**安卓压缩工具集说明文档**

1. **工具集介绍**

安卓压缩工具集提供了一个极为简洁的方法，能够比安卓原有的Zip提供更高压缩比的存储应用内的so文件 （后期版本还可以支持压缩动态加载的jar包，以及游戏资源文件），同时提供了应用内网络更新下载压缩文件的方法，使得应用可以将部分so存储到云端，减小应用的尺寸。

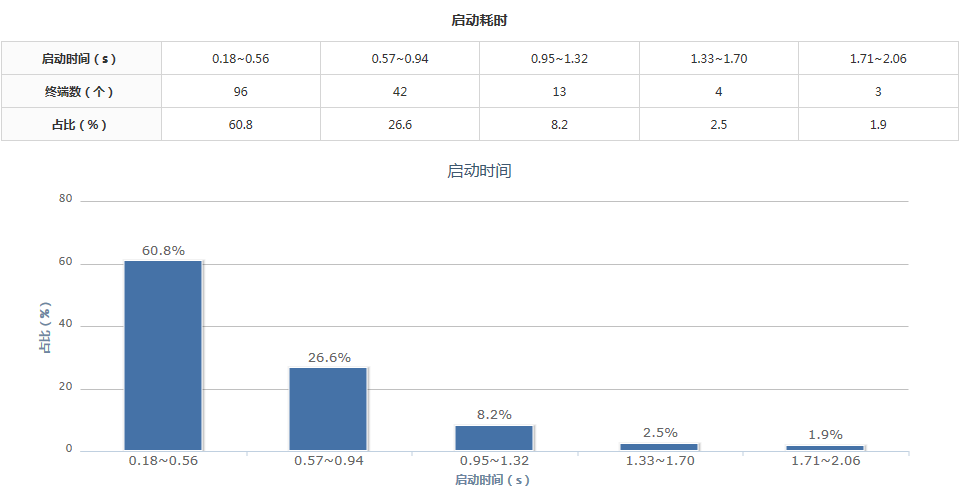
**节省50%空间！**

在云测平台上测试了158款终端，涵盖2.3到4.4多个系统版本，100%通过



**100%通过测试！**



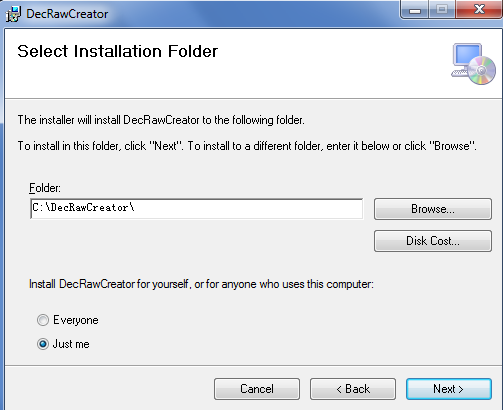


**8M文件1s内解压**

压缩原理： 压缩工具会把所有的so使用LZMA算法压缩到assert目录，应用在第一次启动的时候，会解压到应用的私有目录下

1. **工具集组成**

工具集为一个安装程序，建议安装在默认路径下，安装在program files下在win7可能有读写权限的问题导致一些异常



安装后，你可以看见4个目录，此目录内都含有源码。

安装后的四个目录如下



其中 ApkLibComrepss 为java命令行程序的源码，在此目录的bin子目录中，你可以找到ApkCompress.jar ，使用这个文件可以把一个普通的apk文件转换为压缩的apk文件

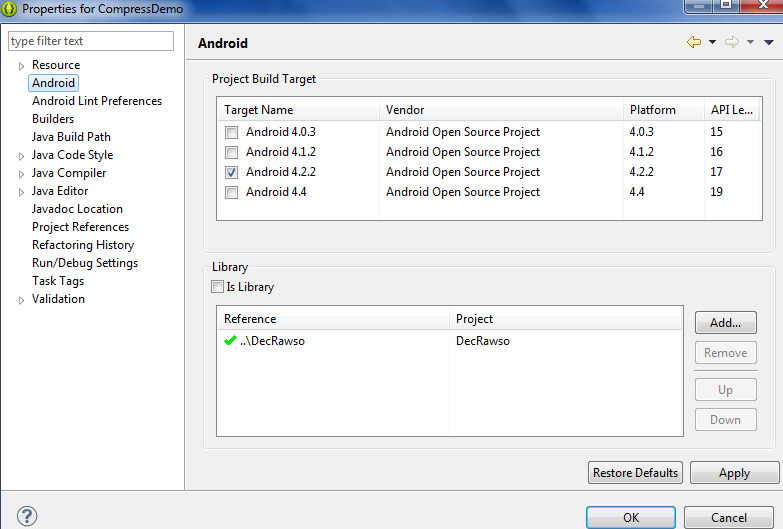
CompressDemo为一个样例代码，你可以参考这个代码知道如何整合压缩的SDK。

DecRawso是压缩的SDK，你的开发工程需要引用这个SDK，并进行一些源码上的修改，才能整合压缩的功能

RawsoCreator为windows下的转换工具， 这个工具一般无需使用， 仅仅在调试和二次开发压缩SDK的时候使用。

1. **如何整合压缩SDK**

打开CompressDemo，我们以这个工程为例子讲解如何整合压缩SDK

1. 首先需要引入DecRawso工程  
   
2. 然后需要在你的工程内最初始的地方调用**DecRawso.NewInstance**。在此demo工程内，是在MainActivity.java的OnCreate内调用了此方法， 此方法是创建了一个解压的唯一实例。注意：此方法是异步的，所以你可以传入一个handler接受异步解码完成的消息，如果同时传入参数showProgress=true，SDK内会产生一个进度对话框以阻塞主进程。不推荐使用DecRawso.NewInstance(mContext,null,false);的方式，此方式不接受任何消息，且无进度对话框，解压会在后台自动完成，并且在应用第一次load so的时候阻塞直到后台解压完成。所以如果阻塞时间过长，可能会导致应用无响应。
3. 修改load so文件的方法：所有的System.loadlibrary(\*\*\*) 改为  System.load(DecRawso.GetInstance().GetPath(“\*\*\*"));  
   新版本， 这步可以省略了，sdk会修改system的libaray加载路径，一般情况下，系统升级不会出问题 （非正规代码，小概率会随android升级修改新的代码），如果方便的话，还是采用System.load(DecRawso.GetInstance().GetPath(“\*\*\*"))

经过这几个简单的步骤，压缩的SDK已经整合到工程内了。

1. **如何压缩发布APK**

使用ApkCompress.jar压缩发布APK。 此工具为命令行工具。一般的此命令使用方式为：在命令行运行ComPressApk.jar -a C:/my/test.apk -k c:/key \*\*\* ### alias -x86 <http://www.test.com> （也可以运行 java –jar ComPressApk.jar ）

-a 后面跟apk路径名， 可以不是全路径

-k 后面是签名文件[key storepass keypass alias name] ，key可以不是全路径名 (name 如果不写， 默认就是CERT)

-x86 表示需要存储x86库文件在云端， 后面跟以http://开头的链接，最后实际的存储位置应该为 [**http://www.test.com/cloudrawso\_x86**](http://www.test.com/cloudrawso_x86)

命令执行完以后， 会生成test\_CompressAlign.apk. 这个apk就是压缩后的apk

1. **开发模式和压缩模式**

为了方便开发，在实现开发的过程中（修改了源码支持压缩后），也可以不压缩so，apk也可以正常运行，压缩的SDK内部会自动判断是否有压缩包， 如果没有压缩包，则加载的路径恢复成android默认的路径。所以最方便的开发是，先整合代码，在开发过程中和原来一样开发（不压缩），在发布的时候才压缩apk

1. **X86和ARM库混合调用**

在实现开发过程中，可能会有某些第三方库确实没有x86版本，通常情况下ISV并不在x86目录下放置arm的第三方库，那么在实际运行过程中会导致缺库现象的发生。在缺库的情况下，压缩的SDK会在x86设备上自动解压arm的压缩包，避免缺库现象的发生。（只有真正加载了缺失的库才是缺库，库文件不一致并不一定就是缺库）

但是显然这样会导致运行的低效率，如果在第三方so和x86的库完全没有相互引用的情况下（也就是说这些库都是java层使用JNI调用的，在native层没有相互调用），可以拷贝arm的第三方库到x86目录下，这样就不会出现缺库的情况。当然这种情况会导致arm库多余的拷贝，在以前的zip压缩情况下，会使得压缩包变大，但是在新的LZMA压缩情况下，库大小完全不会增大，因为LZMA压缩由于字典比较大，能够尽可能的压缩关联的几个文件，如果文件完全相同，LZMA的压缩会和单个文件基本一致。如下图

