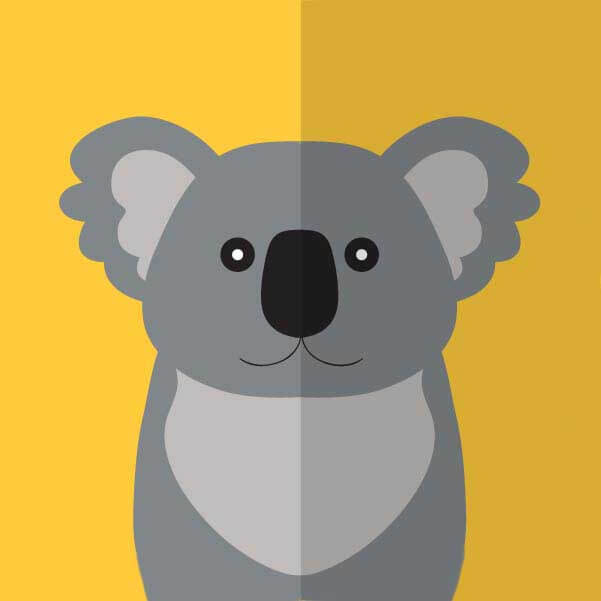
**Android 系统 overlay 机制重点小结**

[](https://www.jianshu.com/u/703d7600ceb3)

[qiracle](https://www.jianshu.com/u/703d7600ceb3) 关注

2017.12.18 09:37 字数 747 阅读 654评论 0喜欢 0

**静态 overlay 和动态 overlay**

静态 overlay：又称为编译时 overlay，编译时资源就已经覆盖了，一般用在有源码的apk中。  
动态 overlay：又称为运行时 overlay，当 apk 在手机中运行时才发生资源覆盖，一般用在无源码的apk中。

**overlay资源替换的前提和原则**

前提：资源所在路径必须与overlay下资源路径完全相同。如要替换 Settings 这个应用的String.xml里的资源，该资源文件所在路径为packages/apps/res/values/,则对应overlay的路径必须为overlay/packages/apps/res/values/。

原则：overlay替换的是资源，不是文件。举个例子，应用中String.xml里的内容如下：

<String name="a">aaa</String>

<String name="b">bbb</String>

<String name="c">ccc</String>

overlay中的String.xml里的内容如下：

<String name="a">abc</String>

则最终，apk调用的资源如下：

<String name="a">abc</String>

<String name="b">bbb</String>

<String name="c">ccc</String>

而不是想象的如下：

<String name="a">abc</String>

注意，关于替换，查询相关英文文档发现有如下描述

For color, bool, string, array, style/theme types, the resource values are identifed by their keys, so for these types, there is no need to put the resources in a file with the same name as in the original base package.

For layout, animation, picture drawables and raw types, the resources are indentifed by their file name, and overlay for these resources should keep the file name same as in the base packages.

翻译成中文的大概意思就是：  
1.对于color，bool，String，array，style等资源的值是有他们的键确定的，比如

<String name="a">abc</String>

该字符串资源通过键 name = "a" 来唯一确定值 abc 。也就是说，overlay里的这类资源文件的文件名不需要与应用包里的资源文件的文件名保持一致。只需要，资源文件里的键保持一致就行了。

2.对于布局文件，动画文件，图片资源文件等，这些资源文件是通过文件名来唯一确定，所以overlay里的这类资源文件需要与应用包里的资源文件的文件名保持一致。

**多个overlay的优先级判定**

在mk文件中通过定义PRODUCT\_PACKAGE\_OVERLAYS或DEVICE\_PACKAGE\_OVERLAYS变量，后面可以加上多个overlay目录路径，以此来实现多个overlay目录。但是这些目录是有优先级顺序的，PRODUCT\_PACKAGE\_OVERLAYS下的目录优先级高于DEVICE\_PACKAGE\_OVERLAYS下目录的优先级，写在前面的目录优先级高于写在后面目录的优先级，举个例子:

PRODUCT\_PACKAGE\_OVERLAYS = overlay\_A overlay\_B

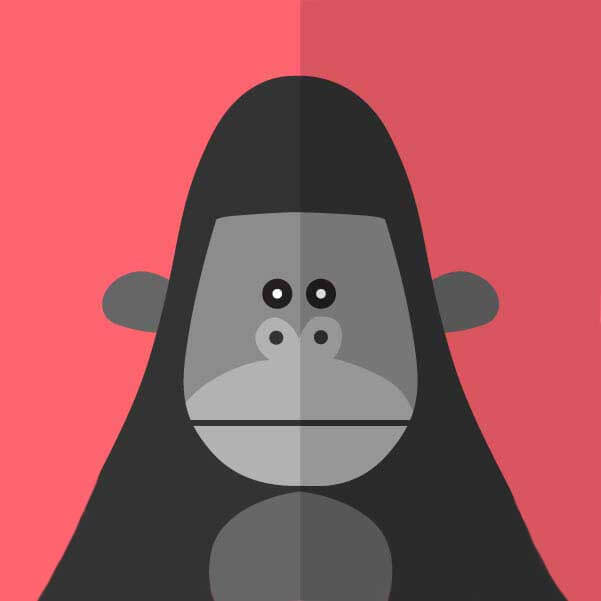
DEVICE\_PACKAGE\_OVERLAYS = overlay\_C overlay\_D

上述overlay目录优先级顺序:overlay\_A >overlay\_B> overlay\_C >overlay\_D

**overlay用于新增资源**

默认情况下，overlay目录的资源文件内容只能覆盖原有软件包中的资源，而不能新增资源。不让会造成编译错误。如要允许增加资源，可以将资源放入标签中，或者一种更加简便的方法是给aapt命令增加--auto-add-overlay选项。

# Android Overlay机制

[](https://www.jianshu.com/u/1a957b7a4c92)

[GrayMonkey](https://www.jianshu.com/u/1a957b7a4c92) 关注

2017.10.23 11:03\* 字数 2134 阅读 3110评论 0喜欢 9

#### 前言

Android Overlay是一种资源替换机制，它能在不重新打包apk的情况下，实现资源文件的替换（res目录非assert目录），Overlay又分为静态Overlay(Static Resource Overlay)与运行时Overlay(Runtime Resource Overlay)。

#### 源码编译流程简介

Overlay的定制需要在源码编译环境中完成，所以先简单介绍Android的源码编译流程。  
首先下载系统源码，配置好编译环境，然后进入源码根目录，在Linux shell终端中执行以下步骤即可。

图1-android源码目录结构

按照google给出的编译步骤如下：

1. source build/envsetup.sh  
   该步骤主要是将Shell脚本envsetup.sh中定义的一些函数命令加载到环境变量中，方便直接使用，source命令可替换成“.”命令。脚本文件的开头说明如下：

图2-envsetup.sh

1. lunch  
   该步骤主要是选择平台编译选项，该命令会罗列出所有平台编译选项，如下图，输入相应数字即可

图3-来自“手动编译源码”一文

参考图1中lunch命令说明，“build\_varaint”可分为以下几类：

| **build\_varaint** | **Description** |
| --- | --- |
| eng | 工程机，具备root权限以及一些debug工具 |
| user | 用户机，可正式销售给用户，无root权限、debug权限 |
| userdebug | 在user的基础上开放root、debug权限 |

1. make  
   执行编译，make命令会调用根目录下的Makefile文件，Makefile又包含 build/core/main.mk，main.mk又会包含build目录下其他的makefile文件来完成编译并打包得到系统镜像文件。  
   在这个编译过程会将packages/apps目录下的源码项目通过AAPT打包成APK，存放至system/app或者system/priv-app目录下，最后连同其他资源打包得到系统镜像，当然也可以将APK直接打包到系统镜像。

#### 静态Overlay

静态Overlay，简称为SRO，发生在编译时，需要在Android系统源码环境中进行配置。

##### 配置步骤

官方源码示例Demo如下：  
1.修改产品（backup应用）的mk文件，添加如下代码  
示例文件路径：  
android-6.0.0\_r1\device\sample\products\backup\_overlay.mk

图4-官方示例

PRODUCT\_PACKAGE\_OVERLAYS := device/sample/overlays/backup

其中，AndroidProducts.mk定义了哪些APP会被编译打包，其他mk文件则是对应APP的配置文件。  
一般厂商都会有一些个性化定制的产品，会参照device目录结构创建一个vendor目录，所以有的产品的overlay需要在这个vendor目录中进行修改。

2.在overlays目录下添加对应产品的overlay项目，overlay项目实际上就是源项目剔除代码和替换资源后的项目，注意包名路径需要与源项目保持一致。

#### 原理

实际上，SRO的实现很简单，只是在通过AAPT打包成APK时，通过-S命令多增加了一个资源目录(overlay目录)而已，我们平时打包APK只是通过AAPT -S指定了一个资源目录，而Overlay又额外增加了一个资源目录而已。  
AAPT -S命令可以指定多个资源目录，overlay的资源将替换原res中的重名文件。

图5-SRO

可见SRO在编译成APK时已经完成。

#### 运行时Overlay

运行时Overlay，简称RRO，顾名思义，该机制的资源替换发生在运行时。

##### 与SRO的区别

1. RRO能直接定制替换第三方APK的资源，而不需要其源码。SRO如上节所述，则需要对应APK的源码才能完成，一般而言，第三方是不会提供项目源码的。
2. RRO的编译结果会得到一个xxx\_overlay.apk,加上原项目的apk，总共会有2个apk，而SRO最终只会得到一个已经完成资源替换的apk。得到的overlay.apk可以视为一个正常的apk，因为它能被安装，含有自己的AndroidManifest.xml文件，当然正常下，overlay.apk是不含有执行代码的。
3. **RRO不能替换AndroidManifest.xml文件及reference resource 类型的文件，如layout、anim、xml目录中的xml文件**。虽然RRO具有自己的AndroidManifest.xml文件，但它却不能替换源项目中的AndroidManifest.xml文件。关于layout目录中的xml文件，SRO是可以替换的。

//以下几种，就属于reference resource

Resources resources = getResources();

XmlResourceParser parser0 = resources.getXml(R.xml.test);

XmlResourceParser parser1 = resources.getLayout(R.layout.activity\_main);

XmlResourceParser parser2 = resources.getAnimation(R.anim.test);

##### 配置步骤

下面以创建Launcher3的RRO为例进行说明

1. 创建一个新项目，包名命名为com.android.launcher3.overlay,事实上包名可以随意命名，这样命名可读性高，一看包名就知道是哪个项目的overlay。
2. 编辑overlay项目的AndroidManifest.xml文件,文件内容如下：

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

package="com.android.launcher3.overlay">

<overlay android:targetPackage="com.android.launcher3" android:priority="1"/>

</manifest>

其中，  
android:targetPackage：需要overlay的项目的包名  
android:priority：设置overlay.apk的优先级，值越大，优先级越高，用于存在多个overlay.apk情况下的判断。

1. 替换相应的res资源
2. 编写mk文件，编译、打包、签名，并将生成的overlay.apk输出到/vendor/overlay目录下，其中签名需要与源项目签名一致，否则不会生效。  
   **注意:**  
   把app放到vendor/overlay下面以后，还要activate，默认是disable的  
   activate有两种方式:  
   一种是overlay的app的清单文件中，overlay标签写 isStatic="true"  
   一种是加系统属性ro.boot.vendor.overlay.theme=overlay的app的包名，分号分割

PS:没有源码环境，或者不懂mk文件，可以通过root手机，直接将我们生成的overlay.apk直接push到/vendor/overlay目录下，来测试RRO机制。  
部分厂商可能更改了路径，可通过adb shell pm path xxxx.overlay 查找到路径，前提是手机中已经安装了某个overlay的apk。

##### 资源ID

我们知道资源文件在打包之后，会在R.java文件中生成一个对应的int类型资源id，这个id遵循以下的规则：  
**0xppttiiii**  
其中，  
**pp**：01表示是系统资源，7f表示是应用资源  
**tt**：用于表示资源类型，如string、array、图片资源等  
**iiii**：用于表示相同类型的不同资源

##### 原理

以我们常见的图片资源访问为例，当我们在应用层调用getDrawable方法的时候，会调用系统的getResource方法，该方法会根据资源id及设备配置信息（语言、像素密度等）来查找并返回合适的图片资源。此处，我们只关注根据id来选择资源。

如果id表明是系统资源，那么就会去framework-res.apk中查找并返回资源，一般位于/system/framework目录下，可root手机查看；如果是应用资源则在overlay和app中查找并返回。

图6-RRO流程

###### 单个overlay包

当对应项目只有一个overlay包时，那么查找资源时，会先从overlay.apk中进行查找，查找成功直接返回，反之则从app中查找。

###### 多个overlay包

当对应项目具有多个overlay包时，会按照overlay包的优先级从高到底依次进行查找，如果overlay包都查找失败，才会到app中进行查找。  
以访问R.drawable.ic\_01为例，会先从overlay1包先查找，因为其优先级最高，查找到则直接返回。  
如果访问R.drawable.ic\_02，先从overlay1查找，查找失败，接着从overlay2中查找，查找到直接返回。  
如果访问R.drawable.ic\_03，依次从overlay1，overlay2中查找，查找失败，最后会从app中查找到，并返回。

#### 总结

SRO实际上只是利用AAPT重新打包，发生在编译时；RRO，才是overlay机制的关键，其本质是Android系统的动态资源查找机制。  
实际运用当中，Overlay机制一般用于手机厂商为不同运营商做客制化定制，或者做主题换肤；一般，第三方应用是无法享受该机制的，除非与手机厂商合作，不过应用层现在也有了换肤框架[Android-Skin-Loader](https://github.com/fengjundev/Android-Skin-Loader)，其本质也是利用Android的资源查找机制。

【参考文章】  
[Sony contributes Runtime Resource Overlay framework to Android](https://developer.sonymobile.com/2014/04/22/sony-contributes-runtime-resource-overlay-framework-to-android-code-example/)  
[make file](http://blog.csdn.net/ruglcc/article/details/7814546)  
[Android源码编译流程](http://www.cnblogs.com/mr-raptor/archive/2012/06/12/2547030.html)  
<https://www.ibm.com/developerworks/cn/opensource/os-cn-android-build/index.html>

# Android动态改变App在Launcher里面的icon

2016年12月31日 13:08:13 [火山石](https://me.csdn.net/zhangcanyan) 阅读数：4449

如果呆萌的产品童鞋让你动态更换App在Launcher里面的Icon，你怎么回答他，下文就提出一种实现该效果的方法。

原理1——activity-alias

在AndroidMainifest中，有两个属性：

// 决定应用程序最先启动的Activity

android.intent.action.MAIN

// 决定应用程序是否显示在程序列表里

android.intent.category.LAUNCHER

另外，还有一个activity-alias属性，这个属性可以用于创建多个不同的入口，相信做过系统Setting和Launcher开发的开发者在系统的源码中应该见过很多。

原理2——PM.setComponentEnabledSetting

PackageManager是一个大统领类，可以管理所有的系统组件，当然，如果Root了，你还可以管理其它App的所有组件，一些系统优化工具就是通过这个方式来禁用一些后台Service的。

使用方式异常简单：

private void enableComponent(ComponentName componentName) {

    mPm.setComponentEnabledSetting(componentName,

            PackageManager.COMPONENT\_ENABLED\_STATE\_ENABLED,

            PackageManager.DONT\_KILL\_APP);

}

private void disableComponent(ComponentName componentName) {

    mPm.setComponentEnabledSetting(componentName,

            PackageManager.COMPONENT\_ENABLED\_STATE\_DISABLED,

            PackageManager.DONT\_KILL\_APP);

}

根据PackageManager.COMPONENT\_ENABLED\_STATE\_ENABLED和PackageManager.COMPONENT\_ENABLED\_STATE\_DISABLED这两个标志量和对应的ComponentName，就可以控制一个组件的是否启用。

动态换Icon

有了上面的两个原理，来实现动态更换Icon就只剩下思路问题了。

首先，我们创建一个Activity，作为默认的入口并带着默认的图片，再创建一个双11的activity-alias，指向默认的Activity并带有双11的图片，再创建一个双12的activity-alias，指向默认的Activity并带有双12的图片……等等等。

<activity android:name=".MainActivity">

    <intent-filter>

        <action android:name="android.intent.action.MAIN"/>

        <category android:name="android.intent.category.LAUNCHER"/>

    </intent-filter>

</activity>

<activity-alias

    android:name=".Test11"

    android:enabled="false"

    android:icon="@drawable/s11"

    android:label="双11"

    android:targetActivity=".MainActivity">

    <intent-filter>

        <action android:name="android.intent.action.MAIN"/>

        <category android:name="android.intent.category.LAUNCHER"/>

    </intent-filter>

</activity-alias>

<activity-alias

    android:name=".Test12"

    android:enabled="false"

    android:icon="@drawable/s12"

    android:label="双12"

    android:targetActivity=".MainActivity">

    <intent-filter>

        <action android:name="android.intent.action.MAIN"/>

        <category android:name="android.intent.category.LAUNCHER"/>

    </intent-filter>

</activity-alias>

等等，这样有个问题，那就是这样会在Launcher上显示3个入口，所以，默认我们会把这些activity-alias先禁用，等到要用的时候再启用，养兵千日，用兵一时。

public class MainActivity extends AppCompatActivity {

    private ComponentName mDefault;

    private ComponentName mDouble11;

    private ComponentName mDouble12;

    private PackageManager mPm;

    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity\_main);

        mDefault = getComponentName();

        mDouble11 = new ComponentName(

                getBaseContext(),

                "com.xys.changeicon.Test11");

        mDouble12 = new ComponentName(

                getBaseContext(),

                "com.xys.changeicon.Test12");

        mPm = getApplicationContext().getPackageManager();

    }

    public void changeIcon11(View view) {

        disableComponent(mDefault);

        disableComponent(mDouble12);

        enableComponent(mDouble11);

    }

    public void changeIcon12(View view) {

        disableComponent(mDefault);

        disableComponent(mDouble11);

        enableComponent(mDouble12);

    }

    private void enableComponent(ComponentName componentName) {

        mPm.setComponentEnabledSetting(componentName,

                PackageManager.COMPONENT\_ENABLED\_STATE\_ENABLED,

                PackageManager.DONT\_KILL\_APP);

    }

    private void disableComponent(ComponentName componentName) {

        mPm.setComponentEnabledSetting(componentName,

                PackageManager.COMPONENT\_ENABLED\_STATE\_DISABLED,

                PackageManager.DONT\_KILL\_APP);

    }

}

OK了，禁用默认的Activity后，启用双11的activity-alias，结果不变还是指向了默认的Activity，但图标已经发生了改变。

根据ROM的不同，在禁用了组件之后，会等一会，Launcher会自动刷新图标。

产品：我们可以动态更换App在Launcher里面的Icon吗

开发：让我想想……

**原理1——activity-alias**

在AndroidMainifest中，有两个属性：

// 决定应用程序最先启动的Activity

android.intent.action.MAIN

// 决定应用程序是否显示在程序列表里

android.intent.category.LAUNCHER

另外，还有一个activity-alias属性，这个属性可以用于创建多个不同的入口，相信做过系统Setting和Launcher开发的开发者在系统的源码中应该见过很多。

**原理2——PM.setComponentEnabledSetting**

PackageManager是一个大统领类，可以管理所有的系统组件，当然，如果Root了，你还可以管理其它App的所有组件，一些系统优化工具就是通过这个方式来禁用一些后台Service的。

使用方式异常简单：

private void enableComponent(ComponentName componentName) {

    mPm.setComponentEnabledSetting(componentName,

            PackageManager.COMPONENT\_ENABLED\_STATE\_ENABLED,

            PackageManager.DONT\_KILL\_APP);

}

private void disableComponent(ComponentName componentName) {

    mPm.setComponentEnabledSetting(componentName,

            PackageManager.COMPONENT\_ENABLED\_STATE\_DISABLED,

            PackageManager.DONT\_KILL\_APP);

}

根据PackageManager.COMPONENT\_ENABLED\_STATE\_ENABLED和PackageManager.COMPONENT\_ENABLED\_STATE\_DISABLED这两个标志量和对应的ComponentName，就可以控制一个组件的是否启用。

**动态换Icon**

有了上面的两个原理，来实现动态更换Icon就只剩下思路问题了。

首先，我们创建一个Activity，作为默认的入口并带着默认的图片，再创建一个双11的activity-alias，指向默认的Activity并带有双11的图片，再创建一个双12的activity-alias，指向默认的Activity并带有双12的图片……等等等。

<activity android:name=".MainActivity">

    <intent-filter>

        <action android:name="android.intent.action.MAIN"/>

        <category android:name="android.intent.category.LAUNCHER"/>

    </intent-filter>

</activity>

<activity-alias

    android:name=".Test11"

    android:enabled="false"

    android:icon="@drawable/s11"

    android:label="双11"

    android:targetActivity=".MainActivity">

    <intent-filter>

        <action android:name="android.intent.action.MAIN"/>

        <category android:name="android.intent.category.LAUNCHER"/>

    </intent-filter>

</activity-alias>

<activity-alias

    android:name=".Test12"

    android:enabled="false"

    android:icon="@drawable/s12"

    android:label="双12"

    android:targetActivity=".MainActivity">

    <intent-filter>

        <action android:name="android.intent.action.MAIN"/>

        <category android:name="android.intent.category.LAUNCHER"/>

    </intent-filter>

</activity-alias>

等等，这样有个问题，那就是这样会在Launcher上显示3个入口，所以，默认我们会把这些activity-alias先禁用，等到要用的时候再启用，养兵千日，用兵一时。

public class MainActivity extends AppCompatActivity {

    private ComponentName mDefault;

    private ComponentName mDouble11;

    private ComponentName mDouble12;

    private PackageManager mPm;

    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity\_main);

        mDefault = getComponentName();

        mDouble11 = new ComponentName(

                getBaseContext(),

                "com.xys.changeicon.Test11");

        mDouble12 = new ComponentName(

                getBaseContext(),

                "com.xys.changeicon.Test12");

        mPm = getApplicationContext().getPackageManager();

    }

    public void changeIcon11(View view) {

        disableComponent(mDefault);

        disableComponent(mDouble12);

        enableComponent(mDouble11);

    }

    public void changeIcon12(View view) {

        disableComponent(mDefault);

        disableComponent(mDouble11);

        enableComponent(mDouble12);

    }

    private void enableComponent(ComponentName componentName) {

        mPm.setComponentEnabledSetting(componentName,

                PackageManager.COMPONENT\_ENABLED\_STATE\_ENABLED,

                PackageManager.DONT\_KILL\_APP);

    }

    private void disableComponent(ComponentName componentName) {

        mPm.setComponentEnabledSetting(componentName,

                PackageManager.COMPONENT\_ENABLED\_STATE\_DISABLED,

                PackageManager.DONT\_KILL\_APP);

    }

}

OK了，禁用默认的Activity后，启用双11的activity-alias，结果不变还是指向了默认的Activity，但图标已经发生了改变。

根据ROM的不同，在禁用了组件之后，会等一会，Launcher会自动刷新图标。