Android应用程序资源管理框架

罗升阳

http://weibo.com/shengyangluo

http://blog.csdn.net/luoshengyang

About Me

- 《老罗的Android之旅》博客作者
- 《Android系统源代码情景分析》书籍作者
- 博客：http://blog.csdn.net/Luoshengyang
- 微博：http://weibo.com/shengyangluo

Agenda

- Android资源框架概述
- Android资源编译过程
- Android资源查找过程

Android资源框架概述

- 背景
  - Android设备屏幕大小和密度各不相同
  - Android设备运行在不同国家、地区和语言上
- 问题
  - 同一个应用程序需要支持不同的语言和UI布局
- 方案
  - 代码和资源分离

Android资源框架概述

- 流行的资源管理框架
  - Web开发：CSS文件
  - MFC开发：RC文件
  - WPF开发：XAML文件
  - QT开发：QML文件
  - COCOA开发：XIB文件
- Android设备的多样性导致其资源组织和管理更复杂

Android资源框架概述

- 资源分类
  - assets
  - res
    - animator
    - anim
    - color
    - drawable
    - layout
    - menu
    - raw
    - values
    - xml

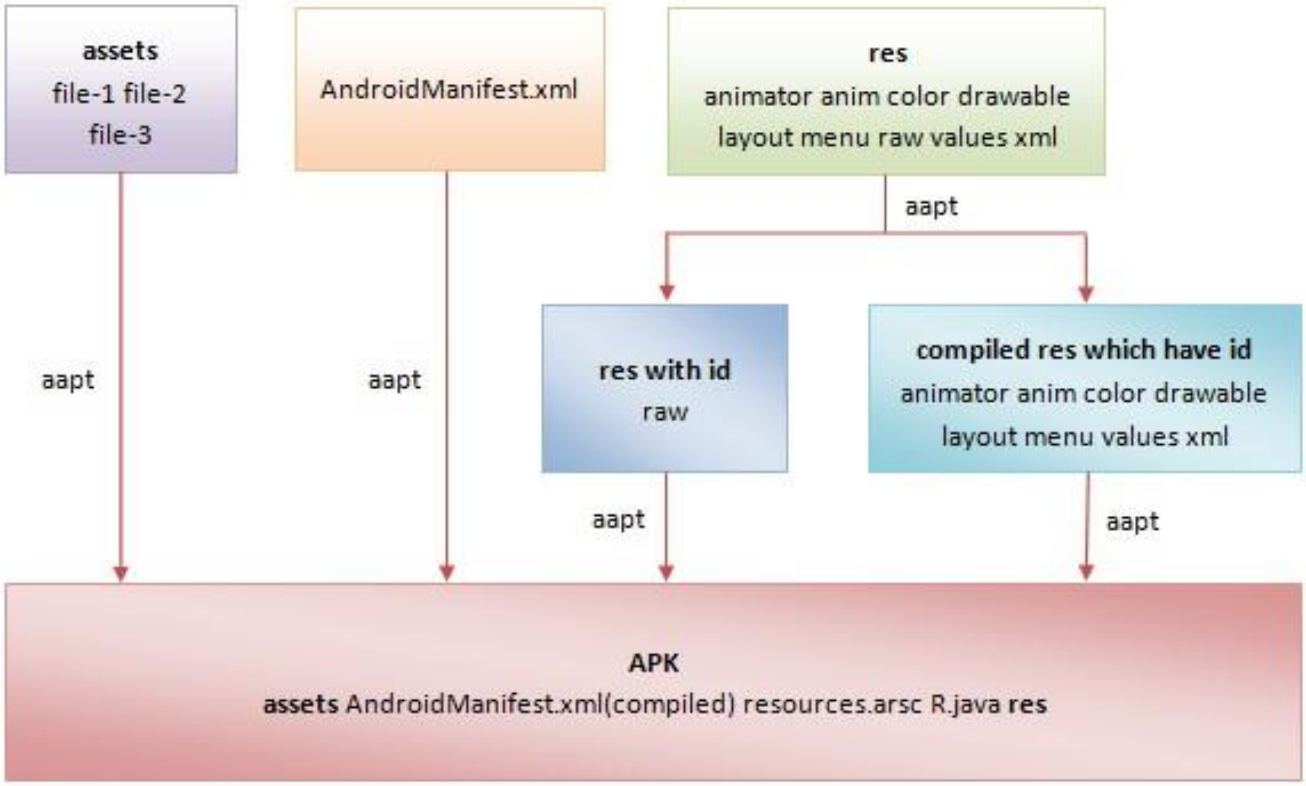
Android资源框架概述

- 资源目录组织
  - *<resources_name>-<config_qualifier>*

Configuration	Qualifier Values
MCC and MNC	Examples: mcc310,mcc310-mnc004,mcc208-mnc00, etc.
Language and region	Examples: en,fr,en-rUS,fr-rFR,fr-rCA,etc.
Layout Direction	ldrtl,ldltr
smallestWidth	sw<N>dp, Examples: sw320dp,sw600dp,sw720dp,etc.
Available width	w<N>dp, Examples: w720dp,w1024dp,etc.
Available height	h<N>dp, Examples: h720dp,h1024dp,etc.
Screen size	small,normal,large,xlarge
Screen aspect	long,notlong
Screen orientation	port,land
UI mode	car,desk,television,appliance
Night mode	night,notnight
Screen pixel density (dpi)	ldpi,mdpi,hdpi,xhdpi,nodpi,tvdpi
Touchscreen type	notouch,finger
Keyboard availability	keysexposed,keyshidden,keyssoft
Primary text input method	nokeys,qwerty,12key
Navigation key availability	navexposed,navhidden
Primary non-touch navigation method	nonav,dpad,trackball,wheel
Platform Version (API level)	Examples: v3,v4,v7,etc.

Android资源编译过程

- Android资源编译框架



Android资源编译过程

- 编译过程主要是将XML文件从文本格式编译为二进制格式
  - 二进制格式的XML文件占用空间更小。这是由于所有XML元素的标签、属性名称、属性值和内容所涉及到的字符串都会被统一收集到一个字符串资源池中去，并且会去重。有了这个字符串资源池，原来使用字符串的地方就会被替换成一个索引到字符串资源池的整数值，从而可以减少文件的大小。
  - 二进制格式的XML文件解析速度更快。这是由于二进制格式的XML元素里面不再包含有字符串值，因此就避免了进行字符串解析，从而提高速度。

Android资源编译过程

- 为了支持运行时快速定位最匹配资源，编译过程还会有为资源生成ID，以及生成资源索引表
  - 赋予每一个非assets资源一个ID值，这些ID值以常量的形式定义在一个R.java文件中
  - 生成一个resources.arsc文件，用来描述那些具有ID值的资源的配置信息，它的内容就相当于是一个资源索引表

Android资源编译过程

- 资源ID是一个4字节的无符号整数，其中，最高字节表示Package ID，次高字节表示Type ID，最低两字节表示Entry ID
- Package ID相当于是一个命名空间，限定资源的来源。Android系统当前定义了两个资源命令空间，其中一个系统资源命令空间，它的Package ID等于0x01，另外一个是应用程序资源命令空间，它的Package ID等于0x7f。所有位于[0x01, 0x7f]之间的Package ID都是合法的，而在这个范围之外的都是非法的Package ID
- Type ID是指资源的类型ID。资源的类型有animator、anim、color、drawable、layout、menu、raw、string和xml等等若干种，每一种都会被赋予一个ID
- Entry ID是指每一个资源在其所属的资源类型中所出现的次序。注意，不同类型的资源的Entry ID有可能是相同的，但是由于它们的类型不同，我们仍然可以通过其资源ID来区别开来

Android资源编译过程

- 例子

```
project
  --AndroidManifest.xml
  --res
    --drawable-ldpi
      --icon.png
    --drawable-mdpi
      --icon.png
    --drawable-hdpi
      --icon.png
    --layout
      --main.xml
      --sub.xml
    --values
      --strings.xml
```

Android资源编译过程

- Step 1: 解析AndroidManifest.xml
  - 为了获得要编译资源的应用程序的包名称。
  - 在AndroidManifest.xml文件中，manifest标签的package属性的值描述的就是应用程序的包名称。
  - 有了这个包名称之后，就可以创建一个资源表(Resource Table)

Android资源编译过程

- Step 2:添加被引用资源包
  - android:orientation=“vertical”中的vertical实际上是由系统定义的一个资源。
  - 在AOSP中，系统资源经过编译后，位于out/target/common/obj/APPS/framework-res_intermediates/package-export.apk文件中。
  - 因此，在AOSP中编译的应用程序资源，都会引用到系统资源包package-export.apk

Android资源编译过程

- Step 3: 收集资源文件
  - 在我们的例子中，包含三种类型的资源drawable、layout和values
  - 名称相同的资源归为一组。例如，名称为icon.png的资源项res/drawable-ldpi/icon.png、res/drawable-mdpi/icon.png和res/drawable-hdpi/icon.png归为一组。它们通过屏幕密度(ldpi、mdpi和hdpi)来区分

Android资源编译过程

- Step 4:将收集到的非values资源增加到资源表

Entry Name	Item Value	Config Description
icon.png	res/drable-ldpi/icon.png	ldpi
Icon.png	res/drable-mdpi/icon.png	mdpi
Icon.png	res/drable-hdpi/icon.png	hdpi
main.xml	res/layout/main.xml	default
sub.xml	res/layout/sub.xml	default

Android资源编译过程

- Step 5: 编译values类资源
  - strings.xml文件的内容

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Activity</string>
    <string name="sub_activity">Sub Activity</string>
    <string name="start_in_process">Start sub-activity in process</string>
    <string name="start_in_new_process">Start sub-activity in new process</string>
    <string name="finish">Finish activity</string>
</resources>
```

  - strings.xml文件编译后得到的资源项

Entry Name	Item Value	Config Description
app_name	Activity	default
sub_activity	Sub Activity	default
start_in_process	Start sub-activity in process	default
start_in_new_process	Start sub-activity in new process	default
finish	Finish activity	default

Android资源编译过程

- Step 6: 给自定义资源分配资源ID
  - 假设在res/values目录下有一个attrs.xml文件

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <attr name="custom_orientation">
        <enum name="custom_vertical" value="0" />
        <enum name="custom_horizontal" value="1" />
    </attr>
</resources>
```

  - attrs.xml文件被解析后

Entry Name	Item Value	Config Description
custom_orientation	-	default
custom_vertical	0	default
custom_horizontal	1	default

Android资源编译过程

- Step 7: 编译Xml资源文件
  - 以res/layout/main.xml为例

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center">
    <Button
        android:id="@+id/button_start_in_process"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:text="@string/start_in_process" >
    </Button>
    <Button
        android:id="@+id/button_start_in_new_process"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:text="@string/start_in_new_process" >
    </Button>
</LinearLayout>
```

Android资源编译过程

- Step 7.1: 解析xml文件
  - 解析Xml文件是为了可以在内存中用一系列树形结构的XMLNode来表示
- Step 7.2:赋予属性名称资源ID
  - 对于main.xml的根节点LinearLayout来说，就是要将它的属性名称android:orientation、android:layout_width、android:layout_height和android:gravity转换为资源ID

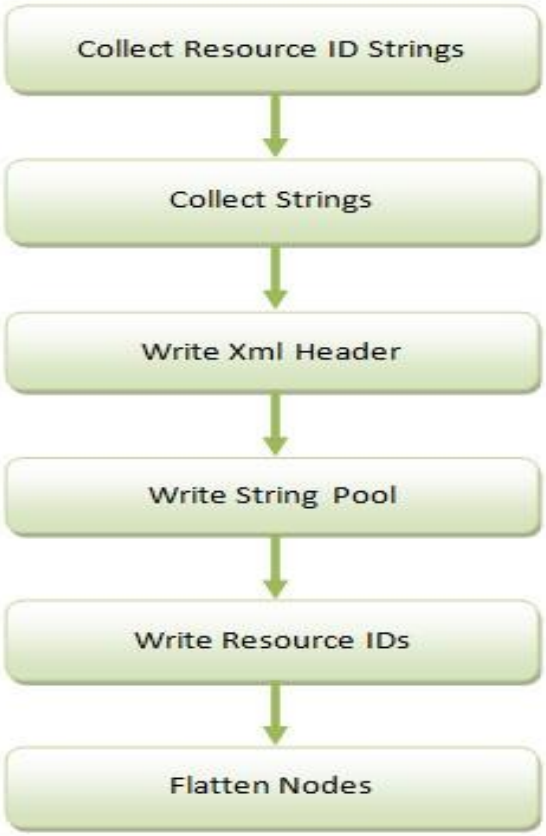
Android资源编译过程

- Step 7.3: 解析属性值
  - 例如，对于main.xml的属性android:orientation来说，它的合法取值为horizontal或者vertical，这里需要进行验证，并且将它们转换为ID值
  - 有些属性值是属于引用类型的，例如main.xml文件的两个Button节点的android:id属性值”@+id/button_start_in_process”和”@+id/button_start_in_new_process”，将会生成新的资源项

Entry Name	Item Value	Config Description
button_start_in_process	-	default
button_start_in_new_process	-	default

Android资源编译过程

- Step 7.4：将xml文件从文本格式转换为二进制格式



Android资源编译过程

- Step 7.4.1: 收集有资源ID的属性的名称字符串
  - 将收集到的字符串及其资源ID分别保存一个字符串池以及资源数组中
  - 对于main.xml文件来说，具有资源ID的Xml元素属性的名称字符串
    有”orientation”、”layout_width”、”layout_height”、”gravity”、”id”和”text”，假设它们的资源ID分别为0x010100c4、0x010100f4、0x010100f5、0x010100af、0x010100d0和0x0101014f:

	0	1	2	3	4	5
String Pool	orientation	layout_width	layout_height	gravity	id	text
Resource ID Array	0x010100c4	0x010100f4	0x010100f5	0x010100af	0x010100d0	0x0101014f

Android资源编译过程

- Step 7.4.2: 收集其它字符串
  - 这一步收集的字符串是不具有资源ID的
  - 对于main.xml文件来说，这一步收集到的字符串如下所示：

	.....	6	7	8	9
String Pool	.....	android	http://schemas.android.com/apk/res/android	LinearLayout	Button

Android资源编译过程

- Step 7.4.3:写入Xml文件头

```
struct ResChunk_header
{
    // Type identifier for this chunk.  The meaning of this value depends
    // on the containing chunk.
    uint16_t type;

    // Size of the chunk header (in bytes).  Adding this value to
    // the address of the chunk allows you to find its associated data
    // (if any).
    uint16_t headerSize;

    // Total size of this chunk (in bytes).  This is the chunkSize plus
    // the size of any data associated with the chunk.  Adding this value
    // to the chunk allows you to completely skip its contents (including
    // any child chunks).  If this value is the same as chunkSize, there is
    // no data associated with the chunk.
    uint32_t size;
};

struct ResXMLTree_header
{
    struct ResChunk_header header;
};
```

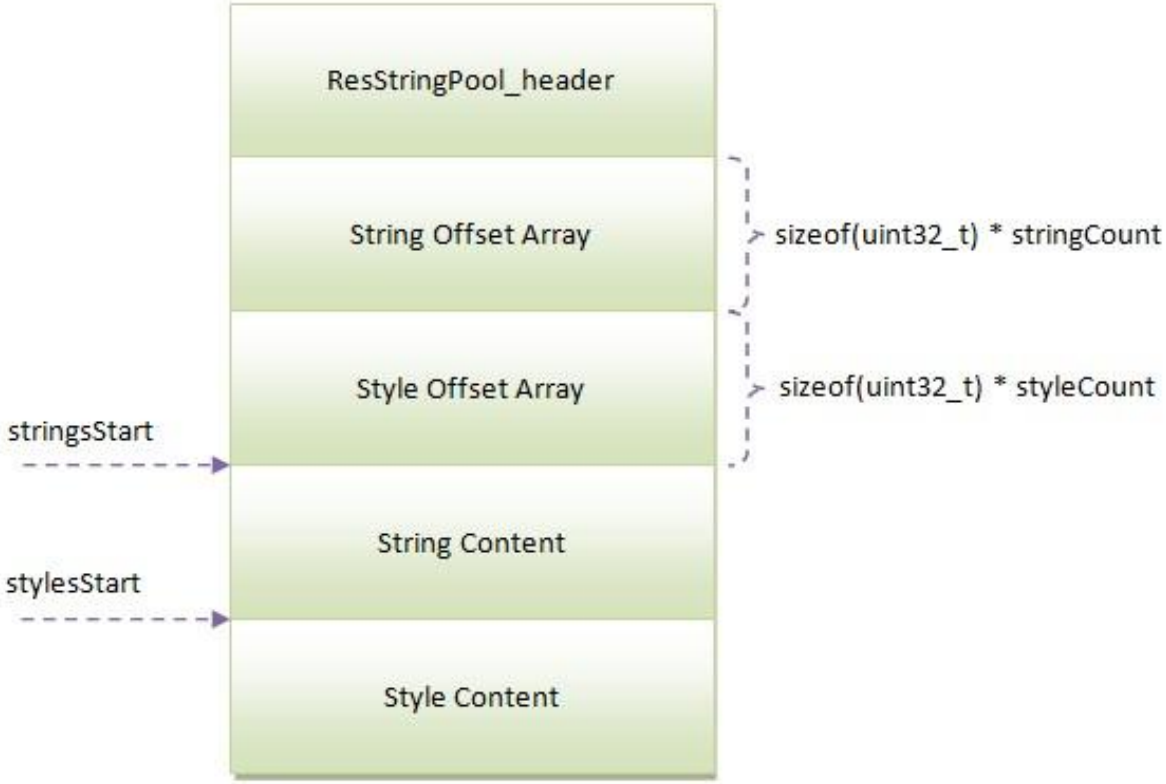
Android资源编译过程

- Step 7.4.4:写入字符串资源池
  - 对于main.xml来说，依次写入的字符串为”orientation”、”layout_width”、”layout_height”、”gravity”、”id”、"text"、"android"、”http://schemas.android.com/apk/res/android”、”LinearLayout”和”Button”
  - 写入的字符串池同样有一个头部

```
struct ResStringPool_header
{
    struct ResChunk_header header;

    // Number of strings in this pool (number of uint32_t indices that follow
    // in the data).
    uint32_t stringCount;

    // Number of style span arrays in the pool (number of uint32_t indices
    // follow the string indices).
    uint32_t styleCount;

    // Flags.
    enum {
        // If set, the string index is sorted by the string values (based
        // on strcmp16()).
        SORTED_FLAG = 1<<0,

        // String pool is encoded in UTF-8
        UTF8_FLAG = 1<<8
    };
    uint32_t flags;

    // Index from header of the string data.
    uint32_t stringsStart;

    // Index from header of the style data.
    uint32_t stylesStart;
};
```

Android资源编译过程

- Step 7.4.4:字符串池的结构



Android资源编译过程

- Step 7.4.4：样式字符串
  - 假设有一个字符串”mango”，实际上包含三个字符串”mango”、”b”和”I”，其中”mango”来有两个sytle，第一个style表示第1到第3个字符是粗体的，第二个style表示第4到第5个字符是斜体的
  - 样式字符串由ResStringPool_span和ResStringPool_ref两个结构描述

```
struct ResStringPool_ref
{
    // Index into the string pool table (uint32_t-offset from the indices
    // immediately after ResStringPool_header) at which to find the location
    // of the string data in the pool.
    uint32_t index;
};

struct ResStringPool_span
{
    enum {
        END = 0xFFFFFFFF
    };

    // This is the name of the span -- that is, the name of the XML
    // tag that defined it.  The special value END (0xFFFFFFFF) indicates
    // the end of an array of spans.
    ResStringPool_ref name;

    // The range of characters in the string that this span applies to.
    uint32_t firstChar, lastChar;
};
```

Android资源编译过程

- Step 7.4.5:写入资源ID
  - 资源ID数据块位于字符串池后面，它的头部使用ResChunk_header来描述
  - 以main.xml为例，字符串资源池的第一个字符串为”orientation”，而在资源ID数据块记录的第一个数据为0x010100c4，那么就表示属性名称字符串”orientation”对应的资源ID为0x010100c4

Android资源编译过程

- Step 7.4.6:压平Xml文件
  - 压平Xml文件其实就是指将里面的各个Xml元素中的字符串都替换掉。这些字符串要么是被替换成到字符串资源池的一个索引，要么是替换成一个具有类型的其它值。

Android资源编译过程

- Step 7.4.6.1:首先被压平的是一个表示命名空间的Xml Node
  - 这个Xml Node用两个ResXMLTree_node和两个ResXMLTree_namespaceExt来表示：

Android资源编译过程

- Step 7.4.6.1： ResXMLTree_node和ResXMLTree_namespaceExt的定义

```
struct ResXMLTree_node
{
    struct ResChunk_header header;

    // Line number in original source file at which this element appeared.
    uint32_t lineNumber;

    // Optional XML comment that was associated with this element; -1 if none.
    struct ResStringPool_ref comment;
};

/**
 * Extended XML tree node for namespace start/end nodes.
 * Appears header.headerSize bytes after a ResXMLTree_node.
 */
struct ResXMLTree_namespaceExt
{
    // The prefix of the namespace.
    struct ResStringPool_ref prefix;

    // The URI of the namespace.
    struct ResStringPool_ref uri;
};
```

Android资源编译过程

- Step 7.4.6.2:接下来被压平的是标签为LinearLayout的Xml Node
  - 这个Xml Node由两个ResXMLTree_node、一个ResXMLTree_attrExt、一个ResXMLTree_endElementExt和四个ResXMLTree_attribute来表示

Android资源编译过程

- Step 7.4.6.2: ResXMLTree_attrExt的定义

```
struct ResXMLTree_attrExt
{
    // String of the full namespace of this element.
    struct ResStringPool_ref ns;

    // String name of this node if it is an ELEMENT; the raw
    // character data if this is a CDATA node.
    struct ResStringPool_ref name;

    // Byte offset from the start of this structure where the attributes start.
    uint16_t attributeStart;

    // Size of the ResXMLTree_attribute structures that follow.
    uint16_t attributeSize;

    // Number of attributes associated with an ELEMENT.  These are
    // available as an array of ResXMLTree_attribute structures
    // immediately following this node.
    uint16_t attributeCount;

    // Index (1-based) of the "id" attribute. 0 if none.
    uint16_t idIndex;

    // Index (1-based) of the "class" attribute. 0 if none.
    uint16_t classIndex;

    // Index (1-based) of the "style" attribute. 0 if none.
    uint16_t styleIndex;
};
```

Android资源编译过程

- Step 7.4.6.2: ResXMLTree_attrExt的定义

```
struct ResXMLTree_attribute
{
    // Namespace of this attribute.
    struct ResStringPool_ref ns;

    // Name of this attribute.
    struct ResStringPool_ref name;

    // The original raw string value of this attribute.
    struct ResStringPool_ref rawValue;

    // Processesd typed value of this attribute.
    struct Res_value typedValue;
};
```

Android资源编译过程

- Step 7.4.6.2: Res_value的定义

```
struct Res_value
{
    // Number of bytes in this structure.
    uint16_t size;

    // Always set to 0.
    uint8_t res0;

    // Type of the data value.
    enum {
        // Contains no data.
        TYPE_NULL = 0x00,
        // The 'data' holds a ResTable_ref, a reference to another resource
        // table entry.
        TYPE_REFERENCE = 0x01,
        ......

        // ...end of integer flavors.
        TYPE_LAST_INT = 0x1f
    };
    uint8_t dataType;

    ......

    // The data for this item, as interpreted according to dataType.
    uint32_t data;

    ......
};
```

Android资源编译过程

- Step 7.4.6.2: ResXMLTree_endElementExt的定义

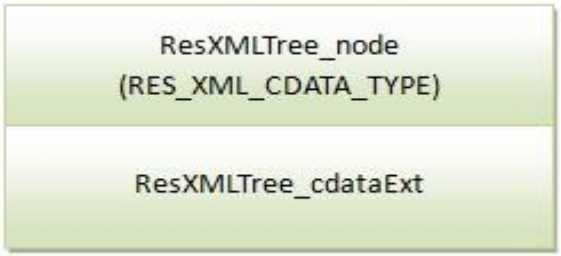
```
/**
 * Extended XML tree node for element start/end nodes.
 * Appears header.headerSize bytes after a ResXMLTree_node.
 */
struct ResXMLTree_endElementExt
{
    // String of the full namespace of this element.
    struct ResStringPool_ref ns;

    // String name of this node if it is an ELEMENT; the raw
    // character data if this is a CDATA node.
    struct ResStringPool_ref name;
};
```

Android资源编译过程

- Step 7.4.6.3:对于一个Xml文件来说，它除了有命名空间和普通标签类型的Node之外，还有一些称为CDATA类型的Node
  - 假设一个Xml文件，它的一个Item标签的内容如下所示:

```
......

    <Item>This is a normal text</Item>

......
```

  - 字符串“This is a normal text”就称为一个CDATA，它在二进制Xml文件中用一个ResXMLTree_node和一个ResXMLTree_cdataExt来描述



Android资源编译过程

- Step 7.4.6.3: ResXMLTree_cdataExt的定义

```
/**
 * Extended XML tree node for CDATA tags -- includes the CDATA string.
 * Appears header.headerSize bytes after a ResXMLTree_node.
 */
struct ResXMLTree_cdataExt
{
    // The raw CDATA character data.
    struct ResStringPool_ref data;

    // The typed value of the character data if this is a CDATA node.
    struct Res_value typedData;
};
```

Android资源编译过程

- Step 8:生成资源符号
  - 这里生成资源符号为后面生成R.java文件做好准备的
  - 目前所有收集到的资源项都按照类型来保存在一个资源表中
  - 只要依次遍历资源表中的每一个Package的每一个Type的每一个Entry，就可以生成一系列的资源符号及其ID
  - 例如对于strings.xml文件中名称为”start_in_process”的Entry来说，它是一个类型为string的资源项，假设它出现的次序为第3，那么它的资源符号就等于R.string.start_in_process，对应的资源ID就为0x7f050002，其中，高字节0x7f表示Package ID，次高字节0x05表示string的Type ID，低两字节0x02就表示”start_in_process”是第三个出现的字符串

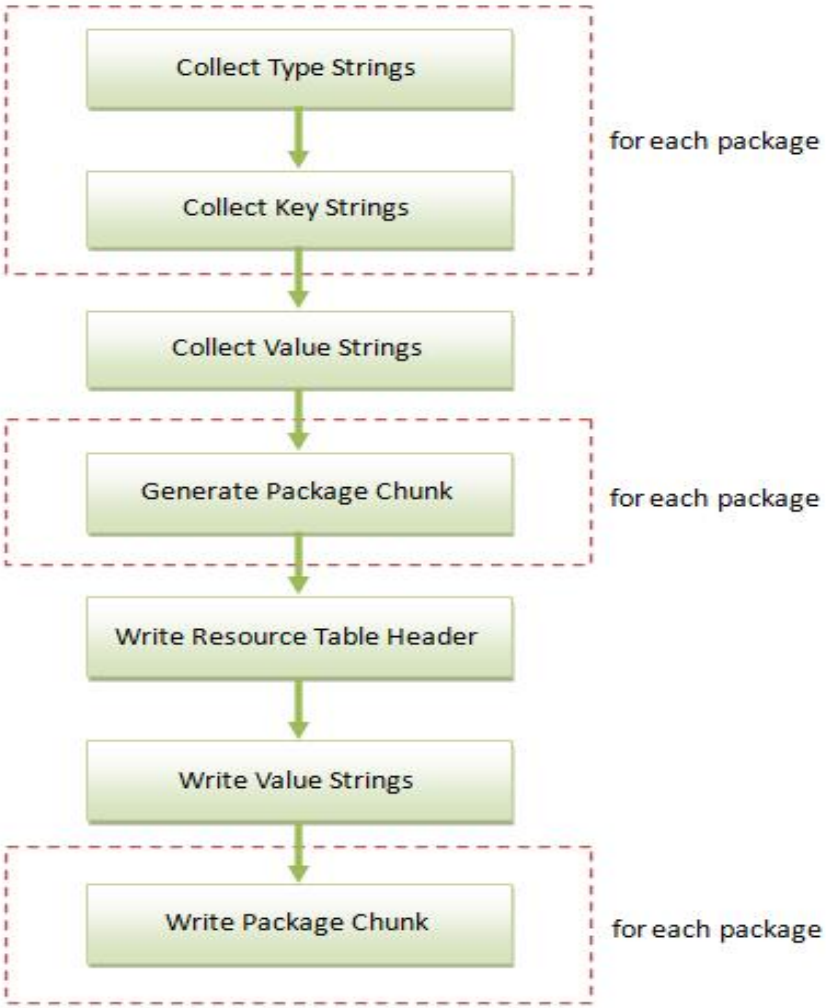
Android资源编译过程

- Step 9:生成资源索引表
  - 经过前面的八个操作，所获得的资源列表如下所示：

Entry Name	Item Value	Config Description	Type
icon.png	res/drawable-ldpi/icon.png	ldpi	drawable
Icon.png	res/drawable-mdpi/icon.png	mdpi	drawable
Icon.png	res/drawable-hdpi/icon.png	hdpi	drawable
main.xml	res/layout/main.xml	default	layout
sub.xml	res/layout/sub.xml	default	layout
app_name	Activity	default	string
sub_activity	Sub Activity	default	string
start_in_process	Start sub-activity in process	default	string
start_in_new_process	Start sub-activity in new process	default	string
finish	Finish activity	default	string
button_start_in_process	-	default	id
button_start_in_new_process	-	default	id

Android资源编译过程

- Step 9: 资源索引表生成过程



Android资源编译过程

- Step 9.1:收集类型字符串
  - 在我们的例子中，一共有4种类型的资源，分别是drawable、layout、string和id，于是对应的类型字符串就为“drawable”、“layout”、“string”和“id”
  - 注意，这些字符串是按Package来收集的，也就是说，当前被编译的应用程序资源有几个Package，就有几组对应的类型字符串，每一个组类型字符串都保存在其所属的Package中

Android资源编译过程

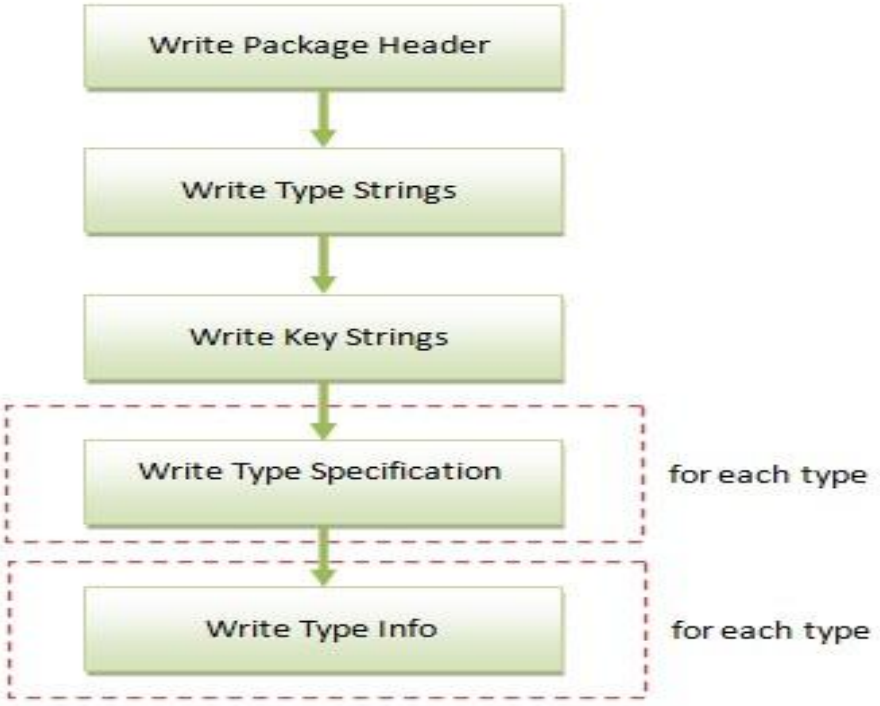
- Step 9.2:收集资源项名称字符串
  - 在我们的例子中，收集到的资源项名称字符串就为”icon”、”main”、”sub”、”app_name”、”sub_activity”、”start_in_process”、”start_in_new_process”、”finish”、”button_start_in_process”和”button_start_in_new_process”。
  - 注意，这些字符串同样是按Package来收集的，也就是说，当前被编译的应用程序资源有几个Package，就有几组对应的资源项名称字符串，每一个组资源项名称字符串都保存在其所属的Package中。

Android资源编译过程

- Step 9.3:收集资源项值字符串
  - 在我们的例子中，一共有12个资源项，但是只有10项是具有值字符串的，它们分别是”res/drawable-ldpi/icon.png”、”res/drawable-mdpi/icon.png”、”res/drawable-hdpi/icon.png”、”res/layout/main.xml”、”res/layout/sub.xml”、”Activity”、”Sub Activity”、”Start sub-activity in process”、”Start sub-activity in new process”和”Finish activity”。
  - 注意，这些字符串不是按Package来收集的，也就是说，当前所有参与编译的Package的资源项值字符串都会被统一收集在一起。

Android资源编译过程

- Step 9.4:生成Package数据块
  - 参与编译的每一个Package的资源项元信息都写在一块独立的数据上，这个数据块使用一个类型为ResTable_package的头部来描述



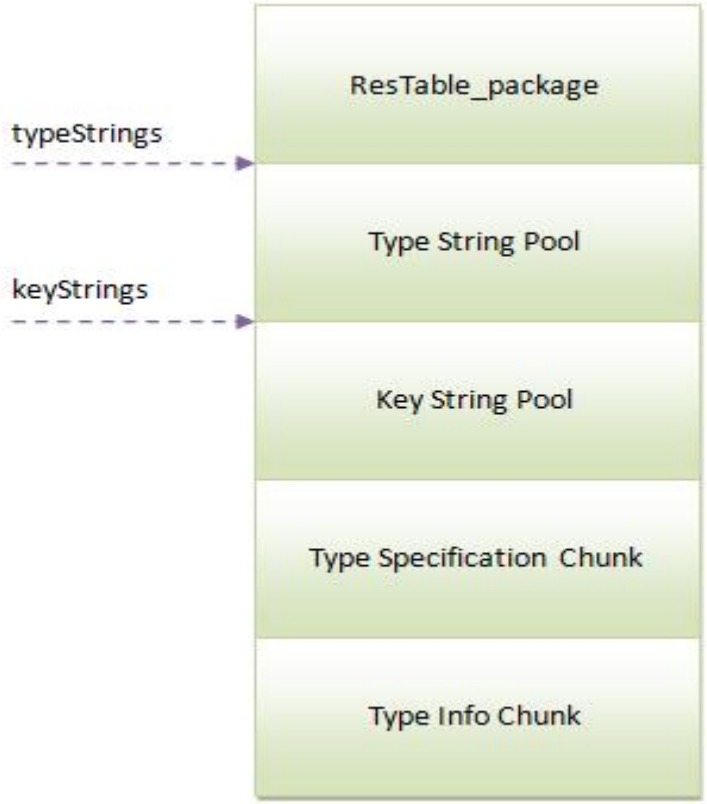
Android资源编译过程

- Step 9.4.1:写入Package资源项元信息数据块头部，即一个ResTable_package结构体

```
struct ResTable_package
{
    struct ResChunk_header header;

    // If this is a base package, its ID.  Package IDs start
    // at 1 (corresponding to the value of the package bits in a
    // resource identifier).  0 means this is not a base package.
    uint32_t id;

    // Actual name of this package, \0-terminated.
    char16_t name[128];

    // Offset to a ResStringPool_header defining the resource
    // type symbol table.  If zero, this package is inheriting from
    // another base package (overriding specific values in it).
    uint32_t typeStrings;

    // Last index into typeStrings that is for public use by others.
    uint32_t lastPublicType;

    // Offset to a ResStringPool_header defining the resource
    // key symbol table.  If zero, this package is inheriting from
    // another base package (overriding specific values in it).
    uint32_t keyStrings;

    // Last index into keyStrings that is for public use by others.
    uint32_t lastPublicKey;
};
```

Android资源编译过程

- Step 9.4.1: ResTable_package结构体图示



Android资源编译过程

- Step 9.4.1: public资源
  - 定义在res/values/public.xml文件，形式如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <public type="string" name="string3" id="0x7f040001" />
</resources>
```

  - 用来告诉Android资源编译工具将类型为string的资源string3的ID固定为0x7f040001，以便第三方应用程序可以固定地通过0x7f040001来访问字符串”string3”

Android资源编译过程

- Step 9.4.2:写入类型字符串资源池
  - 前面已经将每一个Package用到的类型字符串收集起来了，因此，这里就可以直接将它们写入到Package资源项元信息数据块头部后面的那个数据块去。
- Step 9.4.3:写入资源项名称字符串资源池
  - 前面已经将每一个Package用到的资源项名称字符串收集起来了，这里就可以直接将它们写入到类型字符串资源池后面的那个数据块去。

Android资源编译过程

- Step 9.4.4:写入类型规范数据块
  - 类型规范数据块用来描述资源项的配置差异性。通过这个差异性描述，我们就可以知道每一个资源项的配置状况。
  - 知道了一个资源项的配置状况之后，Android资源管理框架在检测到设备的配置信息发生变化之后，就可以知道是否需要重新加载该资源项。
  - 类型规范数据块是按照类型来组织的，也就是说，每一种类型都对应有一个类型规范数据块。

Android资源编译过程

- Step 9.4.4:类型规范数据块的头部用一个ResTable_typeSpec来定义

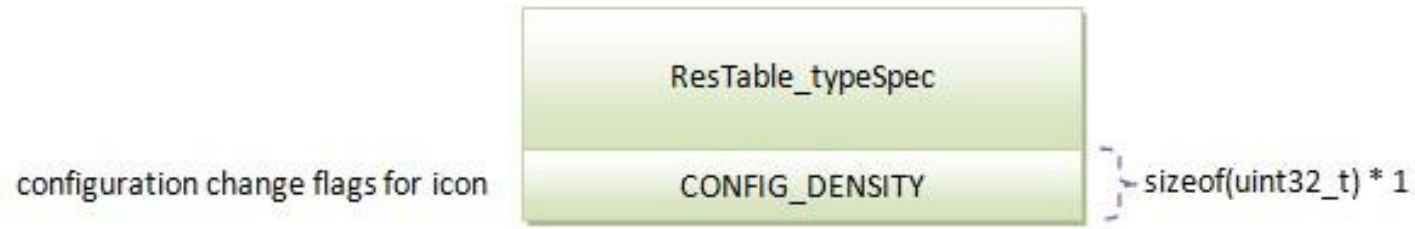
```
struct ResTable_typeSpec
{
    struct ResChunk_header header;

    // The type identifier this chunk is holding.  Type IDs start
    // at 1 (corresponding to the value of the type bits in a
    // resource identifier).  0 is invalid.
    uint8_t id;

    // Must be 0.
    uint8_t res0;
    // Must be 0.
    uint16_t res1;

    // Number of uint32_t entry configuration masks that follow.
    uint32_t entryCount;

    enum {
        // Additional flag indicating an entry is public.
        SPEC_PUBLIC = 0x40000000
    };
};
```

Android资源编译过程

- Step 9.4.4: ResTable_typeSpec后面紧跟着的是一个大小为entryCount的uint32_t数组，
  - 每一个数组元数，即每一个uint32_t，都是用来描述一个资源项的配置差异性的
  - Android资源管理框架根据这个差异性信息，就可以知道当设备配置发生变化时，是否需要重新加载资源

Android资源编译过程

- Step 9.4.4: 在我们的例子中，类型为drawable的规范数据块内容：



  - 由此可 知，类型为drawable的资源项icon在设备的屏幕密度发生变化之后，Android资源管理框架需要重新对它进行加载，以便获得更合适的资源项

Android资源编译过程

- Step 9.4.4: 在我们的例子中，类型为layout的规范数据块内容：

  - 由此可 知，类型为layout的资源项在设备配置变化之后，Android资源管理框架无需重新对它们进行加载，因为只有一种资源

Android资源编译过程

- Step 9.4.4: 在我们的例子中，类型为string的规范数据块内容：

  - 由此可 知，类型为string的资源项在设备配置变化之后，Android资源管理框架无需重新对它们进行加载，因为只有一种资源

Android资源编译过程

- Step 9.4.4: 在我们的例子中，类型为string的规范数据块内容：

  - 由此可 知，类型为string的资源项在设备配置变化之后，Android资源管理框架无需重新对它们进行加载，因为只有一种资源

Android资源编译过程

- Step 9.4.5:写入类型资源项数据块
  - 类型资源项数据块用来描述资源项的具体信息，这样我们就可以知道每一个资源项名称、值和配置等信息。类型资源项数据同样是按照类型和配置来组织的，也就是说，一个具有N个配置的类型一共对应有N个类型资源项数据块。

Android资源编译过程

- Step 9.4.5:类型资源项数据块的头部用一个ResTable_type来定义

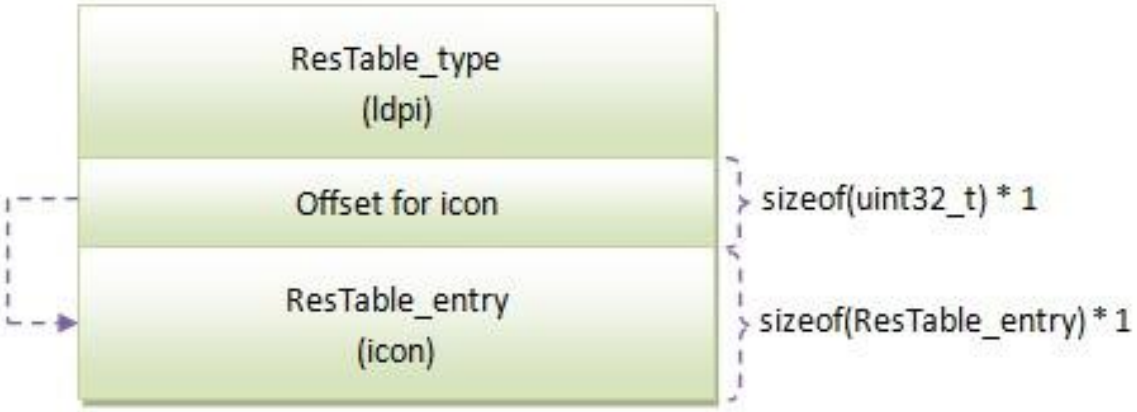
```
struct ResTable_type
{
    struct ResChunk_header header;

    enum {
        NO_ENTRY = 0xFFFFFFFF
    };

    // The type identifier this chunk is holding.  Type IDs start
    // at 1 (corresponding to the value of the type bits in a
    // resource identifier).  0 is invalid.
    uint8_t id;

    // Must be 0.
    uint8_t res0;
    // Must be 0.
    uint16_t res1;

    // Number of uint32_t entry indices that follow.
    uint32_t entryCount;

    // Offset from header where ResTable_entry data starts.
    uint32_t entriesStart;

    // Configuration this collection of entries is designed for.
    ResTable_config config;
};
```

Android资源编译过程

- Step 9.4.5: ResTable_type紧跟着的是一个大小为entryCount的uint32_t数组，每一个数组元数，即每一个uint32_t，都是用来描述一个资源项数据块的偏移位置。紧跟在这个uint32_t数组后面的是一个大小为entryCount的ResTable_entry数组，每一个数组元素，即每一个ResTable_entry，都是用来描述一个资源项的具体信息。在我们的例子中，一共有4种不同类型的资源项，其中，类型为drawable的资源有1个资源项以及3种不同的配置，类型为layout的资源有2个资源项以及1种配置，类型为string的资源有5个资源项以及1种配置，类型为id的资源有2个资源项以及1种配置。这样一共就对应有3 + 1 + 1 + 1个类型资源项数据块。

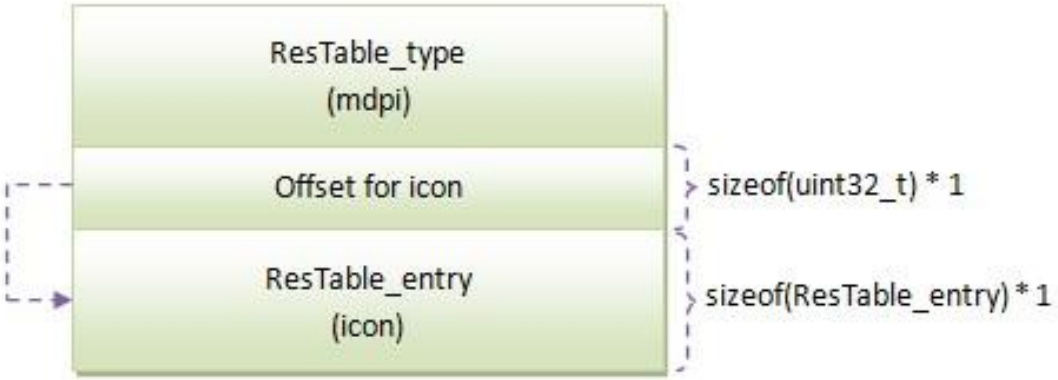
Android资源编译过程

- Step 9.4.5:类型为drawable和配置为ldpi的资源项数据块



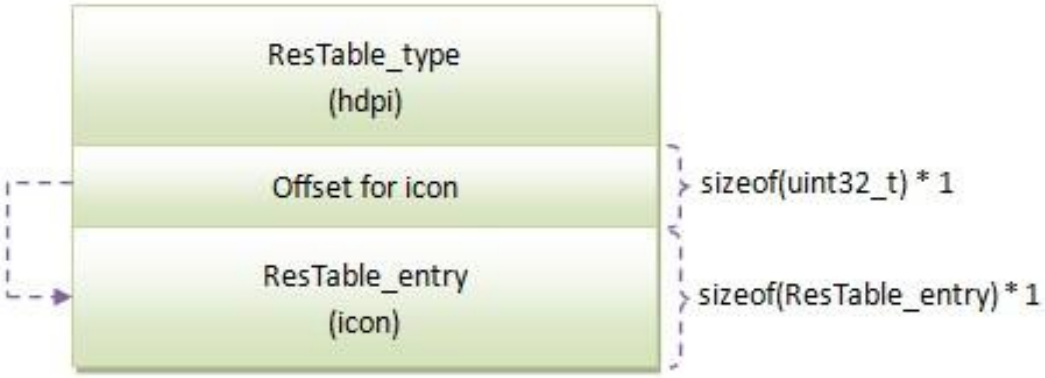
Android资源编译过程

- Step 9.4.5:类型为drawable和配置为mdpi的资源项数据块



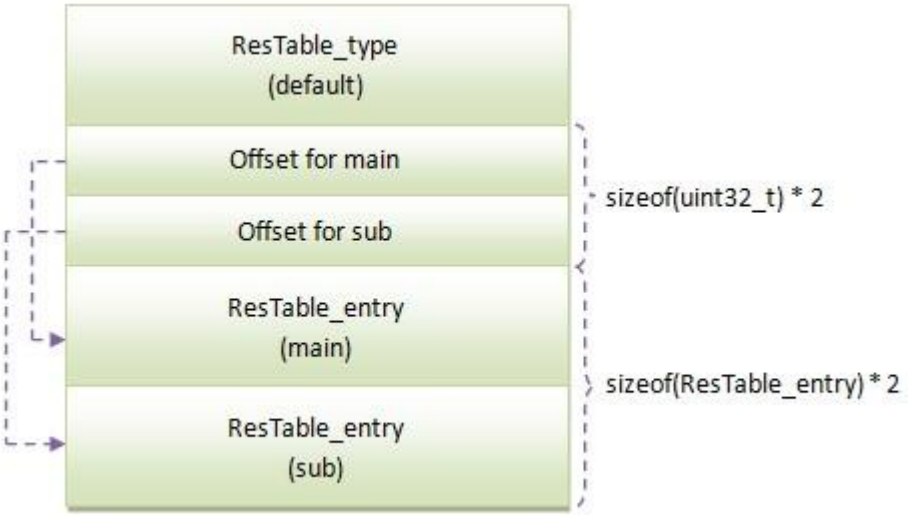
Android资源编译过程

- Step 9.4.5:类型为drawable和配置为hdpi的资源项数据块



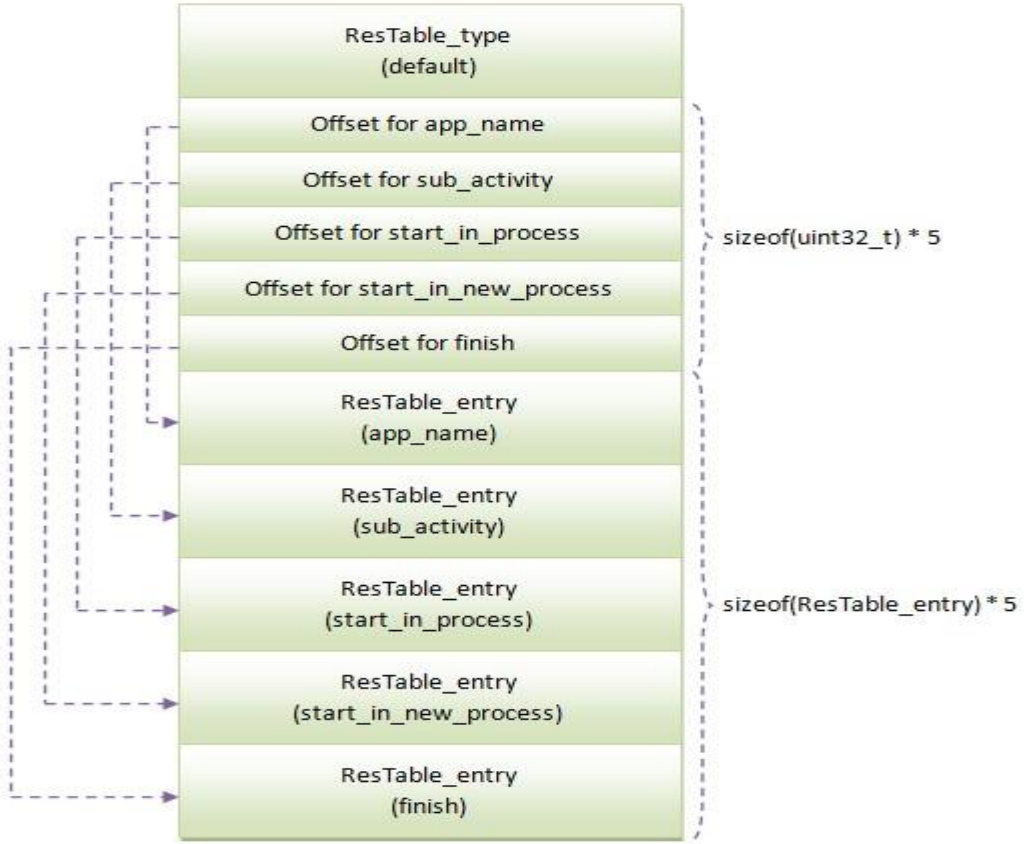
Android资源编译过程

- Step 9.4.5:类型为layout和配置为default的资源项数据块



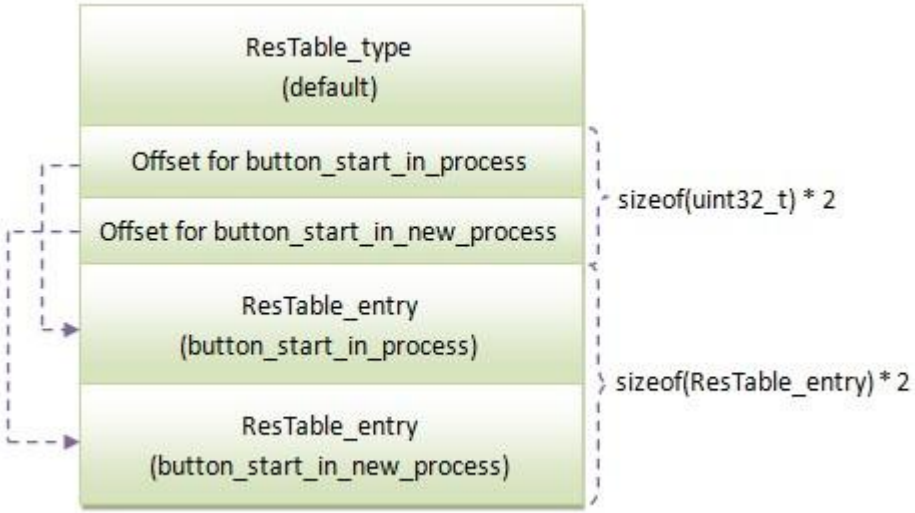
Android资源编译过程

- Step 9.4.5:类型为string和配置为default的资源项数据块



Android资源编译过程

- Step 9.4.5:类型为id和配置为default的资源项数据块



Android资源编译过程

- Step 9.4.5: 每一个资源项数据都是通过一个ResTable_entry来定义的

```
struct ResTable_entry
{
    // Number of bytes in this structure.
    uint16_t size;

    enum {
        // If set, this is a complex entry, holding a set of name/value
        // mappings.  It is followed by an array of ResTable_map structures.
        FLAG_COMPLEX = 0x0001,
        // If set, this resource has been declared public, so libraries
        // are allowed to reference it.
        FLAG_PUBLIC = 0x0002
    };
    uint16_t flags;

    // Reference into ResTable_package::keyStrings identifying this entry.
    struct ResStringPool_ref key;
};
```

Android资源编译过程

- Step 9.4.5: 普通资源项数据都是通过一个Res_value来定义的

Android资源编译过程

- Step 9.4.5: 自定义资源项数据都是通过一个ResTable_map_entry以及若干个ResTable_map来定义的

Android资源编译过程

- Step 9.4.5: 以前面的自定义资源custom_orientation为例：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <attr name="custom_orientation">
        <enum name="custom_vertical" value="0" />
        <enum name="custom_horizontal" value="1" />
    </attr>
</resources>
```

  - 它有三个bag，分别是^type、custom_vertical和custom_horizontal，其中，custom_vertical和custom_horizontal是两个自定义的bag，它们的值分别等于0x0和0x1，而^type是一个系统内部定义的bag，它的值固定为0x10000。注意，^type、custom_vertical和custom_horizontal均是类型为id的资源，假设它们分配的资源ID分别为0x1000000、0x7f040000和7f040001。

Android资源编译过程

- Step 9.4.5: ResTable_map_entry的定义如下所示：

```
struct ResTable_ref
{
    uint32_t ident;
};

/**
 * Extended form of a ResTable_entry for map entries, defining a parent map
 * resource from which to inherit values.
 */
struct ResTable_map_entry : public ResTable_entry
{
    // Resource identifier of the parent mapping, or 0 if there is none.
    ResTable_ref parent;
    // Number of name/value pairs that follow for FLAG_COMPLEX.
    uint32_t count;
};
```

Android资源编译过程

- Step 9.4.5: ResTable_map的定义如下所示:

```
struct ResTable_map
{
    // The resource identifier defining this mapping's name.  For attribute
    // resources, 'name' can be one of the following special resource types
    // to supply meta-data about the attribute; for all other resource types
    // it must be an attribute resource.
    ResTable_ref name;

    .....

    // This mapping's value.
    Res_value value;
};
```

Android资源编译过程

- Step 9.5:写入资源索引表头部
  - 资源索引表头部使用一个ResTable_header来表示

```
/**
 * Header for a resource table.  Its data contains a series of
 * additional chunks:
 *   * A ResStringPool_header containing all table values.
 *   * One or more ResTable_package chunks.
 *
 * Specific entries within a resource table can be uniquely identified
 * with a single integer as defined by the ResTable_ref structure.
 */
struct ResTable_header
{
    struct ResChunk_header header;

    // The number of ResTable_package structures.
    uint32_t packageCount;
};
```

Android资源编译过程

- Step 9.6:写入资源项的值字符串资源池
  - 前面已经将所有的资源项的值字符串都收集起来了，因此，这里直接它们写入到资源索引表去就可以了。
  - 注意，这个字符串资源池包含了在所有的资源包里面所定义的资源项的值字符串，并且是紧跟在资源索引表头部的后面。

Android资源编译过程

- Step 9.7:写入Package数据块
  - 前面已经所有的Package数据块都收集起来了，因此，这里直接将它们写入到资源索引表去就可以了。
  - 这些Package数据块是依次写入到资源索引表去的，并且是紧跟在资源项的值字符串资源池的后面。

Android资源编译过程

- Step 10:编译AndroidManifest.xml文件
  - 经过前面的操作，应用程序的所有资源项就编译完成了，这时候就开始将应用程序的配置文件AndroidManifest.xml也编译成二进制格式的Xml文件。
  - 之所以要在应用程序的所有资源项都编译完成之后，再编译应用程序的配置文件，是因为后者可能会引用到前者。

Android资源编译过程

- Step 11:生成R.java文件
  - 前面已经将所有的资源项及其所对应的资源ID都收集起来了，因此，这里只要将直接将它们写入到指定的R.java文件去就可以了。
  - 例如，假设分配给类型为layout的资源项main和sub的ID为0x7f030000和0x7f030001，那么在R.java文件，就会分别有两个以main和sub为名称的常量，如下所示：

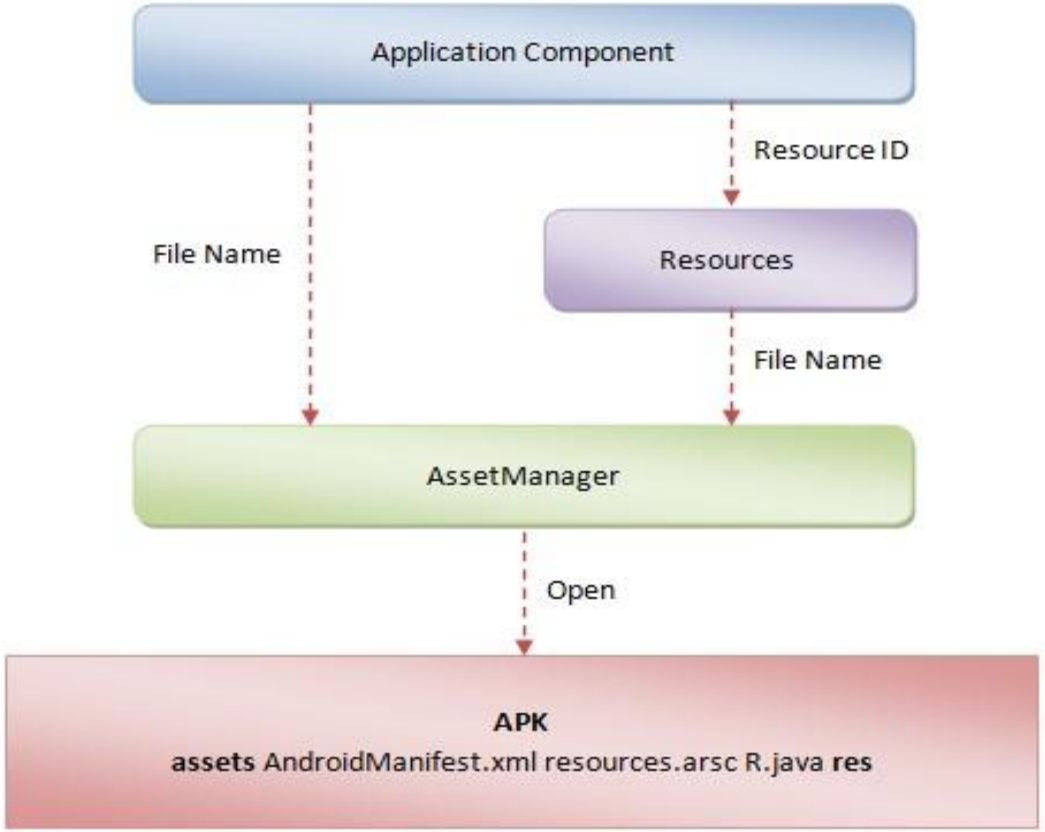
```
public final class R {
    ......

    public static final class layout {
        public static final int main=0x7f030000;
        public static final int sub=0x7f030001;
    }

    ......
}
```

Android资源编译过程

- Step 12:打包资源文件到APK包
  - assets目录
  - res目录，但是不包括values子目录的文件，这些文件的资源项已经包含在resources.arsc文件中
  - resources.arsc
  - AndroidManifest.xml

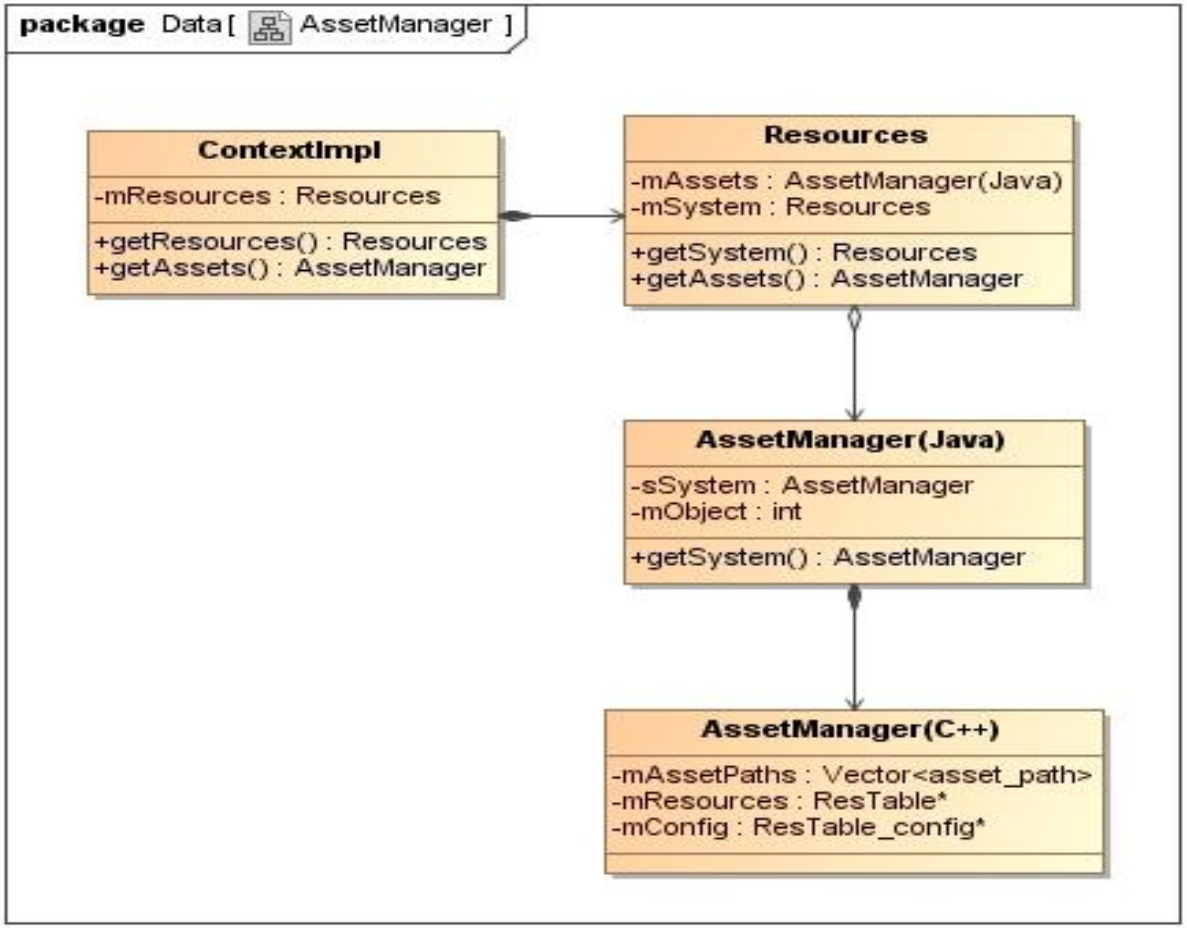
Android资源查找过程

- Android资源查找框架



Android资源查找过程

- Android资源查找框架相关实现类



Android资源查找过程

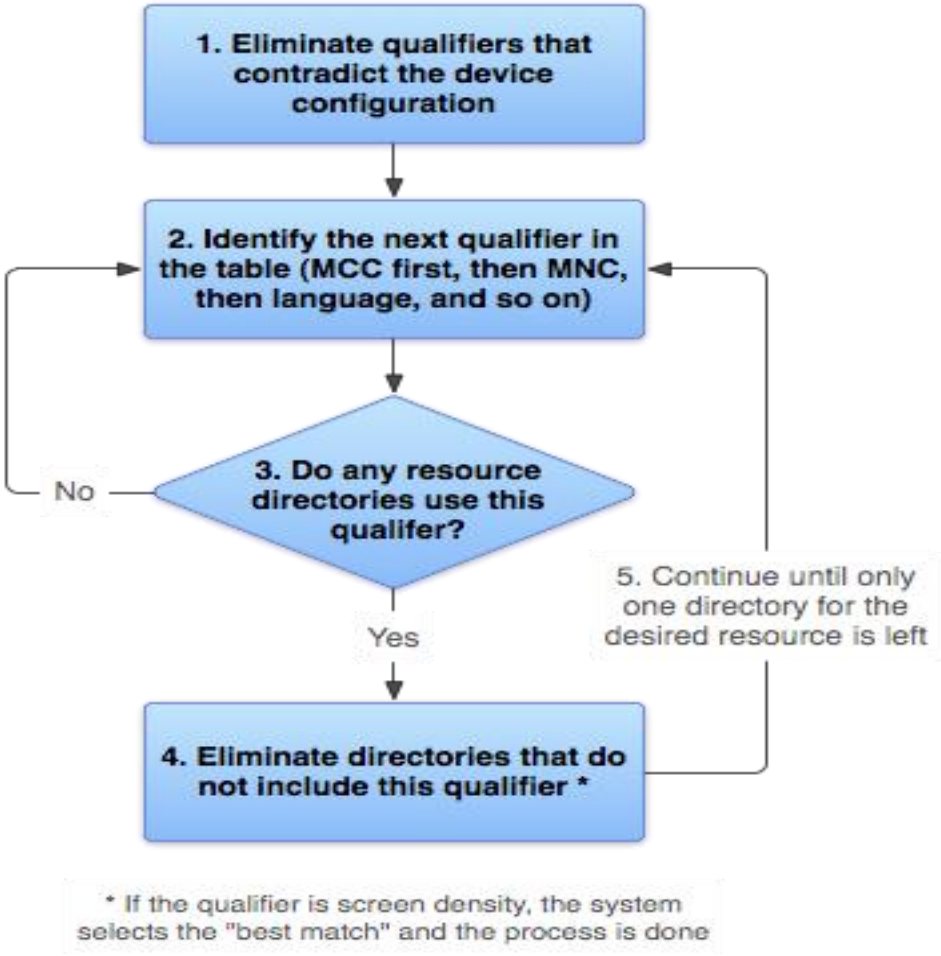
- Android资源查找框架的初始化
  - 设置资源路径
    - AssetManager::addAssetPath
      - /system/framework/framework-res.apk
      - /vendor/overlay/framework/framework-res.apk(optional)
      - Self Apk File
  - 设置配置信息
    - AssetManager::setConfiguration

Android资源查找过程

- 根据资源ID找到资源Value
  - Step 1: 根据资源ID的Package ID在resources.arsc中找到对应的Package数据块(ResTable_package)
  - Step 2: 根据资源ID的Type ID在Package数据块找到对应的类型规范数据块(ResTable_typeSpec)和类型资源项数据块(ResTable_type)
  - Step 3: 根据资源ID的Entry ID在对应的类型资源项数据块中找到与当前设备配置最匹配的资源项
  - Step 4: 返回最匹配的资源项值及其配置差异性
- 找到的资源Value是一个文件路径，则通过AssetManager打开该文件，并且对它进行解析以及使用，否则直接使用

Android资源查找过程

- 资源匹配算法



Android资源查找过程

- 资源匹配算法示例
  - 资源配置清单

```
drawable/
drawable-en/
drawable-fr-rCA/
drawable-en-port/
drawable-en-notouch-12key/
drawable-port-ldpi/
drawable-port-notouch-12key/
```

  - 设备配置信息

```
Locale = en-GB
Screen orientation = port
Screen pixel density = hdpi
Touchscreen type = notouch
Primary text input method = 12key
```

Android资源查找过程

- Step 1: 消除与设备配置冲突的drawable目录，即drawable-fr-rCA目录，因为设备设置的语言是en-GB：

```
drawable/
drawable-en/
drawable-en-port/
drawable-en-notouch-12key/
drawable-port-ldpi/
drawable-port-notouch-12key/
```

Android资源查找过程

- Step 2:从MMC开始，选择一个资源组织维度来过渡从Step 1筛选后剩下来的目录。
- Step 3:检查Step 2选择的维度是否有对应的资源目录。如果没有，就返回到Step 2继续处理。如果有，那么就继续往下执行Step 4。在我们示例中，要一直重复执行Step 2，直到检查到language这个维度时。
- Step 4: 消除那些不包含有Step 2所选择的资源维度的目录。在我们的示例中，就是要消除那些不包含有en这个language的目录：

```
drawable-en/
drawable-en-port/
drawable-en-notouch-12key/
```

Android资源查找过程

- Step 5:继续执行Step 2、Step 3和Step 4，直到找到一个最匹配的资源目录为止，即剩下最后一个目录为止。在我们的示例中，下一个要检查的维度是screen orienation。由于设备的screen orienation为port。因此，所有不包含有port资源维度的目录将被消除：

```
drawable-en-port/
```

Q&A

Thank You