**正图Android开发手册**

（试行版）

## 版本说明

|  |  |  |  |
| --- | --- | --- | --- |
| **版本号** | **制定团队** | **更新日期** | **备注** |
| **1.0.0** | **正图技术团队** | **2020/09/10** | **试行版** |
| **1.0.1** | **正图技术团队** | **2020/09/22** | **试行版** |

## 目录

前言 .................................................................... ....... ....... .......................... 2

一、Java 语言规范 ........................................................................................ 3

二、Android 中java 类文件命名与使用........................................................... 3

三、Android 资源文件命名与使用 .................................................................. 3

四、UI 与布局................................................................................................. 7

五、进程、线程与消息通信 .......................................................................... 8

六、应用项目代码初始化规范....................................................................... 8

七、项目架构和项目依赖库一览表................................................................. 9

八、项目代码文件组件化模块化规范…………………………………………….21

九、项目打包上线流程规范………………………………………………………..22

## 前言

本规范的目标是：

 防患未然，提升质量意识，降低故障率和维护成本；

 标准统一，提升协作效率；

 追求卓越的工匠精神，打磨精品代码。

Android 资源文件命名与使用，Android 基本组件，UI 与布局，进程、线程与消息通信，

文件与数据库，Bitmap、Drawable 与动画，安全，其他等九大部分，根据约束力强弱，

规约依次分为强制、推荐、参考三大类：

 【强制】必须遵守，违反本约定或将会引起严重的后果；

 【推荐】尽量遵守，长期遵守有助于系统稳定性和合作效率的提升；

 【参考】充分理解，技术意识的引导，是个人学习、团队沟通、项目合作的方向。

对于规约条目的延伸信息中，“说明”对内容做了适当扩展和解释；“正例”提倡

什么样的编码和实现方式；“反例”说明需要提防的雷区，以及错误案例。

## 一、Java 语言规范

## 二、Android 中java 类文件命名与使用

1. 【强制】activity文件存放，除BaseActivity、BaseMVPActivity外activity不区分kotlin和java文件需要统一存放在项目ui.activity 包下，命名需满足模块名+功能名称+Activity 标识结尾

2. 【强制】 fragment文件存放，除BaseFragment、BaseMVPFragment外activity不区分kotlin和java文件需要统一存放在项目ui. fragment包下，命名需满足模块名+功能名称+Fragment 标识结尾

3. 【强制】 Dialog文件存放，不区分kotlin和java文件需要统一存放在项目ui. custom.dialog包下，命名需满足模块名+功能名称+Dialog标识结尾

4. 【强制】 service文件存放，不区分kotlin和java文件需要统一存放在项目service包下，命名需满足模块名+功能名称+Service标识结尾

5. 【强制】 Popwindow文件存放，不区分kotlin和java文件需要统一存放在项目ui. custom.popwindow包下，命名需满足模块名+功能名称+Popwindow标识结尾

## 三、Android 资源文件命名与使用

1. 【推荐】资源文件命名使用 {组件名称}\_模块名称。

2. 【强制】layout 文件的命名方式,文件必须全部小写并且必须字母开头。

Activity 的 layout 以 activity\_ module 开头

Fragment 的 layout 以 fragment\_module 开头

Dialog 的 layout 以 dialog\_module 开头

include 的 layout 以 include\_module 开头

ListView 的行 layout 以 item\_module 开头

RecyclerView 的 item layout 以 item\_module 开头

GridView 的行 layout 以item\_module 开头

3. 【推荐】 drawable 资源名称以小写单词+下划线的方式命名，根据分辨率不同存放

在不同的 drawable 目录下，建议只使用一套,例如 drawable-xhdpi。采用规则如下:

模块名\_业务功能描述\_控件描述\_控件状态限定词

如：module\_login\_btn\_pressed,module\_tabs\_icon\_home\_normal

4. 【推荐】anim 资源名称以小写单词+下划线的方式命名，采用以下规则：

模块名\_逻辑名称\_[方向|序号]

tween 动 画 资 源 ： 尽 可 能 以 通 用 的 动 画 名 称 命 名 ， 如 module\_fade\_in ,

module\_fade\_out , module\_push\_down\_in (动画+方向)；

frame 动画资源：尽可能以模 块+功能命名+序号。如：module\_loading\_grey\_001

5. 【推荐】 color 资源使用#AARRGGBB 格式，写入 module\_colors.xml 文件中，命

名格式采用以下规则：

模块名\_逻辑名称\_颜色

如：

<color name="module\_btn\_bg\_color">#33b5e5e5</color>

6. 【推荐】dimen 资源以小写单词+下划线方式命名，写入 module\_dimens.xml 文件中，

采用以下规则：

模块名\_描述信息

如：

<dimen name="module\_horizontal\_line\_height">1dp</dimen>

7. 【推荐】style 资源采用小写单词+下划线方式命名，写入 module\_styles.xml 文件中，

采用以下规则：

父 style 名称.当前 style 名称

如：

<style name="ParentTheme.ThisActivityTheme">

…

</style>

8. 【推荐】string资源文件或者文本用到字符需要全部写入module\_strings.xml文件中，

字符串以小写单词+下划线的方式命名，采用以下规则：

模块名\_逻辑名称

如：moudule\_login\_tips,module\_homepage\_notice\_desc

9. 【推荐】Id 资源原则上以驼峰法命名，View 组件的资源 id 需要以 View 的缩写作为

前缀。常用缩写表如下：

控件 缩写

LinearLayout ll

RelativeLayout rl

ConstraintLayout cl

ListView lv

ScollView sv

TextView tv

Button btn

ImageView iv

CheckBox cb

RadioButton rb

EditText et

RecyclerView rv

二、Android 资源文件命名与使用

其它控件的缩写推荐使用小写字母并用下划线进行分割，例如：

ProgressBar 对应的缩写为 progress\_bar

DatePicker 对应的缩写为 date\_picker

## 四、UI 与布局

1. 【强制】布局中不得不使用 ViewGroup 多重嵌套时，不要使用 LinearLayout 嵌套，改用 RelativeLayout，可以有效降低嵌套数。

说明：

Android 应用页面上任何一个 View 都需要经过 measure、layout、draw 三个步骤才能被正确的渲染。从 xml layout 的顶部节点开始进行 measure，每个子节点都需要向自己的父节点提供自己的尺寸来决定展示的位置，在此过程中可能还会重新measure（由此可能导致 measure 的时间消耗为原来的 2-3 倍）。节点所处位置越深，套嵌带来的 measure 越多，计算就会越费时。这就是为什么扁平的 View 结构会性能更好。

同时，页面拥上的 View 越多，measure、layout、draw 所花费的时间就越久。要缩短这个时间，关键是保持 View 的树形结构尽量扁平，而且要移除所有不需要渲染的View。理想情况下，总共的 measure，layout，draw 时间应该被很好的控制在 16ms

以内，以保证滑动屏幕时 UI 的流畅。

要找到那些多余的 View（增加渲染延迟的 view），可以用 Android Studio Monitor里的 Hierarachy Viewer 工具，可视化的查看所有的 view。

正例：

<?xml version="1.0" encoding="utf-8"?>

<android.support.constraint.ConstraintLayout>

<RelativeLayout>

<TextView/>

...

<ImageView/>

</RelativeLayout>

</android.support.constraint.ConstraintLayout>

反例：

<LinearLayout>

<LinearLayout>

<RelativeLayout>

<TextView/>

...

<ImageView/>

</RelativeLayout>

</LinearLayout>

</LinearLayout>

多重嵌套导致 measure 以及 layout 等步骤耗时过多。

2. 【推荐】源文件统一采用 UTF-8 的形式进行编码。

3. 【强制】禁止在非 ui 线程进行 view 相关操作。

4. 【推荐】灵活使用布局，推荐 Merge、ViewStub 来优化布局，尽可能多的减少 UI

布局层级，推荐使用 FrameLayout，LinearLayout、RelativeLayout 次之。

## 五、进程、线程与消息通信

1. 【强制】不要通过 Intent 在 Android 基础组件之间传递大数据（binder transaction

缓存为 1MB），可能导致 OOM。

2. 【强制】在 Application 的业务初始化代码加入进程判断，确保只在自己需要的进程

初始化。特别是后台进程减少不必要的业务初始化。咱们的项目中的初始化放在application的otherinit（）方法中

## 六、应用项目代码初始化规范

三方库moudle的代码初始化放在 MApplicationInfo 类中，这个是所有三方框架初始化的管理类，当我们引入三方库的时候需要往这个里面增加初始化代码Application，为了解决应用启动时间白屏化问题，我们采用了二级初始化策略，初始化代码分为两部分，第一部分为所有activity界面不可见前的应用必要初始化，这一步分的优先级比较高；第二部分是优先略低的初始化，这个是在应用欢迎页等待中同步执行的，可以存放剩余的所有的三方库初始化

**第一部分的初始化代码片段如下**

init(Application sApplication, Context context) {

//这里存放程序运行必备的初始化，比如日志、CommonUtils等需要在activity首屏需要的使用的框架的初始化，为了避免应用出现白屏，禁止存放耗时初始化操作

}

代码片段需要

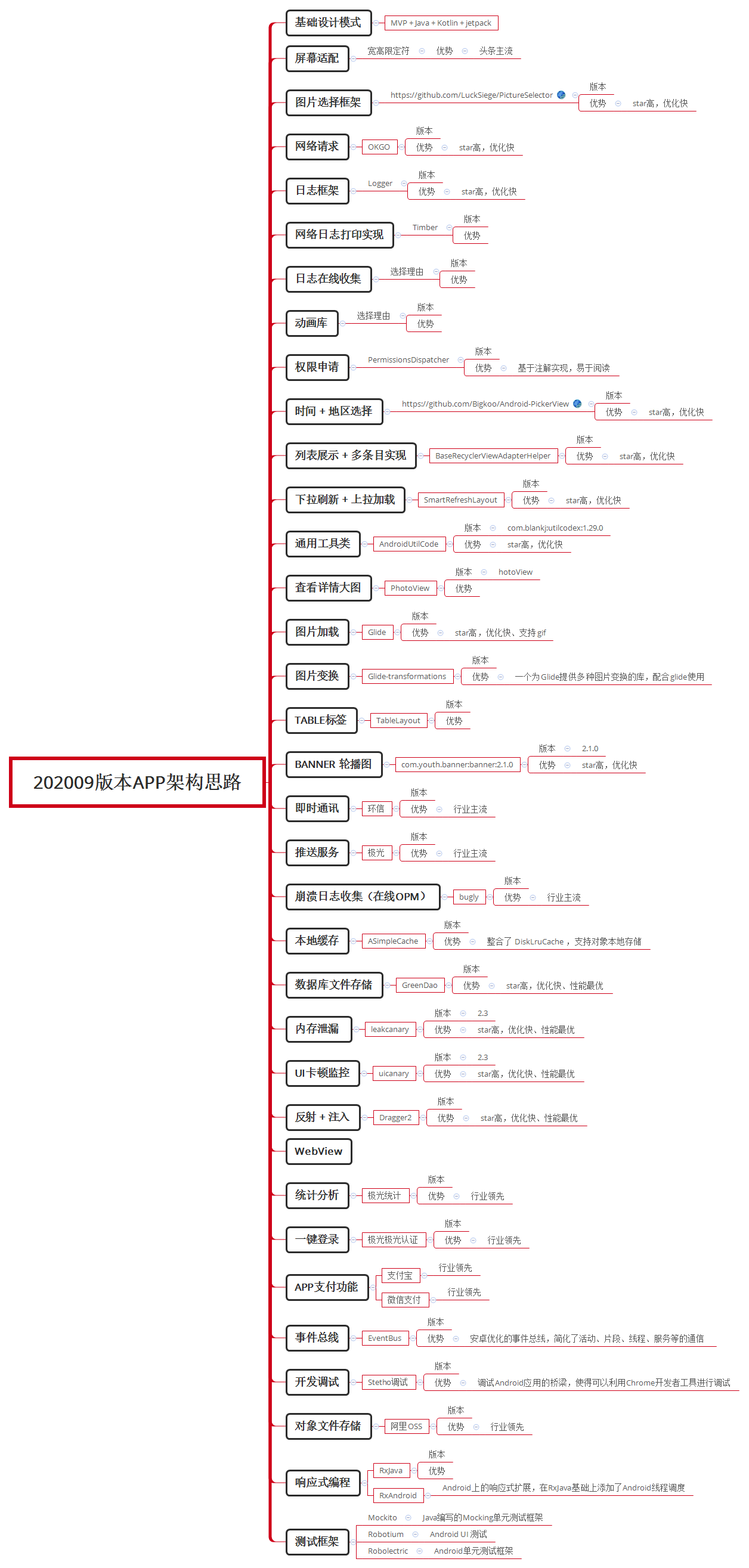
第二部分的初始化片段如下

**public void** otherInit() {

//这里存放其他三方控件的初始化  
 initEasenob();  
 initOkGo();  
 initSwipeBack();  
 initPictureCompress();  
 initImageLoader();  
 initJPush();  
 initJpushLoginSdk();  
 initJAnalytics();  
 initDreenDao();  
 SDKINIT();  
 initBugly();  
 initLeakCanary();  
 aliOSSInit();  
 initCrash();  
 }

## 七、项目架构和项目依赖库一览表

注意：结合目前主流的声音，我们选用mvp+mvc混合开发的形式，对应业务单一的界面（如仅涉及单一列表展示，不需要网络交互的界面，将采用mvp，activity将继承自BaseActivity类完成开发，fragment将继承自BaseFragment进行开发），其他场景将优先采用mvp架构进行ui与业务逻辑的分离达到更好的代码易读性



## 1. 屏幕适配

| **名称** | **描述** |
| --- | --- |
| [smallestWidth 限定符适配](https://www.jianshu.com/p/1302ad5a4b04) | 与设计稿最接近，必要时可以Android 、ISO通用一套设计图 |

## 2.图片选择框架

| **名称** | **描述** |
| --- | --- |
| [PictureSelector](https://github.com/LuckSiege/PictureSelector) | 支持微信样式、可以使用NDK进行压缩、支持gif、视频选择，效果稳定，内存扰动小 |

## 3.网络请求

| **名称** | **描述** |
| --- | --- |
| [**okhttp-**OkGo](https://github.com/jeasonlzy/okhttp-OkGo) | Java实现基于LRU的磁盘缓存 |

## 4.日志框架

| **名称** | **描述** |
| --- | --- |
| [**logger**](https://github.com/orhanobut/logger) | 老牌本地日志框架，针对Android做了优化 |

## 5.日志在线收集

| **名称** | **描述** |
| --- | --- |
| [**timber**](https://github.com/JakeWharton/timber) | 在线日志打印 |

## 6.动画库

| **名称** | **描述** |
| --- | --- |
|  |  |

## 7.权限申请

| **名称** | **描述** |
| --- | --- |
| [**PermissionsDispatcher**](https://github.com/permissions-dispatcher/PermissionsDispatcher) | 基于注解实现，易于阅读 |

## 8.时间 + 地区选择

| **名称** | **描述** |
| --- | --- |
| [**Android-PickerView**](https://github.com/Bigkoo/Android-PickerView) | 支持类型多、样式适配完整 |

## 9.列表展示 + 多条目实现

| **名称** | **描述** |
| --- | --- |
| [**BaseRecyclerViewAdapterHelper**](https://github.com/CymChad/BaseRecyclerViewAdapterHelper) | 代码量小，易扩展，易使用 |

## 10.下拉刷新 + 上拉加载

| **名称** | **描述** |
| --- | --- |
| [**SmartRefreshLayout**](https://github.com/scwang90/SmartRefreshLayout) | 样式多样，滑动冲突处理优秀 |

## 11.通用工具类

| **名称** | **描述** |
| --- | --- |
| [**AndroidUtilCode**](https://github.com/Blankj/AndroidUtilCode) | 内置android交互底层的使用工具类，流行 |

## 12.查看详情大图

| **名称** | **描述** |
| --- | --- |
| [**PhotoView**](https://github.com/chrisbanes/PhotoView) | 性能优秀 |

## 13.图片加载 + 变换

| **名称** | **描述** |
| --- | --- |
| [**Gilde**](https://muyangmin.github.io/glide-docs-cn/) | 支持gif图，官方支持 |
| [**glide-transformations**](https://github.com/wasabeef/glide-transformations) | 和Gilde一起使用 |

## 14. banner 轮播图

| **名称** | **描述** |
| --- | --- |
| [**banner**](https://github.com/youth5201314/banner) | 样式多样，支持底部文字 |

## 15.即时通讯

| **名称** | **描述** |
| --- | --- |
| 环信 | 行业标杆 |

## 16.推送服务

| **名称** | **描述** |
| --- | --- |
| 极光 | 行业标杆 |

## 17.崩溃日志收集（在线OPM）

| **名称** | **描述** |
| --- | --- |
| [**Bugly**](https://bugly.qq.com/v2/workbench/apps) | 行业标杆 |

## 18.本地缓存

| **名称** | **描述** |
| --- | --- |
| [**ASimple**Cache](https://github.com/yangfuhai/ASimpleCache) | 整合了 DiskLruCache ，支持对象本地存储 |

## 19.数据库文件存储

| **名称** | **描述** |
| --- | --- |
| [**greenDAO**](https://github.com/greenrobot/greenDAO) | Android 已针对设备优化 |
| [**reenDaoUpgradeHelper**](https://github.com/yuweiguocn/GreenDaoUpgradeHelper) | 升级数据库使用 |

## 20.内存泄漏 + UI卡顿监控

| **名称** | **描述** |
| --- | --- |
| [**leakcanary**](https://github.com/square/leakcanary) | 内存泄漏检测工具 |
| Ui[**leakcanary**](https://github.com/square/leakcanary) |  |

## 21. 反射 + 注入

| **名称** | **描述** |
| --- | --- |
| Dragger2 | 功能强大 |

## 22. 统计分析

| **名称** | **描述** |
| --- | --- |
| 极光统计 | 行业标杆 |

## 23.一键登录

| **名称** | **描述** |
| --- | --- |
| 极光认证 | 行业标杆 |

## 24. APP支付功能

| **名称** | **描述** |
| --- | --- |
| 支付宝支付 | 行业标杆 |
| 微信支付 | 行业标杆 |

## 25.事件总线

| **名称** | **描述** |
| --- | --- |
| [**EventBus**](https://github.com/greenrobot/EventBus) | 安卓优化的事件总线，简化了活动、片段、线程、服务等的通信 |

## 26.开发调试

| **名称** | **描述** |
| --- | --- |
| [**Stetho**](https://github.com/facebook/stetho) | 调试Android应用的桥梁，使得可以利用Chrome开发者工具进行调试 |

## 27.对象文件存储

| **名称** | **描述** |
| --- | --- |
| OSS | 行业标杆 |

## 28. 响应式编程

| **名称** | **描述** |
| --- | --- |
| [RxJava](https://github.com/ReactiveX/RxJava) | JVM上的响应式扩展 |
| [RxAndroid](https://github.com/ReactiveX/RxAndroid) | Android上的响应式扩展，在RxJava基础上添加了Android线程调度 |

## 29. 测试框架

| **名称** | **描述** |
| --- | --- |
| [Mockito](https://github.com/mockito/mockito) | Java编写的Mocking单元测试框架 |
| [Robotium](https://github.com/RobotiumTech/robotium) | Android UI 测试 |
| [Robolectric](https://github.com/xtremelabs/robolectric) | Android单元测试框架 |

## 30.地图

| **名称** | **描述** |
| --- | --- |
| 高德 | 行业标杆 |

## 九、项目打包流程规范

正图项目打包分为三种场景分别是

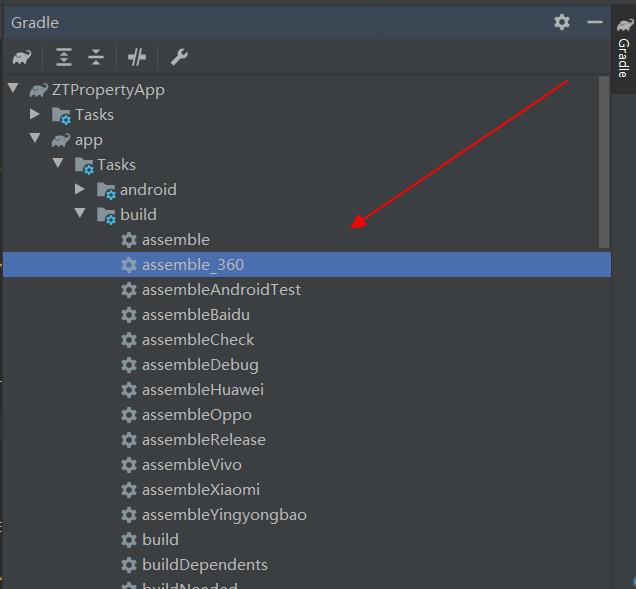
1).研发人员调试

2).测试人员测试版本

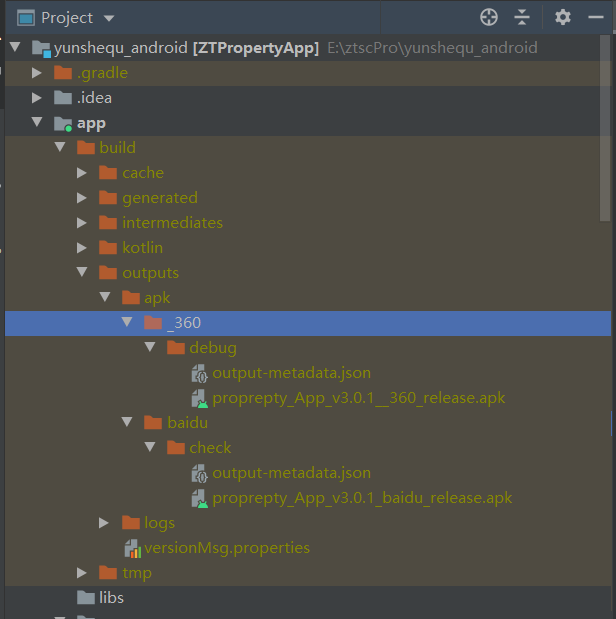
3).正式上线版本

### 1.研发人员调试版本

使用Android的run按钮直接运行或者使用gradle脚本



安装包打完之后使用，Androidstudio左侧可见



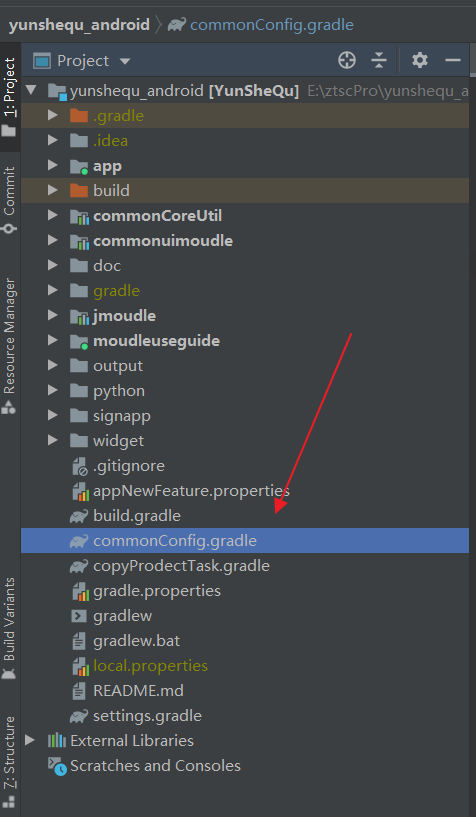
Cmd启动adb 使用命名

adb install c://xxx/xxx/x.apk进行安装

详细命令请参考博客 [Android adb shell 常用命令](https://blog.csdn.net/bencheng06/article/details/87193927)

### 2.打包给测试人员

1） 找到项目的配置文件 commonConfig.gradle 修改其配置项



主要配置以下的信息

//脚本需要上传的build\_type安装包类别，脚本将过滤该类别的安装包，进行上传处理  
app\_build\_type = [  
 build\_type\_check : "check", //测试环境包  
 build\_type\_debug : "debug", //测试开发环境包  
 build\_type\_release: "release" //生产环境包  
]  
  
  
//蒲公英账号配置，程雷账号  
pugongyingAccount = [  
 pugongying\_api\_key\_chenglei: "c8364150e5514d79cf7563013e8b231d", //程雷账号  
 pugongying\_uKey\_chenglei : "29ee634c5e80425b93c30667c168a11b" //程雷账号  
]  
  
  
ztscAccount = [  
 up2ZtscApkUrl: "http://119.3.162.131:7080/pub-service/Service?service=file&function=upload",//apk上传路径  
 up2ZtscMsgUrl: "http://www.correctmap.com.cn/pub-service/Service?service=dataManager&function=newAppVerson" //apk版本记录写入路径  
]  
  
  
appPort = "publicApp" //正图云社区 = publicApp  
appPortName = "正图云社区" //正图云社区 = publicApp

上面的文件的引入是在 appPush 这个数组里面

//控制上传市场  
appPush = [push2ZTSC : "false", //上传公司服务器 【true : 上传 false ： 不上传】  
 push2Pugongying : "true", //上 传 蒲 公 英 【true : 上传 false ： 不上传】  
 appUpType : app\_build\_type['build\_type\_check'], //配置上apk的build\_type  
 pugongying\_api\_key: pugongyingAccount['pugongying\_api\_key\_chenglei'], //api鉴权的key  
 pugongying\_uKey : pugongyingAccount['pugongying\_uKey\_chenglei'], //蒲公英userKey  
 up2ZtscApkUrl : ztscAccount['up2ZtscApkUrl'], //apk上传路径  
 up2ZtscMsgUrl : ztscAccount['up2ZtscMsgUrl'], //apk版本记录写入路径  
 appPort : appPort, //用于区别是哪个端的APP  
 appPortName : appPortName //给人阅读的端的名称  
]

2） 上传脚本参数配置完成了，就可以进行下一步了，配置打包apk的信息，文件同样是commonConfig.gradle

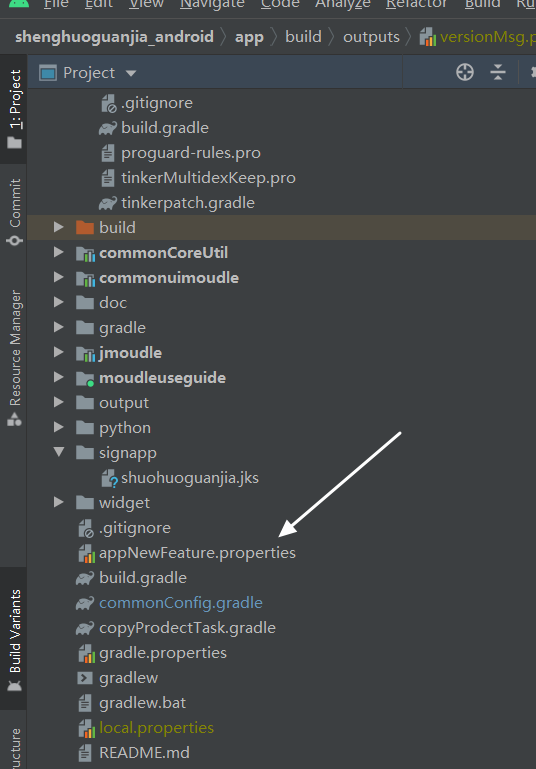
android = [applicationId : 'com.ztsc.house',  
 compileSdkVersion: 29,  
 buildToolsVersion: '29.0.0',  
 minSdkVersion : 21,  
 targetSdkVersion : 29,  
 versionCode : 4301,  
 versionName : '3.0.1', //编码分为9位 ，具体与 versionName 同步，前三位 第一个大版本 中间三位 次大版本，最后三位 小版本号  
 versionSupporPad : '1', //各个项目工程版本的全局配置 ,版本更新前请确认更新,此版本是否支持平板 0:不支持，1:支持  
 updateNecess : 'may', //may : 提示用户更新，用户可以选择忽略 must:强制用户更新 ignore ：直接忽略本次更新  
 greendaoDbVersion: 15  
]

updateNecess：主要配置的是APP内部更新的UI界面

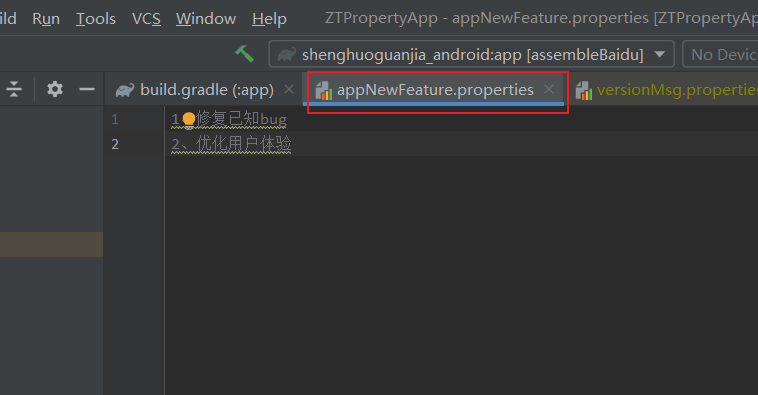
versionSupporPad：配置的是当前需要打包的APK安装包是不是可以兼容pad，0，不兼容、1，兼容。默认为1，在当前框架里面是可以兼容pad的

每次打包之前需要修改 versionCode 、 versionName 版本号只可以增不可以降低，否则会导致内部更新出现异常，请慎重操作

3) 每次打包请编写对应的更新文案



文案中的格式将会保留，跟随脚本同步到对应蒲公英、正图内部更新服务器（当APP内部更新的时候展示在更新对话框中）



4） 打包前请检查对应APP下生成临时文件的task配置正确（位置APP下的build.gralde文件）

//配置任务自动化打包  
 task appUpdateMsg(group: "packageTool", description: "配置并输出应用包日志信息") **{** //读取文件，并装进List。  
 def updateMsg = ''  
 new File("$**{**getRootDir().absolutePath**}**\\appNewFeature.properties").withReader('UTF-8') **{** reader **->** updateMsg = reader.text  
 **}** def appVersionName = android.versionName  
 def appVersionCode = android.versionCode  
 def appSupportPad = android.versionSupporPad  
 def appUpType = appPush.appUpType  
 def pugongying\_api\_key = appPush.pugongying\_api\_key  
 def pugongying\_uKey = appPush.pugongying\_uKey  
 def up2ZtscApkUrl = appPush.up2ZtscApkUrl  
 def up2ZtscMsgUrl = appPush.up2ZtscMsgUrl  
 def appPort = appPush.appPort  
 def appPortName = appPush.appPortName  
 def updateNecess = android.updateNecess  
 def push2ZTSC = appPush.push2ZTSC  
 def push2Pugongying = appPush.push2Pugongying  
 StringBuffer msg = new StringBuffer()  
 msg.append('{')  
 .append(appendStr("newVersionName", appVersionName, ","))  
 .append(appendStr("newVersionCode", appVersionCode, ","))  
 .append(appendStr("isPad", appSupportPad, ","))  
 .append(appendStr("appPort", appPort, ","))  
 .append(appendStr("appPortName", appPortName, ","))  
 .append(appendStr("updateNecess", updateNecess, ","))  
 .append(appendStr("updateMsg", updateMsg, ","))  
 .append(appendStr("push2ZTSC", push2ZTSC, ","))  
 .append(appendStr("push2Pugongying", push2Pugongying, ","))  
 .append(appendStr("appUpType", appUpType, ","))  
 .append(appendStr("pugongying\_api\_key", pugongying\_api\_key, ","))  
 .append(appendStr("pugongying\_uKey", pugongying\_uKey, ","))  
 .append(appendStr("up2ZtscApkUrl", up2ZtscApkUrl, ","))  
 .append(appendStr("up2ZtscMsgUrl", up2ZtscMsgUrl, ""))  
 .append('}')  
  
 def file = new File("$**{**getRootDir().absolutePath**}**/app/build/outputs/versionMsg.properties")  
 if (!file.getParentFile().exists()) {  
 file.getParentFile().mkdirs()  
 }  
 file.withWriter('UTF-8') **{** writer **->** writer.write(msg.toString())  
 **}  
 }  
  
}**

这个task的执行会在gradle的build，打包等只要是触发同步、构建等任何gradle执行阶段，所以只要确认这段代码有，并且是在app下的build.gradle 下与Android标签同级目录即可

这个task的作用主要是生成和apk配置相关的配置json，生成目录在${rootProject}/app/build/outputs/versionMsg.properties

5） 确认APP输出规则配置正确

//自定义打包输出apk的路径和名称  
applicationVariants.all **{** variant **->** //批量修改Apk名字  
 variant.outputs.all **{** output **->** def outputFile = output.outputFile  
  
 if (outputFile != null && outputFile.name.endsWith('.apk')) {  
 //获取签名的名字 variant.signingConfig.name  
 //要被替换的源字符串  
  
 //输出apk名称为wooyun\_v1.0\_wandoujia\_release.apk  
 println "当前文件的名字$**{**outputFile.name**}**"  
 //替换的字符串  
 def replaceFile = "$**{**appPush.appPort**}**\_App\_v$**{**variant.versionName**}**\_$**{**variant.flavorName**}**\_$**{**variant.buildType.buildType.name**}**.apk"  
  
 //输出对应apk信息  
 // outApkLog(outputFile.getParentFile().path, replaceFile)  
  
  
 outputFileName = replaceFile  
  
 }  
 **}  
  
}**

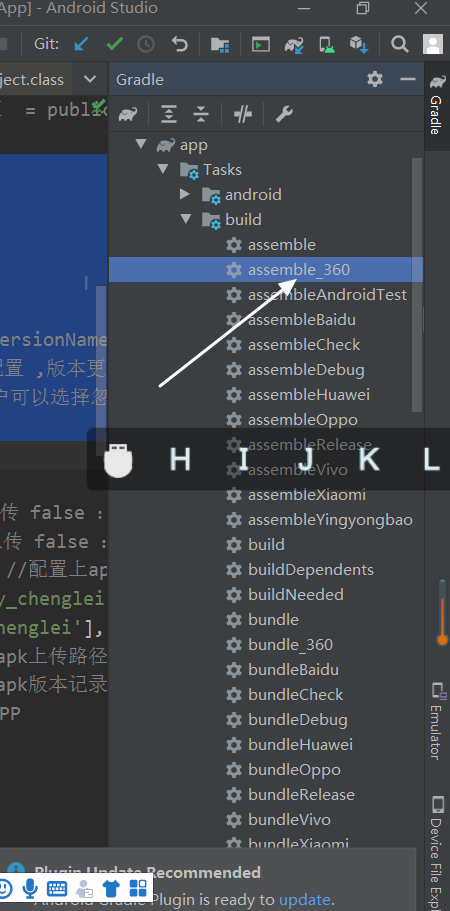
这段代码在app的build.gradle 下的android节点下，不需要做修改请直接拷贝

生成apk的文件格式如下



6）配置完成进入脚本打包阶段

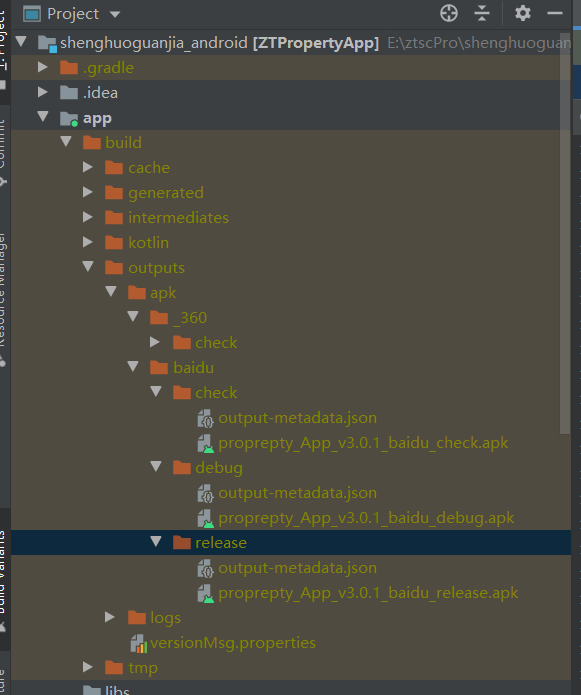
如果你已经看到这里恭喜你，距离打包成功只用最后的一步了，是不是很激动来



打包之前先进行clean操作，切记，防止缓存对apk影响

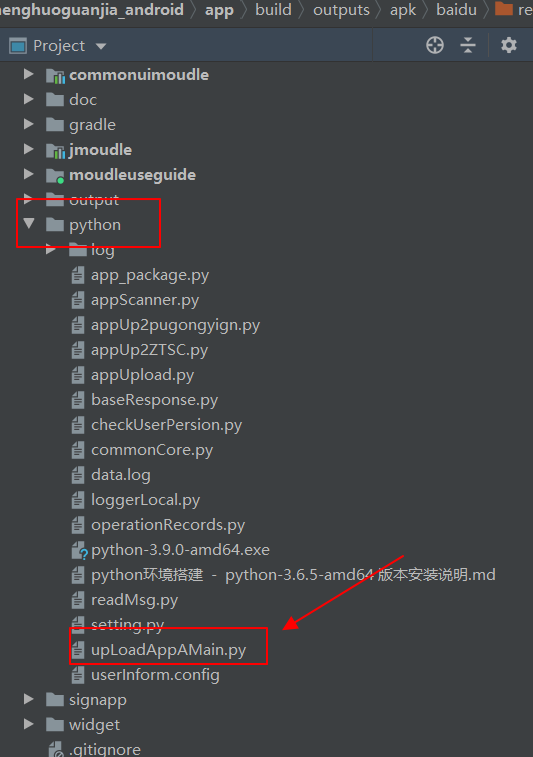
点击右侧的Gradle –》app –》build –》assemble\_360 进行打包，当然如果喜欢你还可以使用assembleBaidu等

等待漫长的编译之后，

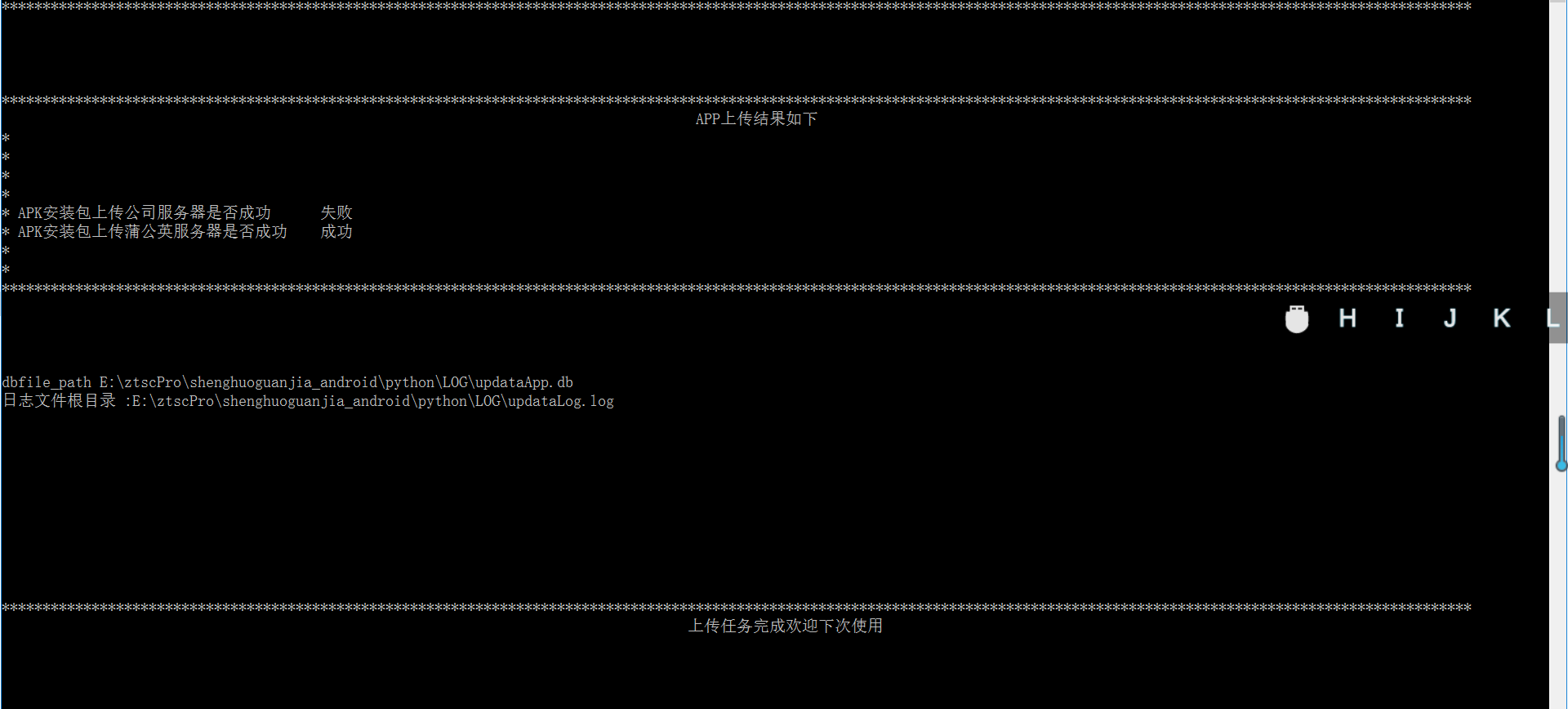


对应的apk生产成了

7） 运行脚本上传蒲公英



Python下的upLoadAppAMain.py为上传脚本的真正入口，在文件夹中双击运行即可，如果有不明的请阅读环境搭建说明

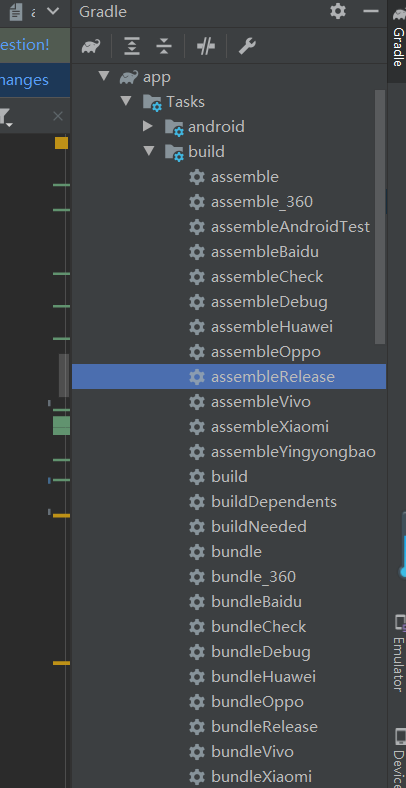


脚本还是很人性化的，会有对一个日志输出和提示完成程度

### 3.多渠道打包，项目上线

参考上面 打包给测试人员

运行



点击右侧的Gradle –》app –》build –》assembleRelease 进行打包,剩下的就是上传市场和蒲公英了，请在市场审核通过后上传内部更新