Android Style & Attr详解

笔记本:	里克分享		
创建时间:	2015/12/22 11:10	**更新时间:**	2015/12/24 11:30
作者:	里克Rick_tan		

里克Note:7

1、先看一个调用栈:

```
call Resources::applyStyle resid=0x7a0e005f force=true
java.lang.Exception: here
    at android.content.res.Resources$Theme.applyStyle(Resources.java:1308)
    at android.view.ContextThemeWrapper.onApplyThemeResource(ContextThemeWrapper.java:132)
    at android.app.Activity.onApplyThemeResource(Activity.java:3376)
    at android.view.ContextThemeWrapper.initializeTheme(ContextThemeWrapper.java:144)
    at android.view.ContextThemeWrapper.setTheme(ContextThemeWrapper.java:89)
    at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2144)
    at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:2233)
    at android.app.ActivityThread.access$800(ActivityThread.java:135)
    at android.app.ActivityThread$H.handleMessage(ActivityThread.java:1196)
    at android.os.Handler.dispatchMessage(Handler.java:102)
    at android.os.Looper.loop(Looper.java:136)
    at android.app.ActivityThread.main(ActivityThread.java:5001)
    at java.lang.reflect.Method.invokeNative(Native Method)
    at java.lang.reflect.Method.invoke(Method.java:515)
    at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:808)
    at com.android.internal.os.ZygoteInit.tos_org_main(ZygoteInit.java:624)
    at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:578)
    at dalvik.system.NativeStart.main(Native Method)
```

在activity启动的时候就根据AndroidManifest.xml设置好了Theme(activity如果没设置主题默认采用application的)，这个Theme里面包含了应用/activity需要的Style。

resid=0x7a0e005f ⟶

QromApiDemos Manifest

```
47    <application
48        android:allowBackup="true"
49        android:icon="@drawable/ic_launcher"
50        android:label="@string/app_name"
51        android:theme="@com.tencent.qrom:style/Theme.qrom.Second" >
```

重点:

```
@param resid The resource ID of a style resource from which to obtain attribute values.
```

`AssetManager.applyThemeStyle( mTheme, resid, force);`通过这个api将theme下面的这个Style的内容保存为一个二维数组[attr][style]映射关系，后面可以通过R.attr.id来直接映射成defStyle。

2、系统控件自定义属性，init（扩展EditText一个属性：showClear）

```
public QromEditText(Context context, AttributeSet attrs) {
    this(context, attrs, com.tencent.qrom.R.attr.qromEditTextStyle);
}
```

init<constructor>会根据之前设置的Theme(Style)，通过`qromEditTextStyle`获取当前init的控件待初始化属性 - 找到对应的Style来获取content - xml对应的初始化值/默认值。

点解：

`qromEditTextStyle`是attrs.xml下的一个attr - id 基本都是一个引用，通过这个id来获取对应的defStyle。

↓

```
public QromEditText(Context context, AttributeSet attrs, int defStyle) {
    super(context, attrs, defStyle);
    final TypedArray editAttributes = context.obtainStyledAttributes(attrs, com.tencent.qrom.R.styleable.QromEditText, defStyle, 0);
    showClearActionBtn = editAttributes.getBoolean(com.tencent.qrom.R.styleable.QromEditText_showClear, true);
```

会通过系统API `obtainStyledAttributes` 来获取关联，eg:如上面 就是根据`R.styleable.QromEditText`来填充到结果:TypeArray。

theme.xml/Style.xml

```
<item name="qromEditTextStyle">@style/Widget.qrom.EditText.QromEditTextStyle</item>
```

```
<style name="Widget.qrom.EditText.QromEditTextStyle">
    <item name="showClear">true</item>
</style>
```

attrs.xml

```
<attr name="qromEditTextStyle" format="reference"/>
```

```
<declare-styleable name="QromEditText">
    <attr name="showClear" format="boolean"/>
</declare-styleable>
```

3、app自定义Style

请查阅https://github.com/rickdynasty/RickBurgeonDemo工程的BurEditText

流程：1、继承原生控件，扩展自己需要的Style属性

```
<style name="BurEditTextStyle" parent="@android:style/Widget.EditText">
    <item name="showClear">true</item>
</style>
```

2、将Style添加到当前应用的Theme里面

```
<style name="AppTheme" parent="AppBaseTheme">
    <item name="burEditTextStyle">@style/BurEditTextStyle</item>
</style>
```

3、添加attr reference 条目 - 用于获取defStyle【就是第一步定义的Style】

```
<attr name="burEditTextStyle" format="reference"/>
```

4、添加declare-styleable 扩展属性集

```
<declare-styleable name="BurEditText">
    <attr name="showClear" format="boolean"/>
</declare-styleable>
```

note:step1扩展的属性条目和step4的属性集是对应的

4、AttributeSet attrs的初始化流程

at android.view.LayoutInflater.createView(LayoutInflater.java:594)
at com.android.internal.policy.impl.PhoneLayoutInflater.onCreateView(PhoneLayoutInflater.java:56)
at android.view.LayoutInflater.onCreateView(LayoutInflater.java:669)
at android.view.LayoutInflater.createViewFromTag(LayoutInflater.java:694)
at android.view.LayoutInflater.rInflate(LayoutInflater.java:755)
at android.view.LayoutInflater.inflate(LayoutInflater.java:492)
at android.view.LayoutInflater.inflate(LayoutInflater.java:397)
at android.view.LayoutInflater.inflate(LayoutInflater.java:353)
at com.android.internal.policy.impl.PhoneWindow.setContentView(PhoneWindow.java:293)
at android.app.Activity.setContentView(Activity.java:1944)

上面的栈很明确的说明了这个过程；大体如下：activity 通过set一个xml的布局 → 然后LayoutInflater去解析这个xml:

①、通过resource id获取parser

```
XmlResourceParser parser = getContext().getResources().getLayout(resource);
```

②、通过parser 先获取AttributeSet

```
// XmlBlock$Parser implements XmlResourceParser [extends XmlPullParser, AttributeSet, AutoCloseable]
final AttributeSet attrs = Xml.asAttributeSet (parser);
```

②、通过parser 来遍历布局里面的需要的View逐个createViewFromTag

```
void rInflate (XmlPullParser parser, View parent, final AttributeSet attrs, boolean finishInflate) throws XmlPullParserException,
IOException {
        final int depth = parser.getDepth();
        int type;
        while (((type = parser.next()) != XmlPullParser.END_TAG || parser.getDepth() > depth) && type != XmlPullParser.END_DOCUMENT) {
            ...
            final String name = parser.getName();

            if (TAG_REQUEST_FOCUS .equals(name)) {
                parseRequestFocus(parser, parent);
            } else if (TAG_INCLUDE.equals(name)) {
                if (parser.getDepth() == 0) {
                    throw new InflateException("<include /> cannot be the root element");
                }
```

```
                parseInclude(parser, parent, attrs);
            } else if (TAG_MERGE.equals(name)) {
                throw new InflateException("<merge /> must be the root element");
            } else if (TAG_1995.equals(name)) {
                final View view = new BlinkLayout(mContext, attrs);
                final ViewGroup viewGroup = (ViewGroup) parent;
                final ViewGroup.LayoutParams params = viewGroup.generateLayoutParams(attrs);
                rInflate(parser, view, attrs, true);
                viewGroup.addView(view, params);
            } else {
                final View view = createViewFromTag(parent, name, attrs);
                final ViewGroup viewGroup = (ViewGroup) parent;
                final ViewGroup.LayoutParams params = viewGroup.generateLayoutParams(attrs);
                rInflate(parser, view, attrs, true);
                viewGroup.addView(view, params);
            }
        }

        if (finishInflate) parent.onFinishInflate();
    }
```

5、部分细节点解

Style可以通过"."来继承 也可以直接通过指定父类来继承，eg: `parent="`，不管哪种都是以类的为优先，都会递归存入映射数组里面。

styleable是一个容器，在public里面没有对应的id,但可以通过attr来直接引用，得到的是这个容器里面的atrr属性id数组。

概要

一、控件属性 attr

1、控件属性是什么?
控件属性是具有唯一标识的32位整型键值。
2、它是怎样关联生效的呢？ obtainStyledAttributes
系统已经为我们提供了在三种场景下获取控件属性的接口。
基本流程：
第一步，提供 attr 列表 int[]，可以自定义数组进行组合，也可以使用由 aapt 预编译产生的 R.styleable.*；
第二步，根据 attr 列表，从 AttributeSet 集中获取目标子集，缓存类型数据 TypedArray；
第三步，根据 attr 列表索引顺序，获取有效值。

二、attr 属性的集合，主题与样式 theme & style

1、先说说样式，declare-styleable 与 style 区别
2、再说说主题 theme， theme 不是 Theme
3、那么问题来了，干贷！如何快速制作 ppt?
其实，Android 主题与样式的设计原则，与 web 设计中样式（stylesheets），与如何快速制作 ppt 中的母版是相通的。UI 通过可以设计一套母版样式（declare-styleable），提供多套主题风格（style），最终有效地将设计工作与开发工作隔离，达到一键换肤的效果。

三、在 lib 工程中使用 attr

1、先抛问题
如何实现 UI SDK 控件库？既能封装 UI 控件，不提供源码、只提供 .jar 包供上层使用。又能不出错地充分利用在 .xml 进行 UI 布局的灵活优势。
2、解决问题
官方的解决方案是，建立 lib 库依赖。此时，aapt 的预编译工作是不同的。对于 lib 工程，在产生的 R.java ， R.attr.* 并不是常量，不会在编译时优化为常数
3、不依赖 R.java 的方案
1） 使用资源名称。Resource 类提供了 int getIdentifier (String name, String defType, String defPackage) 动态获取资源 ID。缺点是，由于是运行时获取，出现错误不易发现；且命名难于维护。
2） 使用 publics.xml 约定资源 ID。优点是，可在编译时优化，源码、资源名称都可以混淆。缺陷是，当双方资源 ID 可能造成冲突。

四、其他

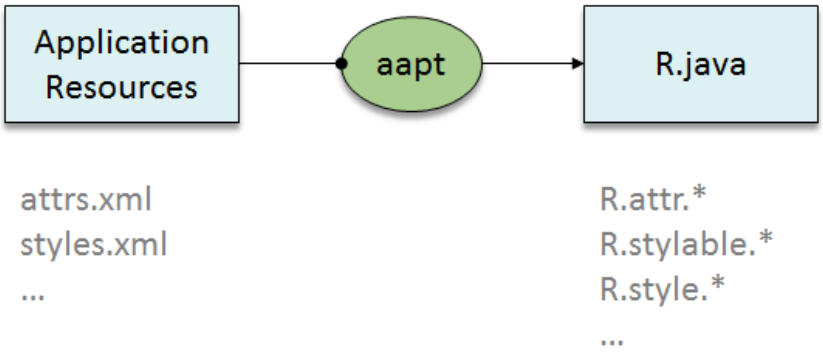
1、资源 ID 格式
2、资源混淆

3、TODO: 尝试使用 aapt 生成 0x02TTNNNN 的资源 ID，类似 android.jar SDK 框架提供 publics.xml，解决 R.java 依赖问题，资源冲突问题与资源混淆问题。

一、控件属性 attr

1、控件属性是什么?

控件属性是具有唯一标识的32位整型键值。通过在 .xml 中定义 <attr /> 属性，经过 aapt 预编译自动产生。.xml 中引用格式为 [package:]<attr-name>，代码中通过 R.attr.* 引用：



控件属性可以单独定义，也可以内嵌在 declare-styleable 中定义。attr 只能被定义一次，否则 aapt 预编译出错。例如：

```
<resources>

    <attr name="attr_explicit_demo" format="reference" />

    <declare-styleable name="stylable_demo">
        <attr name="android:id" />
        <attr name="attr_explicit_demo" />
        <attr name="attr_implicit_demo" format="reference" />
    </declare-styleable>

</resources>
```

控件属性与其他资源 ID 一样，可以被声明、指定为对外使用（publics.xml）的固定值，通常用于 Android 平台属性：

```
<public type="attr" name="text" id="0x0101014f" />
```

2、它是怎样关联生效的呢? obtainStyledAttributes

系统已经为我们提供了在三种场景下获取控件属性的接口：

```
// Theme Scene:public TypedArray obtainStyledAttributes(int[] attrs)
// Style Scene:public TypedArray obtainStyledAttributes(int resid, int[] attrs)
//  View  Scene:public  TypedArray  obtainStyledAttributes(AttributeSet  set,  int[]  attrs,  int  defStyleAttr,  int
defStyleRes)
```

以从 .xml 加载视图为例，基本流程：

第一步，提供 attr 列表 int[]，可以自定义数组进行组合，也可以使用由 aapt 预编译产生的 R.styleable.*；

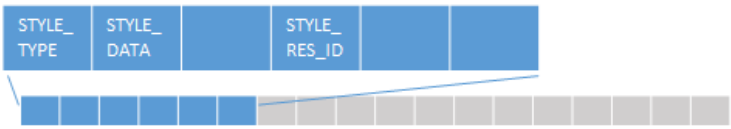
第二步，根据 attr 列表，从 AttributeSet 集中获取目标子集，缓存类型数据 TypedArray；

第三步，根据 attr 列表索引顺序，获取有效值。

示例代码片断：

```
    public SwitchImageView(Context context, AttributeSet attrs) {
        super(context, attrs);
        TypedArray a = null;
        int N = 0;

        a = context.obtainStyledAttributes(attrs, R.styleable.stylable_demo,
//              R.attr.attr_SwitchImageView_defStyleAttr,
                0,
                R.style.SwitchImageView_defStyleRes);
        N = a.getIndexCount();
        for (int i = 0; i < N; i++) {
            int attr = a.getIndex(i);
            switch (attr) {
                case R.styleable.stylable_demo_android_id:
                    Log.i(TAG, "id: " + Integer.toHexString(a.getResourceId(attr, NO_ID)));
                    break;
                case R.styleable.stylable_demo_attr_explicit_demo:
                    Log.i(TAG, "attr_explicit_demo");
                    break;
                case R.styleable.stylable_demo_attr_implicit_demo:
                    Log.i(TAG, "attr_implicit_demo");
                    break;
            }
        }
        a.recycle();
```

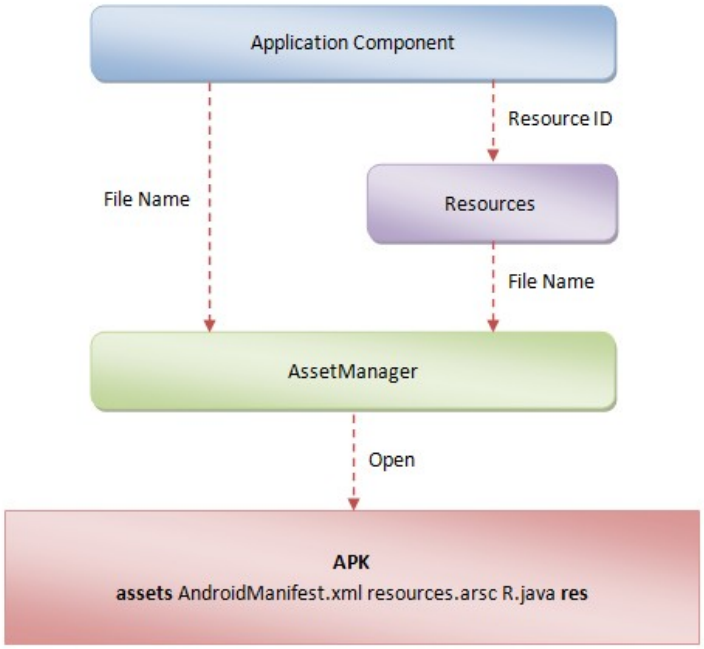
TypedArray 是格式化的缓存数组：



通过主题访问 token mTheme，AttributeSet 解析 token parser.mParseState，交由底层（JNI） AssetManager 解析填充具体数据：

```
//
https://android.googlesource.com/platform/frameworks/base/+/refs/heads/master/core/java/android/content/res/Resources.java


public TypedArray obtainStyledAttributes(AttributeSet set,
                int[] attrs, int defStyleAttr, int defStyleRes) {
            final int len = attrs.length;
            final TypedArray array = TypedArray.obtain(Resources.this, len);
            // XXX note that for now we only work with compiled XML files.
            // To support generic XML files we will need to manually parse
            // out the attributes from the XML file (applying type information
            // contained in the resources and such).
            final XmlBlock.Parser parser = (XmlBlock.Parser)set;
            AssetManager.applyStyle(mTheme, defStyleAttr, defStyleRes,
                    parser != null ? parser.mParseState : 0, attrs, array.mData, array.mIndices);
```

获取 TypedArray 需依赖 aapt 预编译的资源 ID。AssetManager 访问 apk 资源映射表 resources.arsc，解析出正确的文件路径，最终获取对应资源 [1]：



二、attr 属性的集合，主题与样式 theme & style

1、先说说样式，declare-styleable 与 style 区别

declare-styleable 是一组 attr 属性集合的声明，在 R.java 中产生 int[] 数组，并建立数组索引，索引内容为对应的 attr 属性：

```
        public static final int[] stylable_demo = {
            0x010100d0, 0x7f010001, 0x7f010004
        };

        public static final int stylable_demo_android_id = 0;
        public static final int stylable_demo_attr_explicit_demo = 1;
        public static final int stylable_demo_attr_implicit_demo = 2;
```

而 style 是预置的 attr / value 键值对集合，支持样式继承 [2]，如在 /res/value/styles.xml 中自定义主题：

```
<resources>

    <style name="AppBaseTheme" parent="@android:style/Theme.Holo.Light.DarkActionBar"></style>

    <style name="AppTheme" parent="AppBaseTheme">
        <item name="attr_SwitchImageView_defStyleAttr">@style/SwitchImageView_defStyleRes_ThemeDefault</item>
    </style>

    <style name="SwitchImageView_defStyleRes_ThemeDefault">
    </style>

</resources>
```

2、再说说主题 theme， theme 不是 Theme

通常，会在应用配置 AndroidManifest.xml 中指定基础主题：

```
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
```

对比 /res/value/styles.xml 中的 @android:style/Theme，同为主题，但它们本质不一样。

android:theme 是 declare-styleable，而 @android:style/Theme 是 style，Android 预制的默认样式集合，一般为其他自定义主题的父集合：

```
<!-- The default theme for apps on API level 10 and lower. This is the theme used for
         activities that have not explicitly set their own theme.
         <p>You can count on this being a dark
         background with light text on top, but should try to make no
         other assumptions about its appearance. In particular, the text
         inside of widgets using this theme may be completely different,
         with the widget container being a light color and the text on top
         of it a dark color.
         <p>If you're developing for API level 11 and higher, you should instead use {@link
         #Theme_Holo} or {@link #Theme_DeviceDefault}.</p>
    -->
    <style name="Theme">
```

3、那么问题来了，干贷！如何快速制作 ppt?

其实，Android 主题与样式的设计原则，与 web 设计中样式（stylesheets），与如何快速制作 ppt 中的母版是相通的。UI 通过可以设计一套母版样式（declare-styleable），提供多套主题风格（style），最终有效地将设计工作与开发工作隔离，达到一键换肤的效果。

三、在 lib 工程中使用 attr

1、先抛问题

在使用的时候，它有哪些让人困惑的地方？引用 lib 工程的 attr，属性设置无效最让人困惑和抓狂！

延伸问题，如何实现 UI SDK 控件库？既能封装 UI 控件，不提供源码、只提供 .jar 包供上层使用。又能不出错地充分利用在 .xml 进行 UI 布局的灵活优势。

2、解决问题

官方的解决方案是，建立 lib 库依赖。此时，aapt 的预编译工作是不同的。对于 lib 工程，在产生的 R.java ， R.attr.* 并不是常量，不会在编译时优化为常数：

```
/* AUTO-GENERATED FILE.  DO NOT MODIFY.
 */

package com.tencent.sdk;

public final class R {
    public static final class attr {
        public static int grayscale=0x7f010002;
```

在主工程中，aapt 会合并 lib 库工程的，生产2个 R.java 文件：

```
// com.tencent.demo.R

/* AUTO-GENERATED FILE.  DO NOT MODIFY.
 */

package com.tencent.demo;

public final class R {
    public static final class attr {
        public static final int attr_SwitchImageView_defStyleAttr=0x7f010007;
        public static final int attr_explicit_demo=0x7f010005;
        public static final int attr_implicit_demo=0x7f010008;
        public static final int attr_protected_demo=0x7f010006;
        public static final int attr_public_demo=0x7f010000;

        public static final int grayscale=0x7f010002;
```

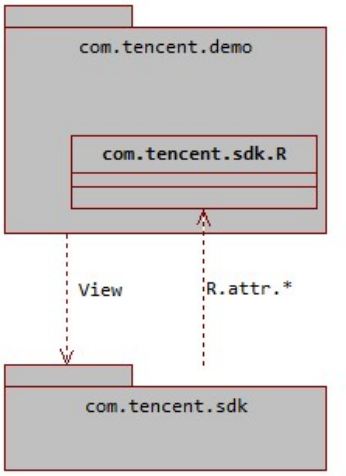
```
// com.tencent.sdk.R

/* AUTO-GENERATED FILE.  DO NOT MODIFY.
 */package com.tencent.sdk;

public final class R {
        public static final class attr {
                public static final int grayscale = 0x7f010002;
```

对比 lib 库工程与主工程的 com.tencnet.sdk.R，会发现 final 修饰符的区别。它的意义在于：

一是，解决了主工程与 lib 库工程的资源 ID 冲突；

二是，lib 库工程反向依赖主工程的 com.tencent.sdk.R，主工程在资源 ID 分配上更加灵活。

如图示：



建立了依赖关系后，便可以在 .xml 指定命名空间 xmlns:lib 访问控件属性：

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:lib="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.tencent.demo.MainActivity" >

    <com.tencent.pb.widget.DecorImageView
        android:layout_width="256dp"
        android:layout_height="256dp"
        android:layout_centerInParent="true"
        android:background="@android:color/darker_gray"
        android:clickable="true"
        android:cropToPadding="true"
        android:padding="8dp"
        android:scaleType="fitXY"
        android:src="@drawable/test"
        lib:grayscale="false"
        lib:mask="@drawable/ic_launcher"
        lib:roundedRadius="15dp" />

</RelativeLayout>
```

3、不依赖 R.java 的方案

1） 使用资源名称。Resource 类提供了 int getIdentifier (String name, String defType, String defPackage) 动态获取资源 ID。缺点是，由于是运行时获取，出现错误不易发现；且命名难于维护。

2） 使用 publics.xml 约定资源 ID。优点是，可在编译时优化，源码、资源名称都可以混淆。缺陷是，当双方资源 ID 可能造成冲突。

四、其他

1、资源 ID 格式

资源 ID 格式是32位整型格式 PPTTNNNN，高位段 PP 是包名编码，中位段 TT 是资源类型编码，低位段 NNNN 是资源名编码。通常，Android 平台公共资源的 PP 总是 0x01，应用私有资源的 PP 总是 0x7f。 [3]

2、资源混淆

resources.arsc 文件中主要记录了资源的名称、资源文件的路径 、字符串资源的字符串值 、资源的配置等信息。通过短命名资源文件，可以同步的减少 resources.arsc 文件中记录该资源的路径信息字符数，从而减小资源的大小，并给反编译制造困难。 [4]

3、TODO: 尝试使用 aapt 生成 0x02TTNNNN 的资源 ID，类似 android.jar SDK 框架提供 publics.xml，解决 R.java 依赖问题，资源冲突问题与资源混淆问题。

引用

[1] Android 应用程序资源的查找过程分析, http://blog.csdn.net/luoshengyang/article/details/8806798

[2] Android UI 指南，主题与样式, http://developer.android.com/guide/topics/ui/themes.html

[3] Android 资源 id 格式, http://stackoverflow.com/questions/6517151/how-does-the-mapping-between-android-resources-and-resources-id-work/6646113#6646113

[4] Android 手机 QQ 资源混淆方案介绍, http://km.oa.com/group/22112/articles/show/185206?kmref=search