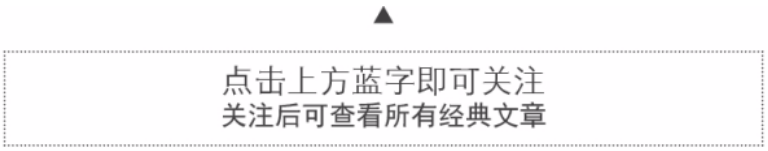
Gradle妙用，统一化自动依赖管理

原创： Young方远



小米集团公告，1月18日斥资约1亿港元回购984.96万B类普通股；小米于17日回购614万股B类股，耗资约6000万港元。在一年的限售期到期之后，更多的股票将在今年7月份解锁出售，而小米的一些控股股东（包括雷军）本月表示，他们将继续持有一年的股份。

本篇来自 Young方远 的投稿文章。文章对高效的dle variant使用知识进行了不错的讲解，希望对大家有所帮助。

Young方远 的博客地址：

https://blog.csdn.net/qxf5777404

如果你也在做着同一套代码，构建多个项目的需求，那么一定要浏览下，或许会带给你启发.清晰化的目录结构，统一化的自动依赖管理。

入坑以来一直和variant打着交道，最初15年还是eclipse开发，那是还没variant概念。当时的项目是企业级app开发，简而言之就是一套代码针对不同企业构建其对应请求地址的包，当然不同企业也会有差异化的需求但大致的业务流程都差不多。(ps:如果是你，你会怎么做？一家企业一套代码？这种想法最初就被否了，因为企业多了，以后增加/修改公共需求都是问题，绝对让你崩溃!)。所以当时的处理只是从代码层面来区分的，当然劣势也有，可能会增加包的大小。再往后推，谷歌爸爸推出了AS正式版，果断拥抱。再后来，了解到了Gradle的Flavor配置。可以根据Flavor从项目结构上选择依赖文件。这不就完美从项目配置上满足了我的需求么。随后就是更改重构的过程~以下只是我的经验心得，献给可能需要的你。

重要性

无论是从项目的健壮性还是可维护性来说好的开发工具搭配项目架构能让你事半功倍。

统一化管理：我一直有统一配置的习惯,无论是代码的配置还是gradle的依赖。这样在项目变动的时候，更改一处即可。想想，假如不这么做，漏该了一处，什么结果...

项目结构

productFlavors:相信大家都了解，用于构建变体。我们可以用来打多渠道包，也可以做一些资源/代码差异化变更。这里我们将充分的依靠它完成后续工作！

- **差异化的实现有两种方式**

首先我创建了两个Flavor:

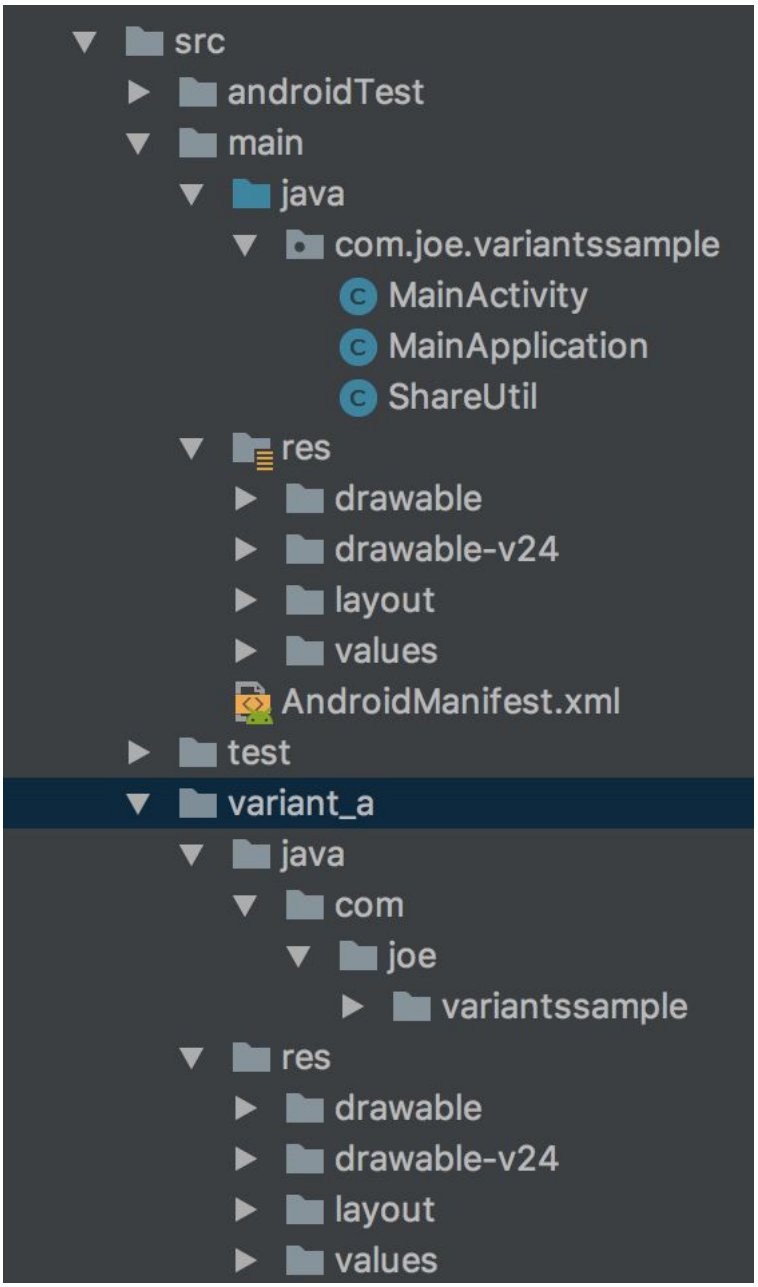
```
productFlavors {
    variant_a {
        applicationId "com.joe.variant_a"
```

```
    }

    variant_b {

        applicationId "com.joe.variant_b"

    }

}
```

第一种

直接在app-module的src下建立对应的flavor名相同的文件夹，并创建相应的java/res文件即可。

如图:



这种是常见的目录结构。也满足了大多的需求，差异化的主题色/代码等都满足了。但是对于想大做文章的就不一定满足需求了，比如:

- 依赖性差异

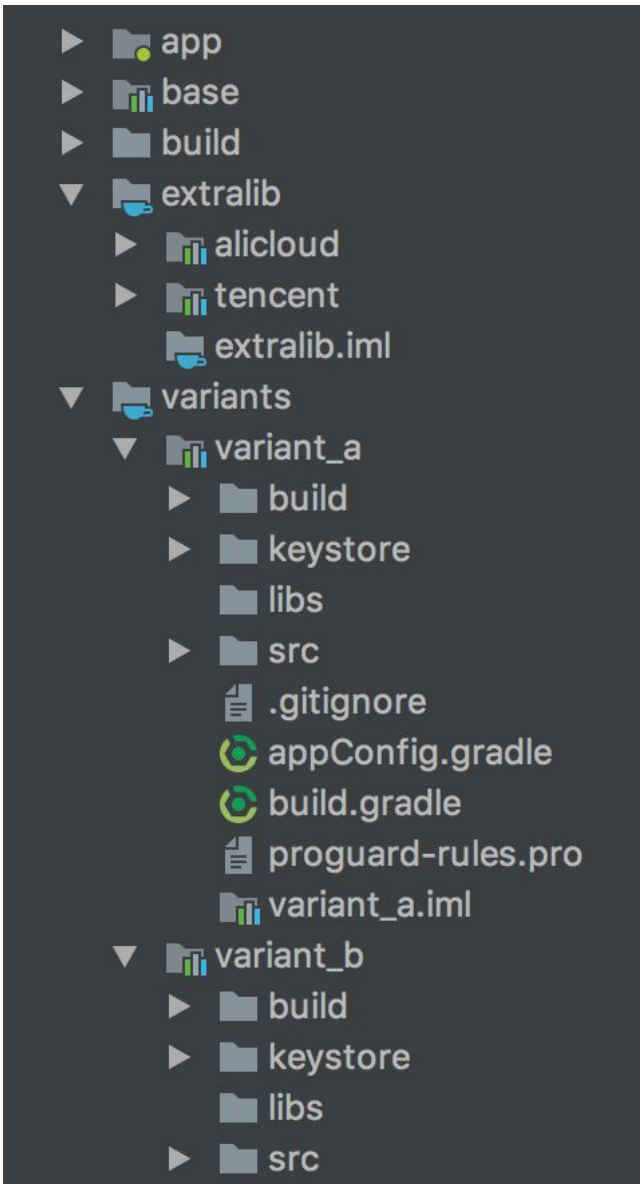
项目A需要添加实现一个扫码功能，而B则不需要。见多识广的可能说，创建扫码的module可以用aImplementation来选择性依赖，不错可以解决。起初我也这么做的。那么问题又来了：B需要依赖支付组件而A又不需要，行呗，接着bImplementation呗，有没有想过如果你的Flavor不止这两个有N多个，而差异化的依赖又有N项...再这么在dependencies {...}中写下去，看着多痛苦啊。

- **业务性差异**

同一个模块，A和B却是不同的业务那你会怎么做?if else来判断?然后在app中处理? 那么问题来了：这种需求贼多，而且不小(一个方法搞不定)也不大(用组件化区分又没必要)。那怎么办?我的想法:运用设计模块，区分不同的Flavor，接口抽象业务处理方法。app层差异性业务触发接口。Flavor层来各自处理。(ps:由于时间关系，Sample中未做此方案，后续会添加。但我觉得大家都能理解，也可能有更好的解决方法。

第二种

将Flavor作为module来处理



看图可以清晰的大致看出:app负责主要的业务包装职能。base:基准的依赖。extralib:三方的依赖。variants:我们所需的变体项目。每个变体依赖各自所需的组件。并处理各自的业务。接下来就是要让它能够统一化自动依赖的管理了。

大功臣Groovy

首先思考一下我们的理想效果:在vatiants中进行各自的配置后。无论是debug还是release都能根据当前编译的flavor来自动将其添加为app的依赖，并找到对应的配置文件和keystore来编译，也就是我们首先就是要知道如何才能拿到当前编译的flavor。很遗憾Gradle并没有提供该方法，但感谢stackoverflow上的大神提供思路，我们可以通过其他方法获得。

- **拿到currentFlavor**

我们在项目的gradle中写下该语句:

```
Gradle gradle = getGradle()
```

```
String taskName = gradle.getStartParameter().getTaskNames()[0]

println "taskName : " + taskName
```

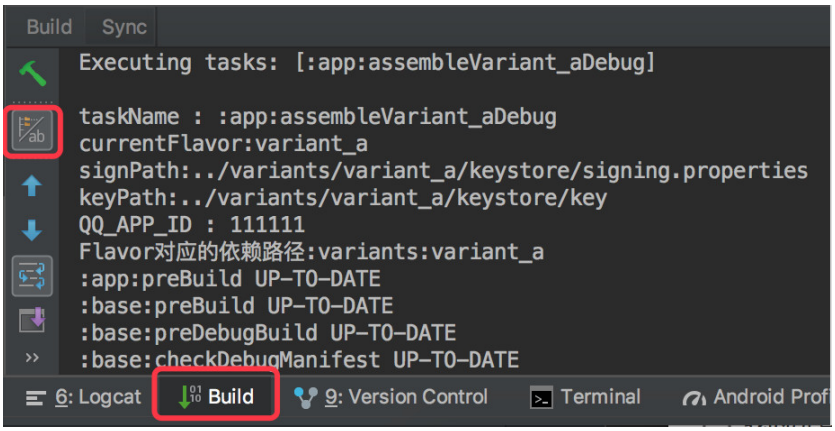
运行后我们可以看到日志:

```
taskName : :app:assembleVariant_aDebug
currentFlavor:variant_a
```

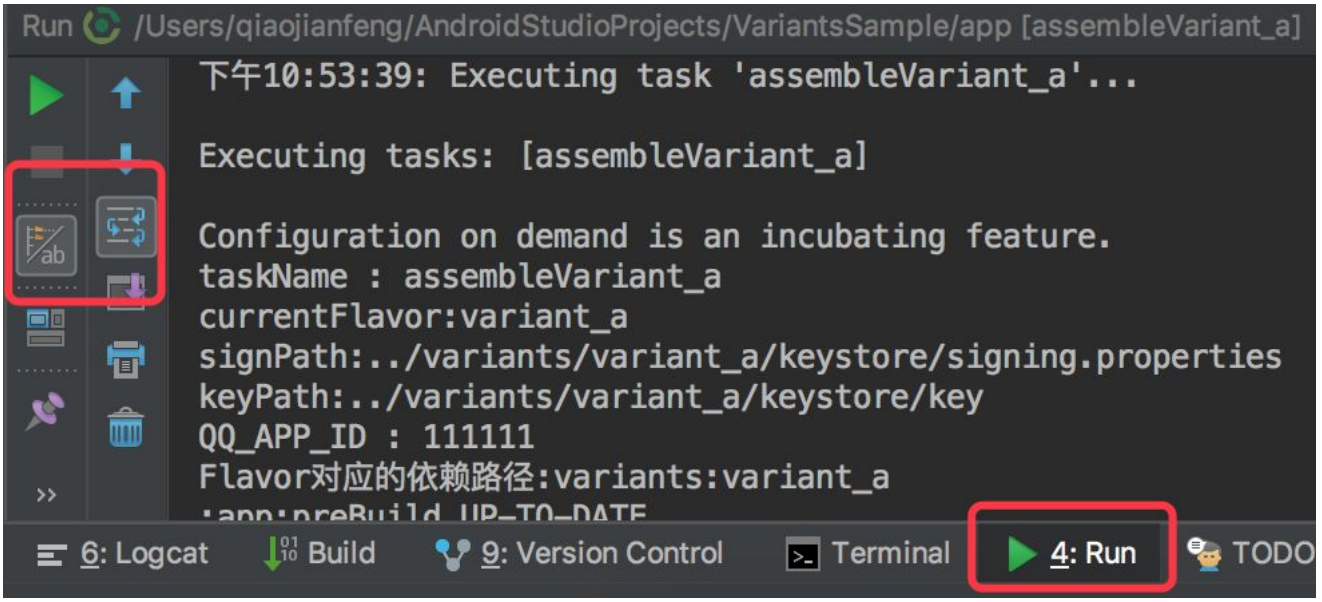
然后我们，针对当前编译的taskName处理下。就拿到了currentFlavor:variant_a.

运行。这里要说下运行如何看打印的日志。

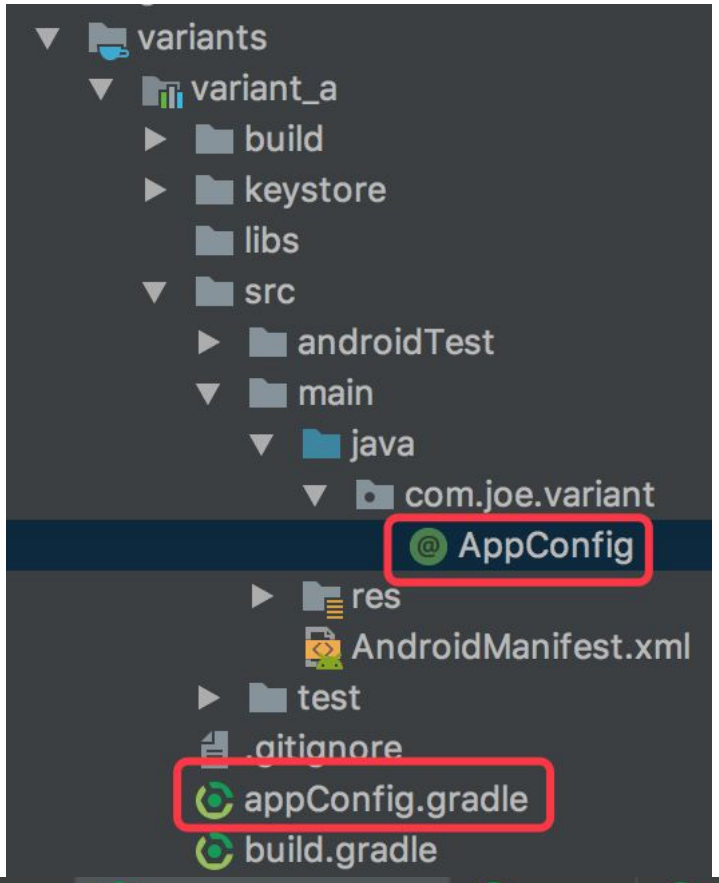
debug情况下(也就是直接运行项目):



assemble情况下(命令行选择构建情况下):可以直接看到



为了能够根据currentFlavor自动化获取配置路径 先看下Flavor中的配置:




```
ext {
    vatiant = [
            app: 'V-Sample-A',
            applicationId: 'com.joe.variant_a',
            versionCode: 11,
            versionName: '1.1.1',
            serverUrl: '"https://www.google.com"'
    ]
    tencent = [
            //注意这里不能将其设置为String类型,因为清单文件中
            QQ_APP_ID: '111111'
    ]
}
```

可以看到，我们在每个Flavor中都设置了其私有的appConfig文件。在里面定义了其对应的app信息。和三方依赖不得不在gradle中定义的参数(不必要的直接在java文件.AppConfig中定义)。这里说下如何引用appConfig.很简单，直接在对应的build.gradle中直接apply from:即可。但我们要实现自动化，所以就要动态化路径。

```
//获取对应variant目录下的私有特殊appConfig.gradle文件

apply  from:  String.format('../variants/%1$s/appConfig.gradle',  rootProject.ext.currentFlavor)
```

这样我们就可以动态的拿到当前编译的Flavor的appConfig。这样就可以愉快的配置了。

- **动态配置项目参数**

```
defaultConfig {
    applicationId "com.joe.variantssample"
      applicationId this.ext.vatiant.applicationId

    minSdkVersion rootProject.ext.minSdkVersion
    targetSdkVersion rootProject.ext.targetSdkVersion

    versionCode this.ext.vatiant.versionCode
    versionName this.ext.vatiant.versionName

    flavorDimensions "dimension"

    testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"

    buildConfigField "String", "SERVER_URL", "${this.ext.vatiant.serverUrl}"
}
```

```
productFlavors {
    variant_a {
        applicationId "com.joe.variant_a"
    }
    variant_b {
        applicationId "com.joe.variant_b"
    }
}
```

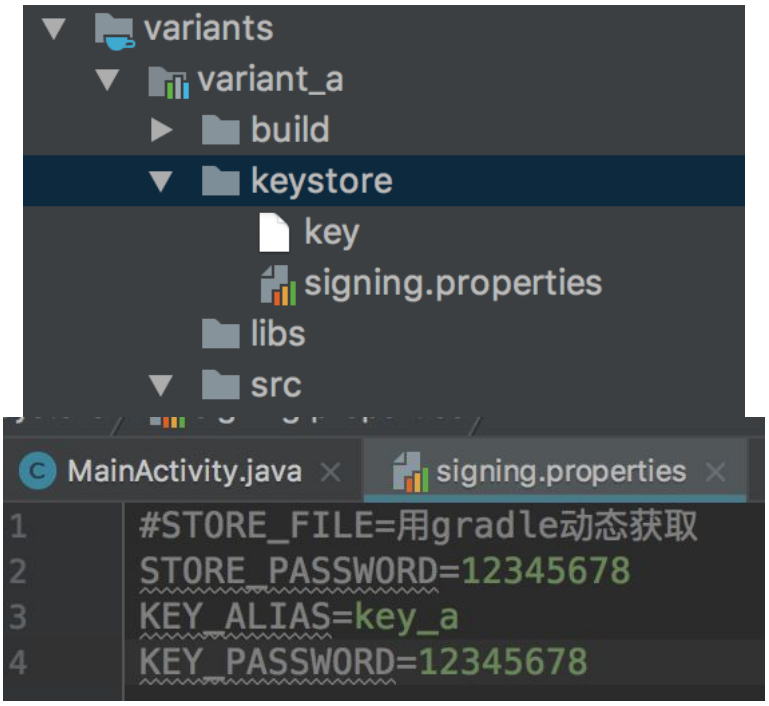
这里简单解释一下:rootProject.ext获取的是根module下gradle内的ext中定义的参数 this.ext则指的是引用的appConfig。而且大家也可以看到我很省事儿的直接将应该在Flavor中定义的versionCode.versionName.SERVER_URL都直接写在了defaultConfig中了。只有applicationId还在Flavor中定义(ps.applicationId不能在defaultConfig中,debug编译有可能会出现先后编译错误.可在项目中查看)。

- **动态化依赖 Flavor**

很关键的一步:直接在dependencies中即可，以后随便你加多少Flavor都不需要再维护。

```
implementation project(':variants:' + rootProject.ext.currentFlavor)
```

动态化签名文件配置



每个Flavor有自己的签名。这样才能...(你懂得) 下面配置Flavor自己的签名(比较重要，请自行睁大眼查看~)首先，让gradle可以自动引入对应Flavor的签名：

```
//获取当前编译flavor对应路径下的sign
```

```
final String signPath = String.format('../variants/%1$s/keystore/signing.properties', rootProject.ext.currentFlavor)

final String keyPath = String.format('../variants/%1$s/keystore/key', rootProject.ext.currentFlavor)
```

然后，需要为其配置独立的签名验证：

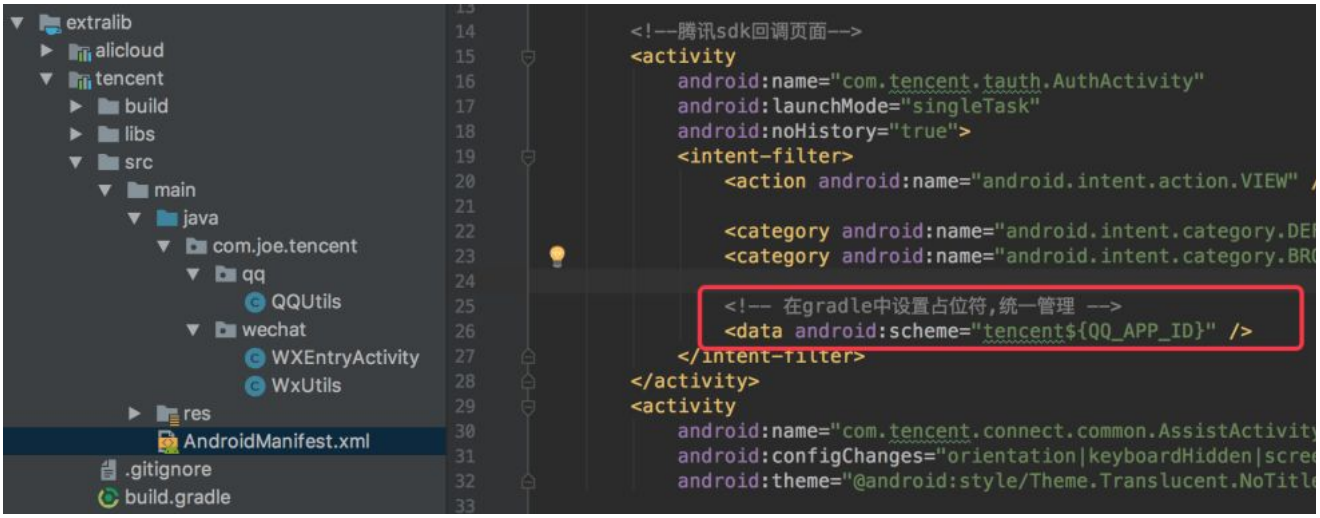
```
signingConfigs {
    println "signPath:" + signPath
    println "keyPath:" + keyPath
    variant_a {
        File keyFile = file(signPath)
        Properties localProps = new Properties()
        localProps.load(new FileInputStream(keyFile))
        storeFile file(keyPath)
        storePassword localProps["STORE_PASSWORD"]
        keyAlias localProps["KEY_ALIAS"]
        keyPassword localProps["KEY_PASSWORD"]
    }
    variant_b {
        File keyFile = file(signPath)
        Properties localProps = new Properties()
        localProps.load(new FileInputStream(keyFile))
        storeFile file(keyPath)
        storePassword localProps["STORE_PASSWORD"]
        keyAlias localProps["KEY_ALIAS"]
        keyPassword localProps["KEY_PASSWORD"]
    }
}
```

- **三方依赖的特殊处理**

普通的依赖，我们正常操作就可以了。但是有些特殊的三方需要在Manifest内定义appId(比如:QQ分享).。好，我们看下如何处理：前面可以看到，我们在appConfig.gradle中已经设置了QQ_APP_ID，现在我们在tencent的module中获取到。

```
//获取对应variant目录下的私有特殊appConfig.gradle文件

apply from: String.format('../../variants/%1$s/appConfig.gradle', rootProject.ext.currentFlavor)

//然后給Manifest设置占位符

manifestPlaceholders = [QQ_APP_ID: this.ext.tencent.QQ_APP_ID]
```

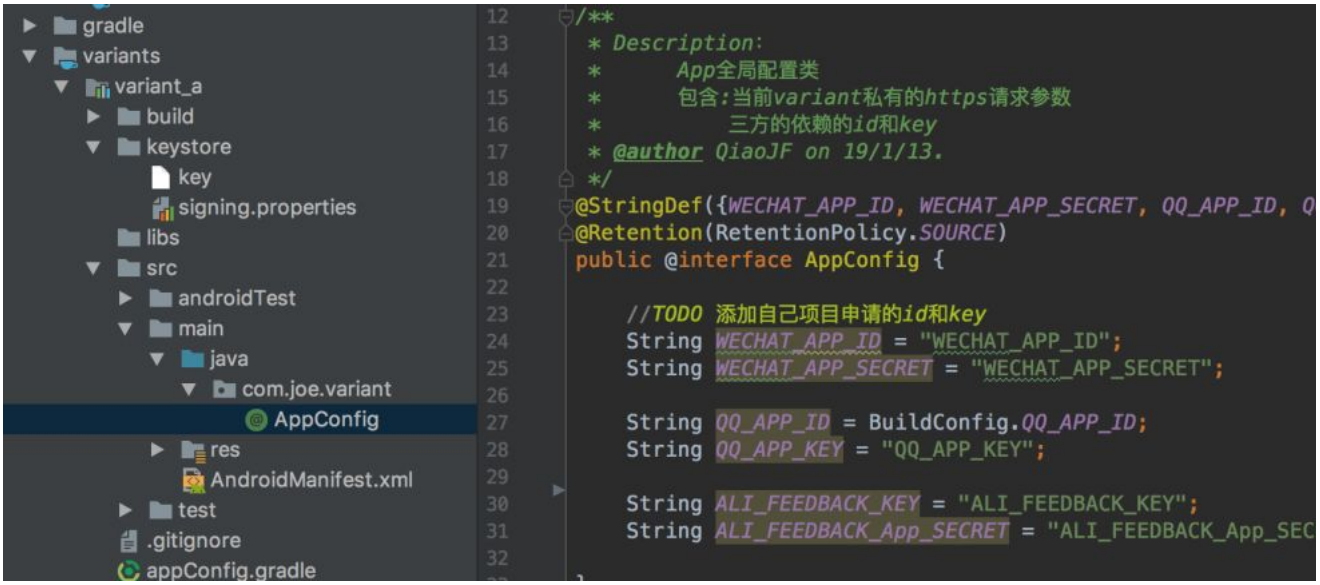
然后在tencent的Manifest中：



这样就依旧可以保证全局只依赖appConfig中的QQ_APP_ID。

- **创建Flavor的AppConfig文件**

现在基本完成了，配置工作。就差Flavor数据的设置了。剩下的使用我想就不用我说了吧~(ps:自行查看路径)



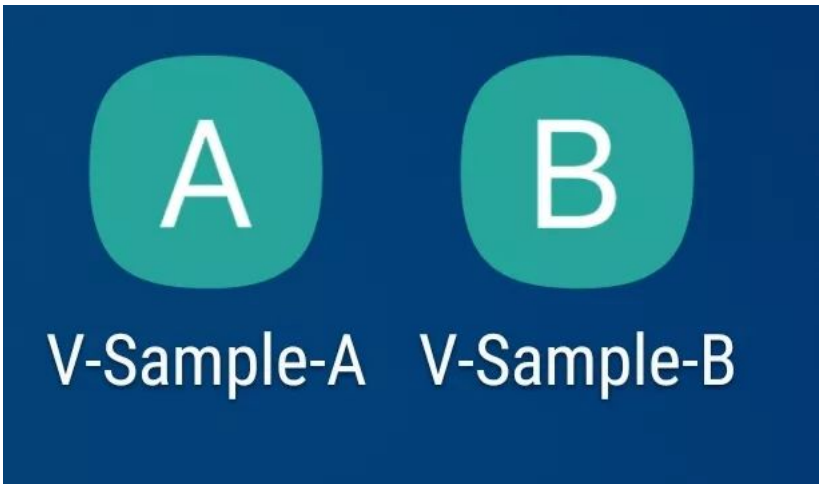
Tip

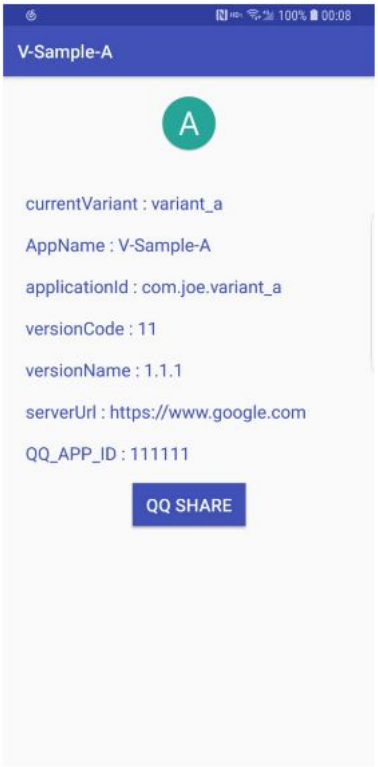
1. 构建:必须构建单一Flavor才行。直接assembleRelease是不行的，拿不到currentFlavor(如有大神知道欢迎告知)。
2. 虽然是完工了，但是一路走下来坑还是略多。加班花了个通宵才完工。你需要一些Groovy的基本语法，比如闭包的特性，以及路径的获取 .\: 当前目录之下, ..\:上级目录之下。
3. 在使用的时候也有很多需要注意的地方。比如flavor的命名和Flavor的包名应一样等一些问题。
4. Sample中我创建了完整的项目，后续也会更新维护。里面也详细了注释。欢迎查看，建议。

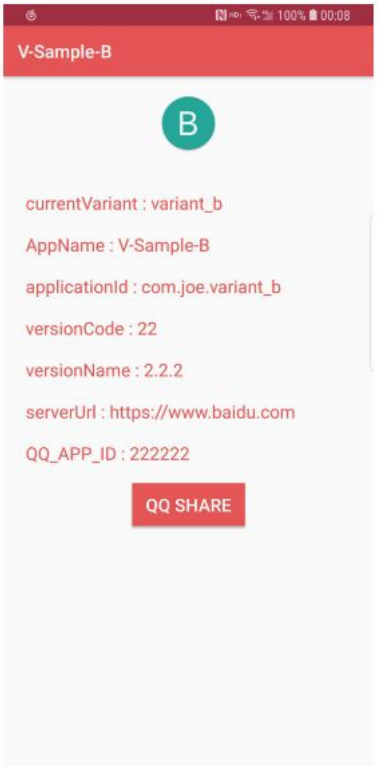
项目地址为:

https://github.com/Young-Joe/VariantsSample

附图为我实现的Sample来打出的两个差异性Release包:


A
B
V-Sample-A
V-Sample-B


V-Sample-A
A
currentVariant : variant_a
AppName : V-Sample-A
applicationId : com.joe.variant_a
versionCode : 11
versionName : 1.1.1
serverUrl : https://www.google.com
QQ_APP_ID : 111111
QQ SHARE


V-Sample-B
B
currentVariant : variant_b
AppName : V-Sample-B
applicationId : com.joe.variant_b
versionCode : 22
versionName : 2.2.2
serverUrl : https://www.baidu.com
QQ_APP_ID : 222222
QQ SHARE

阅读原文