修改记录

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 日期 | 作者 | 说明 |
| V0.01 | 10年9月7日 | 徐申龙 | 初始版本。 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

[Android 资源分布 1](#_Toc278967067)

[aapt工具使用 1](#_Toc278967068)

[更换 apk 包中的图片文件 2](#_Toc278967069)

XML简介

在 Android 应用开发中，XML 文件在 Android 开发中起了很重要的作用，比如应用程序信息、权限管理 ，程序中字符串、图片、数值、Layout、动画，以及一些程序的默认配置信息等都已 XML 为载体。所以我们要经常与 XML 文件打交道，由于涉及面很广，内容也很丰富，想真正掌握好 XML 要费一番功夫的，而且缺乏文档、书籍对此介绍不深入和全面。本文试图提供详细的信息，使我们更深入了解 Android 中 XML 的处理，告别依样画葫芦的尴尬境地。

Java 对 XML 的支持比较充分，Java API for XML Processing (JAXP) 就提供了两种方法来处理 XML：文档对象模型（DOM）方法是用标准的对象模型表示 XML 文档；Simple API for XML (SAX) 方法使用应用程序提供的事件处理程序来处理 XML。JSR-173 提出了一种面向流的新方法：Streaming API for XML (StAX)。

DOM 解析器将所有 XML 文档读取到内存中，解析成树状模型，然后允许您使用 DOM API 遍历 XML 树、检索所需的数据。这种方法结构清晰、操作方便，但是耗费内存。而SAX 是便读入文档便解析，不需要把整个文档读入内存再解析，而且应用程序只在读取时检查数据，不需要将数据存储在内存中。相比，SAX内存和CPU消耗低，解析速度快，适于解析大型文档，资源有限的嵌入式系统， 但是SAX可能需要更多的编码工作。当然，DOM也有其优点，他更适合随时修改数据并输出XML 文件，而SAX一般适合读取数据，而不适合修改数据，而且这些数据必须在解析时就要取出来，解析完后就不能读取了，除非重新解析一遍。

StAX 是面向的流的方式，但是其实跟 SAX 有类似的地方，就是解析器和程序之间都是通过事件交互的，只是SAX采用了“推模式”（push modal），而StAX则使用的是“拉模式”（pull model）。SAX是自己按顺序把整个文档解析完，在它解析的过程，会主动产生事件交给程序中相应的处理函数来处理当前内容；而StAX是由程序来驱动整个解析过程，只有当程序发出解析请求的时候，解析器才会告诉相应的事件，然后程序再决定如何处理当前内容。

三类解析器简单比较

|  |  |  |  |
| --- | --- | --- | --- |
| 技术 | 有利 | 局限 | 适用于 |
| DOM 解析器 | 1.易于上手  2.丰富的API，易于访问  3.整棵树被载入内存，能随机访问节点 | 1.整个文档必须一次解析完  2.载入文档树到内存代价昂贵  3.不利于实现对象类型绑定，需要给所有节点创建单独的类型 | 需要修改xml或者用来处理XSLT（不要用在对XML只有读操作的程序中） |
| SAX 解析器 | 1.没有将整个文档读入内存，内存耗费较低  2.“推模式”允许注册多种内容处理器 | 1.没有内建的文档导航支持  2.不能随机访问XML文档  3.不支持命名空间 | 对XML只有读操作的程序（不要用来操作和修改XML文档） |
| StAX 解析器 | 1.有针对简单和性能的两种解析模式  2.由程序控制解析器，易于支持多输入（easily supporting multiple inputs）  3.强大的过滤功能有利于数据检索 | 1.没有内建的文档导航支持  2.不能随机访问XML文档  3.不支持修改XML文档 | 需要对XML文档进行流处理而且支持命名空间的程序（不要用来操作和修改XML文档） |

通过比较发现StAX比较适合解析Android 应用程序XML 资源。

XML 基本语法

XML 文件就是一般的文本文件，但这是有结构、有组织的文件。这样既便于人的阅读，又利于程序的处理。

声明

属于指示处理的一部分，版本、编码以及其他内容等，比如

<? xml version="1.0" encoding="GB2312" standalone="no" ?>

tag （标签）

也称作element、node， 是XML的基本单元。标签要成对出现 ，标签可以嵌套。属性名和属性值是成对的，一个标记下可以有多个属性，属性值必须用引号标明。其基本结构如下：

<标记名 属性名=”属性值”>

文字内容

</标记名>

注释：

<!--注释 -->

注释不能放在标记之中（即”<”、”>”之间不能放注释），不能被嵌套，注释内容不能有 字符“-”或字符串“--”。

CDATA

<![CDATA[数据]]>

CDATA中的数据不被XML解析器解析，而是原封不懂的传回给程序。 比如在标签中大于号要用”&gt”表示，在CDATA中直接写大于号即可。

特殊字符及大小写

&quot; 表示引号（”）

&amp; 表示&

&lt; 表示 <

&gt; 表示>

XML 对大小写和白空间敏感。

下面的内容就是一个完整的 XML 文件：

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

<manifest package="com.android.speechrecorder"

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

<application android:label="@string/app\_name">

<activity android:name="SpeechRecorderActivity"

android:label="@string/speechrecorderlabel">

<intent-filter>

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

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

</intent-filter>

</activity>

</application>

</manifest>

Expat 分析

expat是用C写的XML解析器 ，采用流的方式来解析XML文件，并且基于事件通知型来调用分析到的数据。由于 expat库是由XML的主要负责人James Clark来实现的，因此它是符合W3C的XML标准的。 Android中 代码的位置在external/expat/lib ， 模块名是libexpat。

external/expat/lib/expat.h 定义了结构体 XML\_Parser ( 它实际上是结构 XML\_ParserStruct 的别名) ，

external/expat/lib/xmlparse.c 是 XML 解析器的主要实现。

基本使用方法

expat 的 API 使用非常方便。

1. 创建解析器

调用 XML\_ParserCreate 或者 XML\_ParserCreateNS 创建一个 XML\_Parser 解析器。

2. 实现 Handler

Handler 就是回调函数。根据需要设置 Handler，

比如在解析器解析元素之前和之后，用户要做处理，就实现一个 XML\_StartElementHandler

及XML\_EndElementHandler。更多的函数原型见external/expat/lib/expat.h。

3. 设置 Handler 和 UserData

XML\_SetUserData ： 设置解析器的用户数据，这个数据一般是指向用户空间的结构体的指针，

解析器在 Handler 中传回此指针， 以便用户把解析出来的数据处理保存。

XML\_SetElementHandler ：设置元素的回调函数

XML\_SetNamespaceDeclHandler : 设置命名空间声明的回调函数

XML\_SetCommentHandler ：声明注释的回调函数

...... ：依此类推。

4. 解析 XML 文档

先调用 read 把要解析的 XML 文件读到一个 buffer 里，再把 buffer 里的数据送给

XML\_Parse 解析， 解析器在解析出数据后，如果用户设置了这部分的

Handler，它会调用这个 Handler，把数据传回给用户处理。这样循环往复，直至解析完文件或解析错误。

5. 释放解析器

当解析器不再使用时，调用 XML\_ParserFree 释放解析器。

Android 中 Expat 解析器包装

libcore/xml/src/main/native/org\_apache\_harmony\_xml\_ExpatParser.cpp 中函数 initialize 和 frameworks/base/tools/aapt/XMLNode.cpp 中函数 XMLNode::parse 均使用了 XML\_Parse 解析XML 文件，前者是 提供给 运行时代码用的，后者是编译 XML 资源文件（即aapt）时用的。

我们有时需要在处理XML文件后，要输出处理结果到文件，而且处理过程需要回溯，比如aapt的XML 编译。而Expat不是 DOM 模型，不会给我们生成XML节点树。所以，Android 作了一些封装，叫libutils ，它把Expat 解析出的数据保存在内存中。

frameworks/base/include/utils/ResourceTypes.h定义资源的数据结构，主要是ResTable， ResXMLParser 及其派生类 ResXMLTree。 frameworks/base/utils/ResourceTypes.cpp 是类的实现。

包含头文件ResourceTypes.h的源文件有：

frameworks/base/core/jni/android\_util\_XmlBlock.cpp 、 android\_util\_AssetManager.cpp

android\_util\_StringBlock.cpp 、 android/graphics/BitmapFactory.cpp

NinePatchImpl.cpp 、 Movie.cpp

frameworks/base/graphics/jni/android\_renderscript\_RenderScript.cpp

frameworks/base/tools/aapt/Images.cpp 、 printapk.cpp 、 ResourceTable.cpp

frameworks/base/utils/AssetManager.cpp 也包含了ResourceTypes.h，但他处理只读文件，所以只涉及android::ResTable\_config，暂不作讨论。

再查看 frameworks/base/libs/utils/Android.mk，看到 libutils 编译了两次，编译成主机静态库的文件给 aapt 使用，编译成共享库的文件是给 Dalvik 使用。而看过external/expat/ Android.mk后发现libexpat也是这样编译成两份不同的库。 实现代码共用，架构相当不错。

从中可以看到， aapt 使用 libutils 把 XML 等资源转变到 APK 文档中，Android 应用通过 JNI 也使用 libutils 解析 APK 文档 XML 资源。 这就是我们预测的结果。

aapt 的分析及使用

源码放在目录frameworks/base/tools/aapt。Main.cpp中的函数main是程序的主入口。

main -> doPackage |-> buildResources -> parsePackage -> parseXMLResource

|-> writeResourceSymbols

|-> writeSymbolClass

|-> writeLayoutClasses

|-> writeAPK

Command.cpp Resource.cpp XMLNode.cpp

XMLNode::parse 是静态成员函数，它是aapt调用expat的入口，所以这个函数很关键。搜索代码，发现 调用这个函数的地方有 ：

ResourceTable.cpp compileXmlFile，

Resource.cpp buildResource (只被 command.cpp doPackage 调用)

XMLNode.cpp parseXMLResource (在 Resource.cpp 里多有调用，如 parsePackage )

（均在frameworks/base/tools/aapt目录下）

可以发现了 ParseXMLResource的身影，这样就能把aapt和expat接上了。

生成R.java

~/workspace/EX07\_11$ aapt package -z -m -J ./gen -M AndroidManifest.xml -S ./res \

-I /home/leo/android/sdk/platforms/android-2.2/android.jar

aapt 用来生成了 R.java，也 AndroidManifest.java。在使用Eclipse时，有时会不生成R.java文件，此时使用这个命令修正。

打包

$ aapt package -S res -M AndroidManifest.xml -F xxt.apk -f

查看资源文件

$aapt list out/target/common/obj/APPS/framework-res-intermediates/package-export-apk

查看某个xml的内容

$ aapt dump xmltree xxx.apk res/layout/main.xml

（xml不能是values\*中的，因为它已被打进resources.arsc）

查看resources.arsc中的资源内容

$ aapt dump --values resources xxx.apk

从包中删除某文件

$ aapt remove xxx.apk res/layout/alert\_dialog.xml

将某文件加入包中

$ aapt add xxx.apk res/layout/alert\_dialog.xml

用appt与zip解压修改再压缩的方式不同，它会编译xml文件和处理9patch图。

XMLPullParser

Android 并未提供对 Java StAX API 的支持。但是，Android 确实附带了一个 pull 解析器，其工作方式类似于 StAX （迭代器基于事件 ，而不是指针）。它允许您的应用程序代码从解析器中获取事件，这与 SAX 解析器自动将事件推入处理程序相反。

Android 处理XML 的核心Java代码放在libcore/xml ，SAX 封装代码放在frameworks/base/sax 。

分析 Android frameworks 中 XMLPullParser 设计为一个接口， (libcore/xml/src/main/java/org/xmlpull/v1/XmlPullParser.java)

而接口 XmlResourceParser 是 XML 资源返回的 XML 解析接口， 是一个标准的 XmlPullParser 接口，同时扩展了 AttributeSet 接口，添加了 close() 方法，以便用户在读取完资源时关闭接口的资源。

frameworks/base/core/java/android/content/res/XmlBlock.java 的内部类 Parser 实现了 XmlResourceParser 接口。

frameworks/base/core/java/android/util/XmlPullAttributes.java 实现了 AttributeSet ，而 XmlPullAttributes 可以看作对 XmlPullParser 属性处理方面API 的封装。

一般使用 Xml.asAttributeSet 获取 AttributeSet，此函数判断参数 XmlPullParser 类型对象 parser 是否实现了 AttributeSet 接口，实现了就返回 parser，否则新建一个以 parser 为参数的 XmlPullAttributes 对象，也就是说 XmlPullAttributes 是 AttributeSet 接口的默认实现。

XmlResourceParser parser; // 或者 XmlPullParser parser;

...

AttributeSet attrs = Xml.asAttributeSet(parser);

XmlBlock 类是编译化 XML 文件的包装器，她有个对应的 JNI 文件，android\_util\_XmlBlock.cpp，他们主要工作就是转发给 ResXMLParser 处理 (libutils 库， ResourceTypes.cpp 中实现)。 另外，编译过的 XML 文件被打包到 .apk文件中，不能修改 ( 如果要支持可写，代价太大了啊！)， 所以其内部类 Parser 对一些写入的操作是不支持的。

使用 XmlPullParser 接口的类有：

frameworks/base/core/java/android/view

LayoutInflater.java inflate (rInflate -> parseRequestFocus 、parseInclude)

MenuInflater.java parseMenu (inflate)

PreferenceInflater.java onCreateCustomFromTag

animation/AnimationUtils.java

createAnimationFromXml

createLayoutAnimationFromXml (called by loadLayoutAnimation)

loadInterpolator (include private method createInterpolatorFromXml)

frameworks/base/graphics/java/android/graphics/drawable

Drawable.java 方法 createFromXmlInner android.graphics.drawable

包中类的方法 inflate 基本上都会调用此方法，如 AnimatedRotateDrawable、 AnimationDrawable、 LayerDrawable、 LevelListDrawable 等类的 inflate 方法。此方法 处理标签 selector，level-list，color，shape等。

frameworks/base/core/java/android/content/

res/Resources.java parseBundleExtra ，loadXmlResourceParser 调用可以获取

布局(getLayout)、动画(getAnimation)、普通 XML 文件 (getXml)等资源

pm/PakcageParser.java parsePackage 以及parseIntent、 parseApplication、 parseActivity、 parseService、 parseProvider、 parsePermission parsePermissionTree 、 parseAllMetaData 等，是对 AndroidManifest.xml 的解析。

IntentFilter.java readFromXml

Intent.java parseIntent

Android XML

Android XML 文件的骨架

1. AndroidManifest.xml

<manifest xmlns:android=”xxx” package=”xxx” android:shareUserId=”xxx”>

<permission android:name=”xxx” android:permissionGroup=”xxx” android:label=”xxx” .../>

<uses-permission android:name=”xxx”/>

<application android:name=”xxx” android:process=”xxx” android:label=”xxx” android:icon=”xxx”>

<activity android:name=”xxx” android:launchMode=”xxx” android:screenOrientation=”xxx” .../>

<intent-filter>

<action android:name=”xxx”/>

<category android:name=”xxx”/>

<data android:mimeType="\*/\*" />

</intent-filter>

<meta-data android:name="xxx" android:resource="@xxx" />

</activity>

<receiver>

<intent-filter>

<action android:name=”xxx”/>

</intent-filter>

</receiver>

<service android:name="xxx" android:exported="true" />

<provider android:name=”xxx” android:authorities=”xxx” android:writePermission=”xxx” ...>

<path-permission android:pathPrefix="xxx" android:readPermission="xxx" />

<provider/>

</application>

</manifest>

2. resource 类 标签

<resource>

<dimen name=”xxx”>xxx</dimen>

<drawable name=”xxx”>xxx</drawable>

<color name=”xxx”>xxx</color>

<string name=”xxx”>xxx</string>

<integer name=”xxx”>xxx</integer>

<string-array name=”xxx” translatable=”xxx”>

<item>xxx</item>

… …

</string-array>

<plurals name=”matches\_foudn”>

<item quantity=”zero”>”没有匹配项”</item>

… …

</plurals>

<style name=”xxx” parent=”xxx”>

<item name=”xxx”>xxx</item>

… …

</style>

<attr name=”xxx”>

<enum name=”xxx” value=”xxx”/>

… …

<enum name=”xxx” value=”xxx”/>

</attr>

<declare-style name=”xxx”>

<attr name=”xxx” format=”xxx”>

… …

</declare-style>

</resource>

3. layout.xml

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

android:orientation="vertical"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

<ImageView android:id="@+id/wallpaper"

android:layout\_width="match\_parent"

android:layout\_height="0dip"

android:layout\_weight="1.0"

android:scaleType="fitCenter" />

</LinearLayout>

类似LinearLayout 的layout 还有AbsoluteLayout 、FrameLayout、RelativeLayout， Widget 还有Button、EditText、CheckBox、ImageView等。

4. 其他

<selector xmlns:android=”http://schemas.android.com/apk/res/android”>

<item android:XXX=”XXX”/>

… …

</selector>

<level-list xmlns:android=”http://schemas.android.com/apk/res/android”>

<item android:maxLevel=”0” android:drawable=”@android:color/transparent”/>

… …

</level-list>

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

<include layout="@layout/all\_apps\_2d" />

</merge>

XML 编译成资源后与 R.java 的转换关系

按文件名和目录

aapt会根据res下一些目录生成类，把他们下面的文件编入相应的类中。

res/anim class anim

res/anim/scaler.xml (只含scaler标签) 类anim的域 int scaler = xxx;

res/ drawable class drawable

res/drawable/baby.png 类drawable的域 int baby = xxx;

这类目录还有anim，color，drawable，layout，menu， raw， xml。

res/xml目录下的文件， aapt不会转换，而是压缩直接放入.apk文件，这里面的内容由应用程序自己处理，Android framework 不会处理的。

res/color目录下的XML 文件一般是selector 标签。selector 是表示一个View 的各种状态的属性集合。

按标签

这类资源文件一般放在 value/ 以及values-xx/， 放在标签容器resources之中 。XML标记名作为R.java中的类名，属性name的值作为类域名。这类标签常见有array、 attr、 id、 dimen、 drawable、 integer、 string。

declare-styleable标记

声明 styleable，进入 sytleable 类，name属性值生成一个数组域，数组元素个数等于旗下 attr 标记个数。 而旗下的attr 标记进入 attr类，以其name属性值作为域名。同时，把declare-styleable标记 name属性值和各个attr标记的name属性值以”\_”链接作为在 sytleable 类的域。

<declare-styleable name="Favorite">

<attr name="className" format="string" />

<attr name="packageName" format="string" />

</declare-styleable>

编译后生成的R.java代码是：

public static final class styleable {

public static final int[] Favorite = {

0x7f010000, 0x7f010001};

public static final int Favorite\_className = 0;

public static final int Favorite\_packageName = 1;

}

public static final class attr {

public static final int className=0x7f010000;

public static final int packageName=0x7f010001;

}

这里面有个对应关系：

R.attr.className = = R.Favorite[R.Favorite\_className]

由属性值生成的类

在res/layout下的xml文件，作为一个整体进入layout类。但当应用程序要对其中的Widget中作处理时，加上属性” android:id="@+id/wallpaper" ” ， 在代码中就可以用 R.id.wallpaper 引用到这个widget。aapt在处理属性值以”@+”开始时，会生成类和域，其中以“/”划分。

其他

string-array 标签会进入array 类。AndroidManifest.xml 文件中若声明了 permission，则会生成Manifest.java文件和 permission 类。

属性值以”?”开始表示的是主题(theme)，setTheme;属性值以”@”开始表示对别的资源的引用;属性值以”@+”，指示aapt生成类和域，格式为”@+<class>/<field>“，<class>、<field>可以是字母、数字和下划线，但不能以数字开始。如 ：

<AbsoluteLayout

android:id="@+id/mywidget"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

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

>

<TextView

android:id="@+id/myTextView"

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

android:text="@string/hello"

android:textColor="?android:attr/textColorPrimary"

/>

</AbsoluteLayout>

“$”表示内部类。下面的内容摘录自development/samples/ApiDemos/AndroidManifest.xml， Binding是 LocalServiceActivities内部类，也就是app/ LocalServiceActivities.java里实现的类：

<activity android:name=".app.LocalServiceActivities$Binding"

android:label="@string/activity\_local\_service\_binding">

<intent-filter>

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

<category android:name="android.intent.category.SAMPLE\_CODE" />

</intent-filter>

</activity>

命名空间的问题

属性名以 xmlns开头的是XML 命名空间的声明，属性名“:”后面的就是命名空间名。一旦声明了命名空间，这个标记的子标记的属性名都要以此命名空间名开始，并以“:”与真正的属性名链接。

error: Error parsing XML: unbound prefix

这是因为命名空间的问题，因为使用了属性名”android:id”，给resources 标签加上属性

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

或者

<drawable name="blue" android:id="@+drawref/refff" xmlns:android="http://schemas.android.com/apk/res/android">#0000FF</drawable>

不过不会在R.java中生成 drawref 类， “@+”只有在Android空间有效。

文件路径和属性值的“/”到Java 代码以”.”替代了，属性名中的”:”在 Java 中以“\_”替代了

XML Java

com/android/foo.java import com.android.foo;

res/layout/main.xml R.layout.main;

“@android:color/transparent” android.R.color.transparent

android:name android\_name

Open quote is expected

原因全角引号造成的。

XML使用举例

1.利用XML 配置程序的默认设置

使用declare-styleable 声明属性

res/values/attr.xml 内容如下：

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

<resources>

<declare-styleable name="Favorite">

<attr name="className" format="string" />

<attr name="packageName" format="string" />

<attr name="screen" format="string" />

<attr name="x" format="string" />

<attr name="y" format="string" />

<attr name="spanX" format="string" />

<attr name="spanY" format="string" />

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

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

<attr name="uri" format="string" />

</declare-styleable>

</resources>

res/xml/default\_workspace.xml 内容如下：

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

<favorites xmlns:launcher="http://schemas.android.com/apk/res/com.android.launcher">

<!-- Far-left screen [0] -->

<!-- Left screen [1] -->

<appwidget

launcher:packageName="com.google.android.apps.genie.geniewidget"

launcher:className="com.google.android.apps.genie.geniewidget.miniwidget.MiniWidgetProvider"

launcher:screen="1"

launcher:x="0"

launcher:y="0"

launcher:spanX="4"

launcher:spanY="1" />

<!-- Middle screen [2] -->

<search

launcher:screen="2"

launcher:x="0"

launcher:y="0" />

<!-- Right screen [3] -->

<appwidget

launcher:packageName="com.android.music"

launcher:className="com.android.music.MediaAppWidgetProvider"

launcher:screen="3"

launcher:x="0"

launcher:y="0"

launcher:spanX="4"

launcher:spanY="1" />

</favorites>

代码片断：

private static final String TAG\_FAVORITES = "favorites";

private static final String TAG\_FAVORITE = "favorite";

private static final String TAG\_CLOCK = "clock";

private static final String TAG\_SEARCH = "search";

private static final String TAG\_APPWIDGET = "appwidget";

private static final String TAG\_SHORTCUT = "shortcut";

… …

try {

XmlResourceParser parser = mContext.getResources().getXml(R.xml.default\_workspace);

AttributeSet attrs = Xml.asAttributeSet(parser);

XmlUtils.beginDocument(parser, TAG\_FAVORITES);

final int depth = parser.getDepth();

int type;

while (((type = parser.next()) != XmlPullParser.END\_TAG ||

parser.getDepth() > depth) && type != XmlPullParser.END\_DOCUMENT) {

if (type != XmlPullParser.START\_TAG) {

continue;

}

final String name = parser.getName();

TypedArray a = mContext.obtainStyledAttributes(attrs, R.styleable.Favorite);

LOG.D(“TEST”, name + “, ” +

a.getString(R.styleable.Favorite\_y) + “,” +

a.getString(R.styleable.Favorite\_screen));

}

} catch (XmlPullParserException e) {

Log.w(TAG, "Got exception parsing favorites.", e);

} catch (IOException e) {

Log.w(TAG, "Got exception parsing favorites.", e);

}

2. XML 动画

资源文件res/anim/scaler.xml 内容如下：

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

android:fromXScale="1.0"

android:toXScale="5.0"

android:fromYScale="1.0"

android:toYScale="5.0"

android:pivotX="50%"

android:pivotY="50%"

android:startOffset="1000"

android:duration="3000"

android:fillBefore="false">

</scale>

代码片断：

public ... foo(View v, ...)

{

...

v.startAnimation(AnimationUtils.loadAnimation(v.getContext(), R.anim.scaler));

...

}

数据解析过程：

AnimationUtils.loadAnimation 调用 AnimationUtils.createAnimationFromXml, Xml.asAttributeSet 获取一个 AttributeSet 接口以便获取属性，在解析程序包资源 R.anim.scaler 发现标签 scale 时创建 ScaleAnimation 对象。

ScaleAnimation 是 Animation 的派生类，所以其构造方法先调用 Animation 的构造方法。Animation 的构造方法以 com.android.internal.R.styleable.Animation 调用 Context.obtainStyledAttributes

解析动画的共有属性，并把这些属性值设置到对象的域。

ScaleAnimation 的构造方法再以参数 com.android.internal.R.styleable.ScaleAnimation 调用

Context.obtainStyledAttributes 方法自身的一些属性，然后设置到对象中。

Context.obtainStyledAttributes 调用了 Resources.Theme.obtainStyledAttributes，后者又调用了AssetManager.applyStyle，这是个 native 方法，对应的 JNI 函数是 android\_content\_AssetManager\_applyThemeStyle

(frameworks/base/core/jni/android\_util\_AssetManager.cpp)

ResTable::Theme::applyStyle (frameworks/base/libs/utils/ResourceTypes.cpp)

Resources.Theme.obtainStyledAttributes 注释中说了，她只能解析编译过的 XML 文件，即由 aapt 编译后放到 .apk 文档中的 XML 文件。 要解析一般的 XML 文件，我们可以使用 SAX 。

参考资料

1 Michael Galpin, 在 Android 上使用 XML

2 Baidu Blog， XML解析技术一览