **"**${variant.buildType.name}**\_**${variant.productFlavors[0].name}**\_**${variant.versionName}**.apk"  
 def** outFile = output.outputFile  
 **if** (outFile != **null** && outFile.name.endsWith(**'.apk'**)) {  
 outputFileName = fileName *// output.outputFile 改为 outputFileName* }  
 }  
}

# 7.sourceSets

sourceSets{  
 main {  
 manifest.srcFile **'src/main/AndroidManifest.xml'** java.srcDirs = [**'src/** main **/java'**]  
 resources.srcDirs = [**'src/main/resources'**]  
 aidl.srcDirs = [**'src/main/aidl'**]  
 renderscript.srcDirs = [**'src/maom'**]  
 res.srcDirs = [**'src/main/res'**]  
 assets.srcDirs = [**'src/main/assets'**]  
 jniLibs.srcDir **'src/main/jniLibs'** }

//下面这一部分可以是根据项目多渠道配置不同资源路径方式，注意如果是java.srcDirs需要配置不同的文件，main下面和渠道下面不能存在相同的类，只能都存在于渠道名的文件下面  
 offline{  
 res.srcDirs = [**'src/offline/res'**]  
 manifest.srcFile **'src/main/AndroidManifest.xml'** java.srcDirs = [**'src/offline/java'**]  
 resources.srcDirs = [**'src/main/resources'**]  
 aidl.srcDirs = [**'src/main/aidl'**]  
 renderscript.srcDirs = [**'src/maom'**]  
 assets.srcDirs = [**'src/main/assets'**]  
 jniLibs.srcDir **'src/main/jniLibs'** }

也可如下方式制定资源路径  
*// offline.res.srcDirs = ['src/main/res-offline']  
// online.res.srcDirs = ['src/main/res-online']  
  
// offline.res.srcDirs = ['src/offline/res']  
// online.res.srcDirs = ['src/online/java']* }

# 8.Gradle忽略文件设置

创建.gitignore，添加忽略的文件，添加忽略之后，已经提交到版本库中的文件是无法忽略的。只能clone到本地，删除后，再进行忽略。

.gitignore只能忽略那些原来没有被track的文件，如果某些文件已经被纳入了版本管理中，则修改.gitignore是无效的。

 正确的做法是在每个clone下来的仓库中手动设置不要检查特定文件的更改情况。  
 git update-index --assume-unchanged PATH    在PATH处输入要忽略的文件