<https://drakeet.me/gradle-change-android-resvalue-buildconfig-and-manifest>

# 使用 gradle 在编译时动态设置 Android resValue / BuildConfig / Manifest

首先说明下，本文严重参考并得益于我的朋友 [@linroid](http://linroid.com/" \t "https://drakeet.me/_blank) 的文章：《[xip.io + gradle 在调试时动态设置服务端地址](http://linroid.com/2015/07/24/dynamic-endpoint-when-debug-by-xip-io-and-gradle/" \t "https://drakeet.me/_blank)》，我觉得特别好，他写得好，我用得爽，特别值得再记录下来。于是我将在这里写得更普适、完整一点，感谢与分享！

当我在使用 telegram 的时候，我意识到一些问题，并且自行修改了，改了什么不是关键，有兴趣的朋友可以自行去看看[我的版本](https://github.com/drakeet/Tel4Drakeet" \t "https://drakeet.me/_blank)。在编译的时候，我需要提供 2 个版本，一，我自己的版本，二，分享给大家顺便使用的版本。其实二者唯一的差别就是 App 名称不一样，我自己的版本我想叫「Tel4Drakeet」，但分享给别人，叫这个名字可能不太合适。

因此，我如果每次手动修改再进行编译，显然很麻烦。于是想到借助 gradle，想到林的那篇文章，原本我只知道 buildConfigField 和类似友盟多渠道打包那样，修改清单文件内容 manifestPlaceholders，但这两种都不是我现在想要的，我要的是能够修改 res value 的方式，比如修改 strings.xml 文件中的 AppName 的值。

在林的文章中正好找到这点，貌似在整个互联网上并不容易找到。所以我要总结记录下来：

#### ****使用 gradle 在编译时动态设置 Android resValue****

在你的 gradle 内容 buildTypes 或者 productFlavors 下面，如 release 体内写上类似：

|  |  |
| --- | --- |
| 1 | resValue "string", "AppName", "Tel4Drakeet" |

意为把名为 AppName 的 string value 值改为 Tel4Drakeet，完整地和多个 buildType 结合可以参考我的 [Tel4Drakeet](https://github.com/drakeet/Tel4Drakeet/blob/master/tele4drakeet/build.gradle" \t "https://drakeet.me/_blank) 的 gradle 文件。

#### ****使用 gradle 在编译时动态设置 Android BuildConfig****

在同上的地方写上：

|  |  |
| --- | --- |
| 1 | buildConfigField "String", "ENDPOINT", "\"[http://example.com](http://example.com/)\"" |

gradle sync 一下后，BuildConfig.ENDPOINT 就会被赋值为 http://example.com 就可以供 Java 代码调用了。

#### ****使用 gradle 在编译时动态设置 Android Manifest****

我们在使用友盟多渠道打包的时候，有这么种方式可以在编译的时候动态修改清单文件中的内容，从而实现不同渠道不同标识值。

你需要在 AndroidManifest.xml 中，对友盟的渠道进行如下配置：

|  |  |
| --- | --- |
| 1  2  3 | <meta-data      android:name="UMENG\_CHANNEL"      android:value="${UMENG\_CHANNEL\_VALUE}"/> |

然后在你的 product flavor 中写上：

|  |  |
| --- | --- |
| 1 | manifestPlaceholders = [UMENG\_CHANNEL\_VALUE: "GooglePlay"] |

完整 gradle 文件可以参考我的这个项目：[妹纸&gank.io](https://github.com/drakeet/Meizhi/blob/master/app/build.gradle" \t "https://drakeet.me/_blank). 这样清单文件中原本预设的 ${UMENG\_CHANNEL\_VALUE} 这个值就会被替换为”GooglePlay”。如果要和 Java 配合调用，就是如上所示，结合访问 meta-data 即可，这里不展开了。

最后来说，其实也不算非常完整，有许多步骤和灵活运用我可能没说，一方面因为我并没有系统完整学过 groovy（gradle 文件的编写语言），另一方面，其实做到启蒙就不错了，聪明的人可以通过通过本文的关键词，使用 Google 等等渠道自己获取得到更多的内容。

资源替换方面，也可在src下建立文件夹“xxx”（xxx为渠道名称），在它下面放置要替换的res/values 等，这样打包时，资源就可根据渠道定制不同样式

# xip.io + gradle 在调试时动态设置服务端地址

日常开发中，如果服务端在本地，通常可通过改hosts、写死IP、动态域名等方式来设置服务端地址，但总觉很麻烦，不灵活；比如更换网络导致IP变化，就得重新设置。  
今天突然想到，利用xip.io 和 gradle来自动设置服务端地址

[xip.io](http://xip.io/" \t "http://linroid.com/2015/07/24/dynamic-endpoint-when-debug-by-xip-io-and-gradle/_blank) 是一个直接使用域名来指定IP的域名服务，无需手动设置DNS，同时也不需要任何注册。这解决了使用IP无法使用多个Virtual Host而使用域名又得很麻烦改地DNS的问题。

xip.io支持{custom\_prefix}.{host\_ip}.xip.io的域名格式，解析出来的ip就是{host\_id}。如:

exmple.com.127.0.0.1.xip.ip 会解析为127.0.0.1

下面进行grale配置:

|  |
| --- |
| buildTypes {  debug {  //...  def hostIp = InetAddress.getLocalHost().getHostAddress()  buildConfigField "String", "ENDPOINT", "\"http://example.com.${hostIp}.xip.io/api\""  }  release {  //...  buildConfigField "String", "ENDPOINT", "\"http://example.com/api\""  }  } |

在debug下，gradle会获取当前电脑的IP，然后写入到BuildConfig类中的ENDPOINT属性。gradle sync一下后，BuildConfig.ENDPOINT就被赋值为http://example.com.${hostIp}.xip.io/api

当然，还可以放到xml文件中:)

|  |
| --- |
| resValue "string", "host\_url", "http://example.com.${hostIp}.xip.io"; |

最后记得在apache/nginx 等配置Virtual Host时使用宽域名,如Nginx中:

|  |
| --- |
| server\_name example.com.\* |

# 编译时替换资源 - Android重叠包与资源合并一见

# **前言**

在[Android逆向分析(2) APK的打包与安装](http://blog.zhaiyifan.cn/2016/02/13/android-reverse-2/)一文中对资源编译过程的介绍中，笔者提到了overlay（重叠包）这个概念，一位每天都被自己帅醒的好友看了那篇东西后，来问我这个重叠包究竟是个什么东西，笔者想了想，确实这个概念有很多同学们都不甚了解，搜索了一下网上了介绍，也几乎没有看到任何对这个的讲解，只有[老罗的博客](http://blog.csdn.net/luoshengyang/article/details/8744683" \t "http://blog.zhaiyifan.cn/2016/02/18/android-resource-overlay/_blank)提到过

–mOverlay：表示当前正在编译的资源的重叠包。重叠包是什么概念呢？假设我们正在编译的是Package-1，这时候我们可以设置另外一个Package-2，用来告诉aapt，如果Package-2定义有和Package-1一样的资源，那么就用定义在Package-2的资源来替换掉定义在Package-1的资源。通过这种Overlay机制，我们就可以对资源进行定制，而又不失一般性。

那我们应该怎么怎么去使用重叠包呢？它又能用在什么地方，带来什么便利呢？

本文从aapt overlay切入，介绍其用法和在gradle的实践，并接着介绍了gradle plugin提供的资源合并机制和实践。

测试源码: [ResourceOverlayDemo](https://github.com/markzhai/ResourceOverlayDemo" \t "http://blog.zhaiyifan.cn/2016/02/18/android-resource-overlay/_blank)。

# **aapt overlay**

我们看看aapt的命令help里是怎么描述的，省略版:

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 | Usage:  aapt l[ist] [-v] [-a] file.{zip,jar,apk}  List contents of Zip-compatible archive.   aapt d[ump] [--values] [--include-meta-data] WHAT file.{apk} [asset [asset ...]]  ...   aapt p[ackage] [-d][-f][-m][-u][-v][-x][-z][-M AndroidManifest.xml] \  ...  [--utf16] [--auto-add-overlay] \  ...  [-S resource-sources [-S resource-sources ...]] \  [-F apk-file] [-J R-file-dir] \  ...   Package the android resources. It will read assets and resources that are  supplied with the -M -A -S or raw-files-dir arguments. The -J -P -F and -R  options control which files are output.  ...   Modifiers:  ...  # 特别说明下，这就是前一篇我们说到的include的base set啦，比如android.jar  -I add an existing package to base include set  ...  # overlay通过-S指定，可以指定多个目录，  -S directory in which to find resources. Multiple directories will be scanned  and the first match found (left to right) will take precedence.  ...  # 自动添加overlays包里的资源  --auto-add-overlay  Automatically add resources that are only in overlays.  ... |

举个例子

|  |  |
| --- | --- |
| 1 2 3 4 5 | aapt package \ -M AndroidManifest.xml \ -m -J gen \ -S src/com/example/res \ -S src/com/example/ui/res |

假如我们有如上的aapt命令输入，那么当src/com/example/res与src/com/example/ui/res有相同资源的时候，就会使用前者的，这里对资源替换的粒度是resource而不是文件，比如两个文件夹的values/string.xml都有对同一个string id的描述，最后就会使用前者的字符串。

然后我们再来看看--auto-add-overlay有什么用，

假如我们在src/com/example/ui/res定义了资源string a，但是在src/com/example/res却没有这个string，那就会报错，因为基础包里是没有那个资源的，这时候就需要加上--auto-add-overlay，于是就会自动把新的资源都添加进去。

overlay大致就是这么一回事啦。

# **Gradle实践**

## **aaptOptions**

google的官方文档简直说了和没说一样。还是自己来吧，用AS的模板新建一个Testapp工程，随便建两个res文件夹，各放两个strings.xml，结构为：

├── res

│ ├── drawable

│ ├── layout

│ │ ├── activity\_main.xml

│ │ └── content\_main.xml

│ ├── menu

│ │ └── menu\_main.xml

│ ├── mipmap-xxxhdpi

│ │ └── ic\_launcher.png

│ └── values

│ ├── colors.xml

│ ├── dimens.xml

│ ├── strings.xml

│ └── styles.xml

├── res2

│ └── values

│ └── strings.xml

└── res3

└── values

└── strings.xml

res2和res3分别定义了一个string hehe，value分别为hehe res2和hehe res3。

content\_main.xml的TextView使用了hehe（原来就是那个Hello World）。当然这里as会报错，因为res2和res3并没有标示为资源文件夹。

在module的build.gradle里:

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 | android {  ...  aaptOptions {  additionalParameters '-S',  '/Users/yifan/dev/github/Testapp/app/src/main/res3',  '-S',  '/Users/yifan/dev/github/Testapp/app/src/main/res2',  '--auto-add-overlay'  noCompress 'foo', 'bar'  ignoreAssetsPattern '!.svn:!.git:!.ds\_store:!\*.scc:.\*:<dir>\_\*:!CVS:!thumbs.db:!picasa.ini:!\*~'  }  ... } |

然后我们试图编译:

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 | All input files are considered out-of-date for incremental task ':app:processDebugResources'. Starting process 'command '/Users/yifan/dev/sdk/adt-bundle-mac-sdk/build-tools/23.0.2/aapt''. Working directory: /Users/yifan/dev/github/Testapp/app Command: /Users/yifan/dev/sdk/adt-bundle-mac-sdk/build-tools/23.0.2/aapt package -f --no-crunch -I /Users/yifan/dev/sdk/adt-bundle-mac-sdk/platforms/android-23/android.jar -M /Users/yifan/dev/github/Testapp/app/build/intermediates/manifests/full/debug/AndroidManifest.xml -S /Users/yifan/dev/github/Testapp/app/build/intermediates/res/merged/debug -A /Users/yifan/dev/github/Testapp/app/build/intermediates/assets/debug -m -J /Users/yifan/dev/github/Testapp/app/build/generated/source/r/debug -F /Users/yifan/dev/github/Testapp/app/build/intermediates/res/resources-debug.ap\_ --debug-mode --custom-package cn.zhaiyifan.testapp -0 apk -S /Users/yifan/dev/github/Testapp/app/src/main/res2 --output-text-symbols /Users/yifan/dev/github/Testapp/app/build/intermediates/symbols/debug Successfully started process 'command '/Users/yifan/dev/sdk/adt-bundle-mac-sdk/build-tools/23.0.2/aapt'' /Users/yifan/dev/github/Testapp/app/build/intermediates/res/merged/debug/values-af/values-af.xml:3 : AAPT: Resource at abc\_action\_bar\_home\_description appears in overlay but not in the base package; use <add-resource> to add. ... ...各种类似报错 /usr/local/google/buildbot/src/googleplex-android/mnc-supportlib-release/frameworks/support/v7/appcompat/res/color/switch\_thumb\_material\_light.xml:19 : AAPT: No resource found that matches the given name (at 'color' with value '@color/switch\_thumb\_normal\_material\_light').  :app:processDebugResources FAILED :app:processDebugResources (Thread[main,5,main]) completed. Took 10.493 secs. |

看到<add-resource>这个，大概知道啥问题了…于是在additionalParameters最后又加上了--auto-add-overlay，成功编译运行。

在屏幕中央，显示了hehe res3，交换-S顺序后则变成了hehe res2，符合我们第一节中说到的，选择首个匹配原则。

不仅是string，anim，layout等等资源都可以使用重叠包来进行动态指定。

## **资源合并**

和aapt的overlay有关，但使用场景略有不同，也介绍一下。

Google在Android Tools Project Site专门为此开了一个页面：[Resource Merging(资源合并)](http://tools.android.com/tech-docs/new-build-system/resource-merging" \t "http://blog.zhaiyifan.cn/2016/02/18/android-resource-overlay/_blank)。

在过去的编译系统中，资源合并是通过传给aapt一个作为重叠包的资源文件夹列表来做的，再加上–auto-add-overlay来确保在重叠包里的新资源会被自动添加（默认行为只会重载既有资源）。

基础Gradle的编译系统的一个目标就是提供更大的灵活性，而另一个经常并问到的功能要求则是能拥有多个资源文件夹。aapt无法去处理这个，所以新的编译系统引进了一种新的超越aapt的合并机制，生成一个单独的，合并的，资源文件夹并提供给aapt。这个合并机制拥有增量的优点，既因为Gradle的输入/输出变更检测，又因为其实现方式（可以只使用唯一一个变更文件来做重新merge）。

合并的资源来自3种来源：

1. 主资源，和main sourceSet相关联，大多位于src/main/res
2. Variant重叠包，来自Build Type和Flavor(s).
3. Library项目依赖，通过它们的aar bundle提供资源。

### **优先级**

优先级为：BuildType -> Flavor -> main -> Dependencies.

这意味着如果一个资源同时在Build Type和main存在，会使用Build Type里的。

需要注意的是合并的scope，同样（类型，名字）的资源但标示符不同的，是分开处理的。

即如果src/main/res有:

* res/layout/foo.xml
* res/layout-land/foo.xml  
  而src/debug/res有：
* res/layout/foo.xml

则合并后的资源文件夹会包含默认的来自src/debug/res的foo.xml，但横屏版本则会选择src/main/res下的。

PS: android的资源有19个维度，见[Grouping Resource Types](http://developer.android.com/guide/topics/resources/providing-resources.html" \l "AlternativeResources" \t "http://blog.zhaiyifan.cn/2016/02/18/android-resource-overlay/_blank)的Table 2，这19个维度会唯一指定1个资源（qualifier标示符）。在老罗的资源介绍博客中曾经提到过18个维度，现在变成了19是因为多了Round screen这个维度，用于描述Android Wear，添加于API 23.

### **处理多个资源文件夹**

每个sourceSet可以定义多个资源文件夹，举个例子:

android.sourceSets {

main.res.srcDirs = ['src/main/res', 'src/main/res2']

}

这种情况下，两个资源文件夹具有相同优先级，即如果一个资源在两个文件夹都声明了，合并会报错。

### **Library依赖的优先级顺序**

根据传递的依赖，Library项目的实际集被工程视为一个图，而不是平铺的列表，然后合并机制只会处理一个平优先级列表。

如果我们考虑如下例子的依赖关系：  
项目 -> A, B (意味着A的优先级高于B)  
A -> C, D  
B -> C  
则最后的优先级list为A, D, B, C，同时保证了A和B可以重载C。

### **小测试**

继续在之前我们建立的工程的基础上做个小测试吧。在sourceSet加上res2文件夹，最后build.gradle的android域如下:

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 | android {  compileSdkVersion 23  buildToolsVersion "23.0.2"   defaultConfig {  applicationId "cn.zhaiyifan.testapp"  minSdkVersion 14  targetSdkVersion 23  versionCode 1  versionName "1.0"  }  buildTypes {  release {  minifyEnabled false  proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'  }  }  aaptOptions {  additionalParameters '-S',  '/Users/yifan/dev/github/Testapp/app/src/main/res3',  '-S',  '/Users/yifan/dev/github/Testapp/app/src/main/res2',  '--auto-add-overlay'  noCompress 'foo', 'bar'  ignoreAssetsPattern '!.svn:!.git:!.ds\_store:!\*.scc:.\*:<dir>\_\*:!CVS:!thumbs.db:!picasa.ini:!\*~'  }  android.sourceSets {  main.res.srcDirs = ['src/main/res', 'src/main/res2']  } } |

运行后发现界面显示了 hehe res2，符合预期，因为res2已经和res合并了，所以先找到了build/intermediates/res/merged/debug下的string，没有用res3的。

# **使用场景**

重叠包可以让我们在不同的buildType、渠道下的包，使用不同的资源，做一些定制，而不用侵入代码本身的逻辑。

资源合并则可以帮助资源分组，比如按业务分组之类的，当然也能做重叠包能做的事（比如在不同的buildType申明不同的res.srcDirs）…所以你问我有了resource merge之后重叠包还有什么用…我还真没法回答，可能有什么奇异的场景会用到吧，毕竟Google也说了，这是一种超越aapt的合并机制。

# **总结**

我们了解了Android aapt overlay的机制，gradle下的资源合并是怎么一回事，并编写运行demo验证了资源应用两种机制后的结果。

# **参考文献**

* [https://android.googlesource.com/platform/frameworks/native/+/jb-dev/libs/utils/README](https://android.googlesource.com/platform/frameworks/native/+/jb-dev/libs/utils/README" \t "http://blog.zhaiyifan.cn/2016/02/18/android-resource-overlay/_blank)
* [http://blog.csdn.net/luoshengyang/article/details/8744683](http://blog.csdn.net/luoshengyang/article/details/8744683" \t "http://blog.zhaiyifan.cn/2016/02/18/android-resource-overlay/_blank)
* [http://tools.android.com/tech-docs/new-build-system/resource-merging](http://tools.android.com/tech-docs/new-build-system/resource-merging" \t "http://blog.zhaiyifan.cn/2016/02/18/android-resource-overlay/_blank)

[#Android](http://blog.zhaiyifan.cn/tags/Android/) [#Resource](http://blog.zhaiyifan.cn/tags/Resource/) [#aapt](http://blog.zhaiyifan.cn/tags/aapt/)