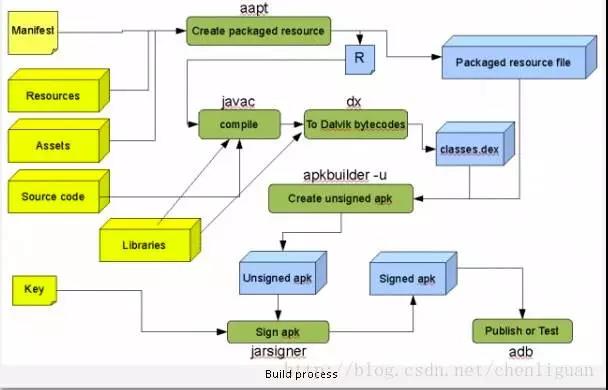
Android应用程序主要由两部分内容组成：代码和资源。

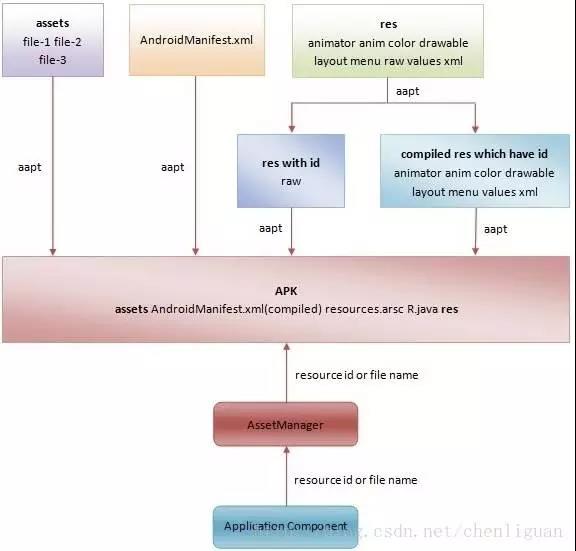
资源主要就是指那些与UI相关的东西，例如UI布局、字符串和图片等。

代码和资源分开可以使得应用程序在运行时根据实际需要来组织UI。这样就可使得应用程序只需要编译一次，就可以支持不同的UI布局。这种特性使得应用程序在运行时可以适应不同的屏幕大小和密度，以及不同的国家和语言等。



aapt编译：

aapt即Android Asset Packaging Tool，在SDK的build-tools目录下。该工具可以查看，创建， 更新ZIP格式的文档附件(zip, jar, apk)。也可将资源文件编译成二进制文件。



通过图我们就可以看出：

A. 除了assets和res/raw资源被原装不动地打包进APK之外，其它的资源都会被编译或者处理。

B. 除了assets资源之外，其它的资源都会被赋予一个资源ID。

C. 打包工具负责编译和打包资源，编译完成之后，会生成一个resources.arsc文件和一个R.java，前者保存的是一个资源索引表，后者定义了各个资源ID常量。

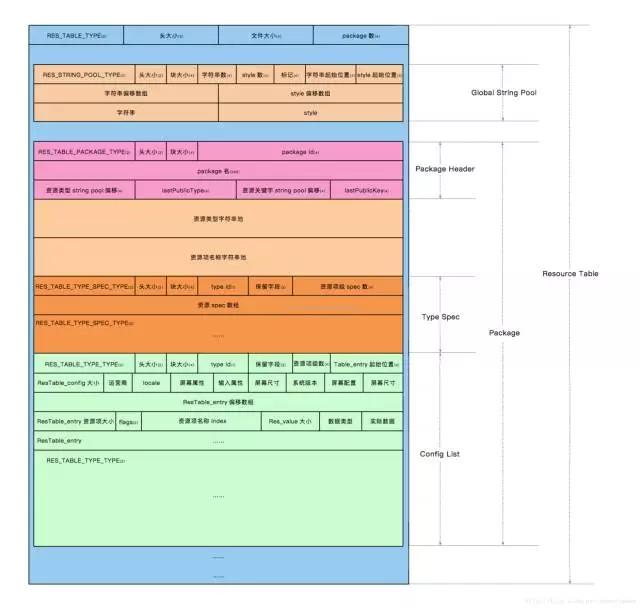
D. 应用程序配置文件AndroidManifest.xml同样会被编译成二进制的XML文件，然后再打包到APK里面去。

E. 应用程序在运行时通过AssetManager来访问资源，或通过资源ID来访问，或通过文件名来访问。

其中resources.arsc文件，这个文件记录了所有的应用程序资源目录的信息，包括每一个资源名称、类型、值、ID以及所配置的维度信息。我们可以将这个resources.arsc文件想象成是一个资源索引表，这个资源索引表在给定资源ID和设备配置信息的情况下，能够在应用程序的资源目录中快速地找到最匹配的资源。

resources.arsc文件解析





resources.arsc一共有五种chunk类型，分别为TYPETABLE，TYPEPACKAGE，TYPE\_STRING ,TYPETYPE，TYPECONFIG。

— table，是整个reousces table的开始，它的chunksize即是整个文件的大小。

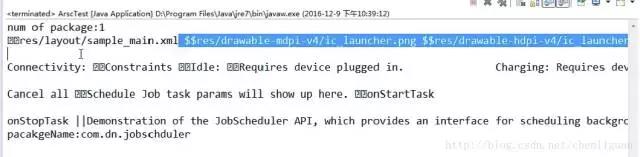
— package，指的是一个package的开始，其实在resources,arsc是可以有多个package的。而packageID即是资源resID的最高八位，一般来说系统android的是1(0x01)，普通的例如com.tencent.mm会是127(0x7f)，剩下的是从2开始起步。当然这个我们在aapt也是可以指定的(1-127即八位的合法空间,一些混合编译就是改这个packageID)。

—string, 代表stringblock，我们一共有三种类型的stringblock。分别是table stringblock,typename stringblock, specsname stringblock。

— type，这里讲的是typename stringblock里面我们用到的各种type(用到多少种类型的type,就有多少个type chunk)，例如attr, drawable, layout, id, color, anim等，Type ID是紧跟着Package ID。

— config, 即是Android用来描述资源维度，例如横竖屏，屏幕密度，语言等。对于每一种type，它定义了多少种config，它后面就紧跟着多少个config chunk,例如我们定义了drawable-mdpi,drawable-hdpi,那后面就会有两个config。

将Resource.arsc文件部分内容读出显示如下：



混淆过程

系统编译完成apk文件以后：

映射关系：res/drawable/ic\_launcher.png —– > 0x7f020000

再做“混淆”：要实现将res/drawable/ic\_launcher.png图片改成a.png

drawable文件的名字

String文件的名字

layout的名字

比如：R.string.description—>R.string.a

res/drawable/ic\_launcher.png图片改成a.png

还可以更加夸张

res/drawable—>r/d

res/value–>r/v

res/drawable/ic\_launcher.png图片改成r/d/a.png

读取resources.arsc二进制文件，然后修改某一段一段的字节。

有一段叫做：res/drawable/ic\_launcher.png 在自己数组当中的第800位-810位

将这一段第800位-810位替换成改成r/d/a.png 的字节码。

zipalign优化

我们知道APK其实就是一个Zip压缩文件，从原理上来讲就是通过格式化Zip文件夹中二进制文件的序列，达到提升系统解析速度。就像我们在阅读代码的过程中先格式化一遍代码，会让我们更容易理解其含义一样。

在Android平台中，数据文件存储在apk文件中，可以多进程的访问，如果你开发过Win32可能知道程序的粒度对齐问题，不错虽然不是PE格式的文件，在Zip中一样，资源的访问可以通过更好的对其优化，而zipalign使用了4字节的边界对齐方式来影射内存，通过空间换时间的方式提高执行效率。

我理解的是：通俗的说就是按着有利于系统处理的排列方式，

对我们apk中的资源文件进行排列，提高资源的查找速度，从而去提高应用的运行效率。

先签名再对齐,否则先对齐再签名会破坏对齐

使用

release {

// 不显示Log

buildConfigField "boolean", "LOG\_DEBUG", "false"

//混淆

minifyEnabled true

//Zipalign优化

zipAlignEnabled true

// 移除无用的resource文件

shrinkResources true

......

}

使用ReDex优化

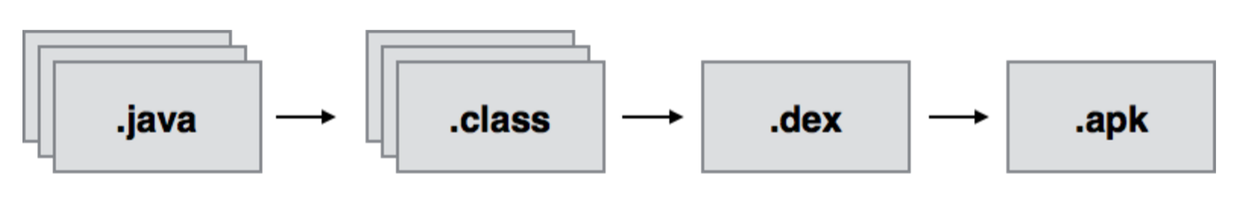
ReDex是Facebook开源一个减小安卓app大小以提高性能的工具，内嵌以及清除僵尸代码这样的优化来减小字节码，主要是对Dex进行了优化，能让APK 运行更快，不过需要多测试是否会崩溃。

经过 Redex 转换后的 APK，体积变得更小，运行速度变得更快。Redex 基于管道的方式来优化 Android 的 .dex 文件，一个源 .dex 文件通过管道进行一系列的自定义转换后，将得到一个优化的 .dex 文件。

我们知道 Android 的编译过程首先是通过 javac 工具将 .java 文件编译成 .class 文件，接着将所有的 .class 文件合并成 Dalvik 虚拟机的可执行文件 .dex，最后再跟其他资源等文件一起压缩成 APK 文件

大致流程如下所示：

转换的时机



Redex 选择基于字节码文件而不是 Java 源码进行优化，是因为字节码相比 Java 源码而言，可以进行更为全局的，类与类之间的优化，而不是单个类文件的局部优化；选择基于 dex 字节码而不是 Java 字节码进行优化，是因为某些优化只能在 dex 文件中进行。

管道的思想

鉴于随着时间的推移，开发人员可能会不断得到新的优化 idea，为了方便的将新的优化点加入既有的代码中，同时也方便不同开发人员并行开发优化点， 所以 Redex 选择基于管道的思想来实现 dex 的优化，这样每一个优化的 idea 可以通过插件的形式集成到管道中，实现即插即用，也不会影响其他的优化插件，整体优化流程如下所示：

