

Walle多渠道打包

介绍

Android 7.0 引入一项新的应用签名方案 **APK Signature Scheme v2**，它能提供更快应用安装时间和更多针对未授权 **APK** 文件更改的保护。在默认情况下，**android Studio 2.2** 和 **Android Plugin for Gradle 2.2** 会使用 **APK Signature Scheme v2** 和传统签名方案来签署您的应用。

注意：如果您使用 **APK Signature Scheme v2** 签署您的应用，并对应用进行了进一步更改，则应用的签名将无效。**Walle**并不是为了解决打包速度而产生的一个工具，而是为了适应**V2**签名而研究的因一代打包工具。

瓦力通过在**Apk**中的**APK Signature Block**区块添加自定义的渠道信息来生成渠道包，从而提高了渠道包生成效率，可以作为单机工具来使用，也可以部署在**HTTP**服务器上来实时处理渠道包**Apk**的升级网络请求。

使用

使用**Walle**生成多渠道的速度是非常快的，比如原来的项目打一个包需要两分钟多，每次发布打7个包需要十几分钟。用了**Walle**后，7个包只要两分钟左右就可以完成。

添加依赖&配置渠道

```
android {  
  
    //签名 引用位置  
    signingConfigs {  
        sankuai {  
            storeFile file("keystore/123keyStore.jks")  
            storePassword "123456"  
            keyAlias "key"  
            keyPassword "123456"  
        }  
    }  
  
    //调用签名设置  
    buildTypes {
```

```

        release {
            minifyEnabled false
            proguardFiles
getDefaultProguardFile('proguard-android.txt'),
'proguard-rules.pro'
            signingConfig signingConfigs.sankuai
        }
        debug {
            minifyEnabled false
            proguardFiles
getDefaultProguardFile('proguard-android.txt'),
'proguard-rules.pro'
            signingConfig signingConfigs.sankuai
        }
    }

    //个性化设置
    productFlavors {
        honry {
            manifestPlaceholders = [UMENG_CHANNEL_VALUE:
"honry"]
        }
    }
}

```

在项目的build.gradle文件中添加依赖

```

buildscript {
    dependencies {
        classpath
'com.meituan.android.walle:plugin:1.1.3'
    }
}

```

在当前app的build.gradle文件中添加依赖，以及配置插件

```
apply plugin: 'walle'
```

```
...
```

```
dependencies {
```

```
    ...
```

```
    compile 'com.meituan.android.walle:library:1.1.3'
```

```
}
```

```
walle {
```

```
    // 指定渠道包的输出路径
```

```
    apkOutputFolder = new
```

```
File("${project.buildDir}/outputs/channels");
```

```
    // 定制渠道包的APK的文件名称
```

```
    apkFileNameFormat = '${appName}-${packageName}-
```

```
${channel}-${buildType}-v${versionName}-${versionCode}-  
${buildTime}.apk';
```

```
    // 渠道配置文件
```

```
    channelFile = new
```

```
File("${project.getProjectDir()}/channel")
```

```
}
```

对应的属性：

- apkOutputFolder：指定渠道包的输出路径，默认值为new File("\${project.buildDir}/outputs/apk")。这里指定为build/outputs/apk。
- apkFileNameFormat：定制渠道包的APK的文件名称, 默认值为'\${appName}-\${buildType}-\${channel}.apk'。

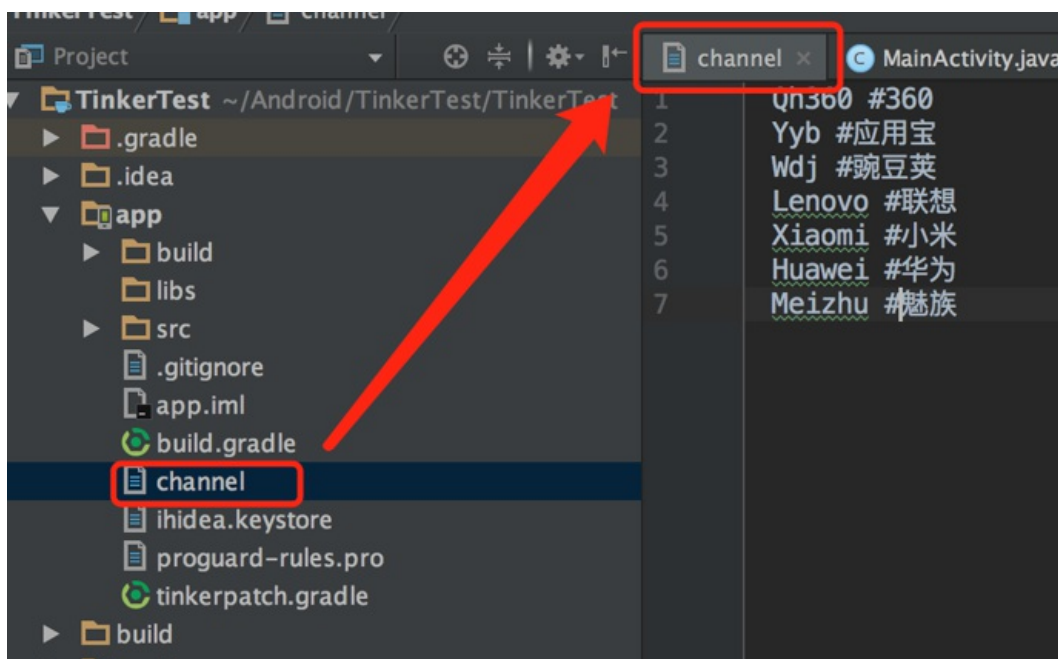
可使用以下变量：

- projectName - 项目名字
- appName - App模块名字

- packageName - applicationId (App包名packageName)
- buildType - buildType (release/debug等)
- channel - channel名称 (对应渠道打包中的渠道名字)
- versionName - versionName (显示用的版本号)
- versionCode - versionCode (内部版本号)
- buildTime - buildTime (编译构建日期时间)
- fileSHA1 - fileSHA1 (最终APK文件的SHA1哈希值)
- flavorName - 编译构建 productFlavors 名
- channelFile: 包含渠道配置信息的文件路径

- channelFile: 包含渠道配置信息的文件路径。

在app目录下创建channel文件，用于配置渠道信息（文件类型：Text）



渠道配置

下面是配置表

```

Qh360 #360
Yyb #应用宝
Wdj #豌豆荚
Lenovo #联想
Xiaomi #小米
Huawei #华为
Meizhu #魅族

```

获取渠道信息

通过以下代码，可以取渠道信息

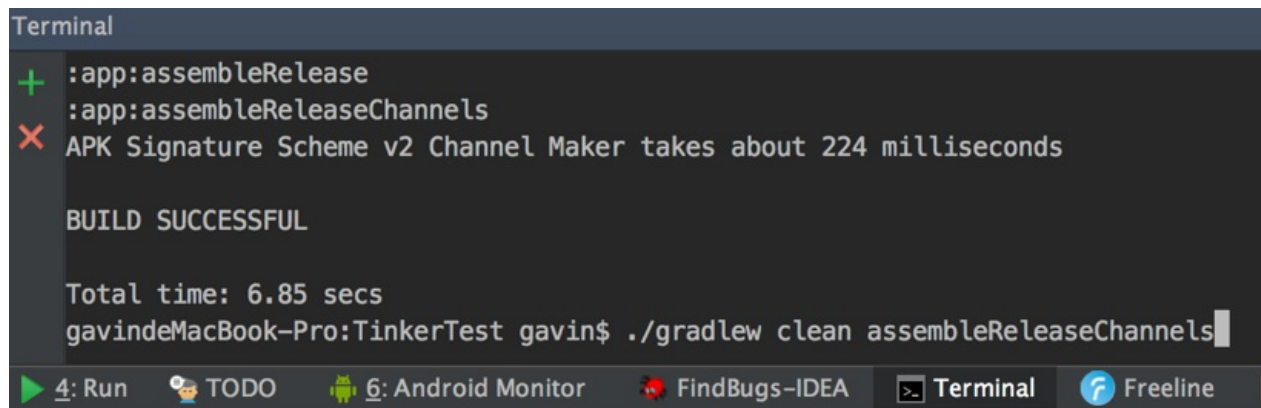
```
String channel =
```

```
WalleChannelReader.getChannel(this.getApplicationContext());
```

生成渠道包

到这里，基本的配置都已完成。接下来可以直接打包了。

在Terminal中输入对应的指令，即可完成打包。看到BUILD SUCCESSFUL后说明已经完成打包了。



```
Terminal
+ :app:assembleRelease
+ :app:assembleReleaseChannels
X APK Signature Scheme v2 Channel Maker takes about 224 milliseconds

BUILD SUCCESSFUL

Total time: 6.85 secs
gavindeMacBook-Pro:TinkerTest gavin$ ./gradlew clean assembleReleaseChannels
```

打包

- 所有渠道

```
./gradlew clean assembleReleaseChannels
```

渠道包的生成目录默认存放在 build/outputs/apk/，也可以通过Walle闭包中的apkOutputFolder参数来指定输出目录

- 指定渠道

生成单个渠道包（huawei渠道）：

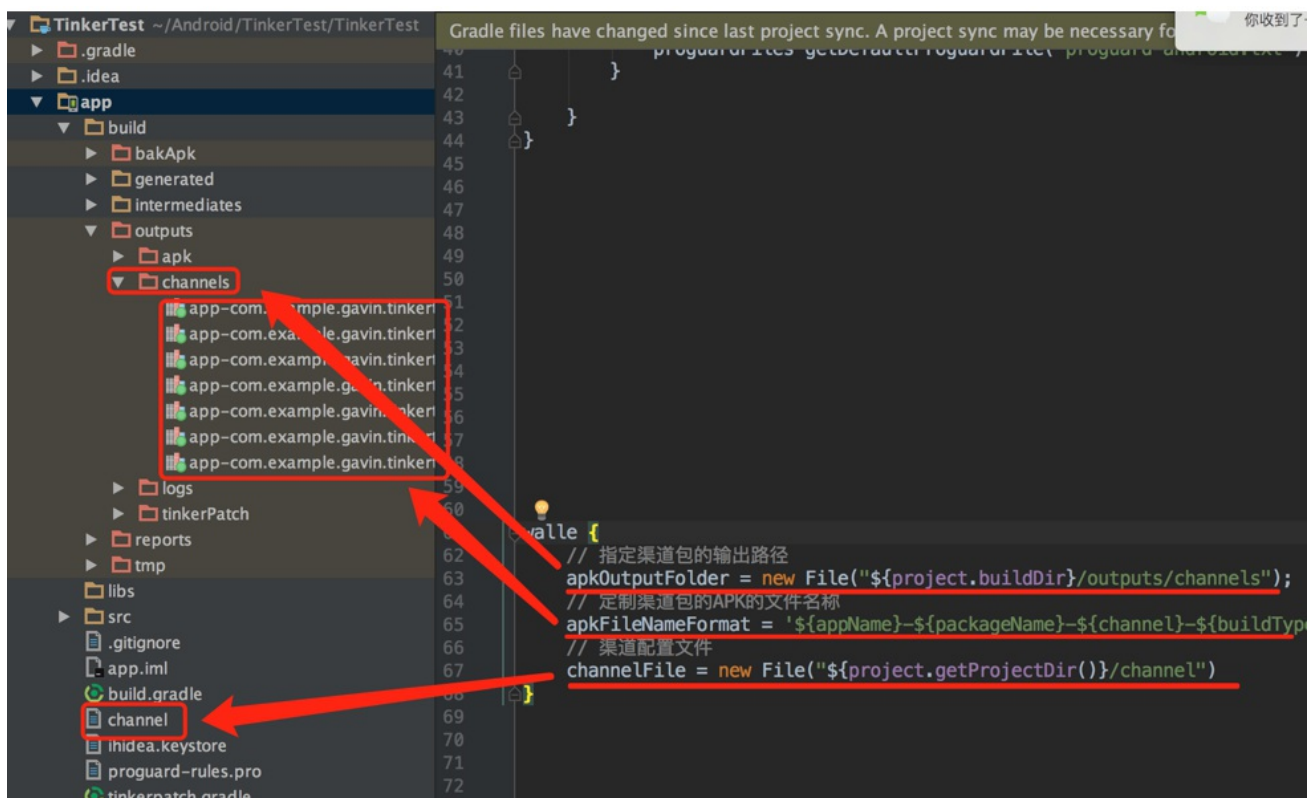
```
./gradlew clean assembleReleaseChannels -PchannelList=huawei
```

生成多个渠道包（huawei、xiaomi渠道）：

```
./gradlew clean assembleReleaseChannels -PchannelList=huawei,xiaomi
```

结果

运行./gradlew clean assembleReleaseChannels后，可以在build/outputs/channels看到对应的渠道包。



—— 以上就是Walle打包的基本流程，想要了解更多可以参考源码

[Walle](#)