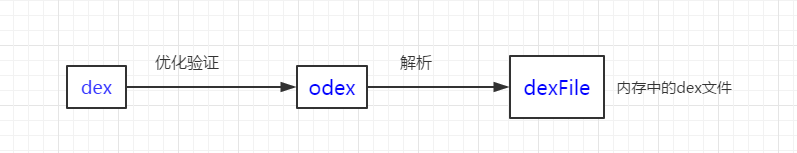
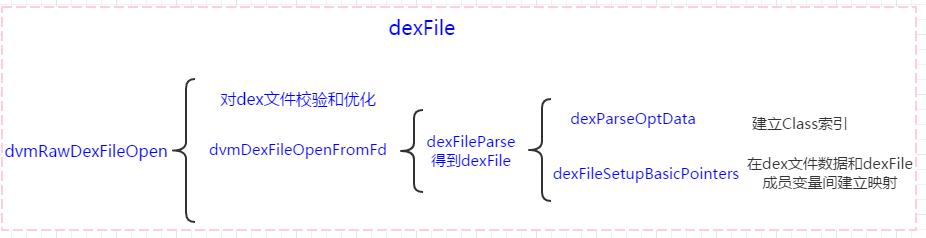
Dalvik虚拟机利用linux进程管理的特点，进行面向对象的设计，可以同时运行多个进程。应用程序运行在Dalvik虚拟机中，并且每个应用程序对应一个Dalvik虚拟机。这样多进程之间不相互影响。Dalvik早期没有JIT编译器，直到Android2.2时候才加入。　android在安装apk时会验证dex文件是否符合规范，并优化dex生成odex；接着会将odex文件解析成易于被dalvik加载和执行的dexFile结构体。下面我们就跟随源码看下dex的心路历程吧。

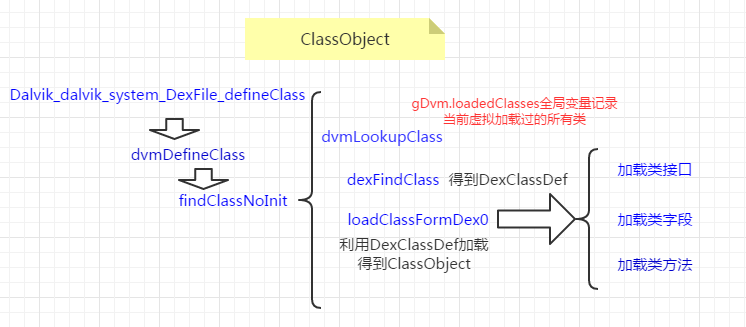
　　apk的安装在java层由Installer类执行，但最终通过socket由C层的installd来实现。dex的优化是由installd的do\_dexopt函数调用dexopt来完成操作。dex优化后生成odex后就要被拉去解析啦。odex解析成dexFile结构体后，类加载器可以通过该结构体实现对Dex文件中各类数据的查找和获取(由此可知，dexFile和dex文件有对应关系)。看解析流程：



　　ok，得到dexFile，至此类加载器的索引文件得到了。在dalvik代码执行中2个数据结构很重要：ClassObject、Method。总所周知java是面向对象的语言，代码是组合在类中的。故虚拟机的加载机制可以通过按代码执行流程来逐渐加载相关类(apk加壳方案？)；类加载通过类描述符在dexFile(所以它叫dex在内存中的描述)结构体得到其在dex文件的地址，并对其解析和加载：



　　执行流程：



　　上面表述了类加载流程，其实质在是在内存中得到类的ClassObejct ！其实这也是FindClass函数(FindClass还是调用findClassNoInit)需要去做的流程。所以说类加载和findclass都只是为了在dalvik中得到该类的ClasssObject。类说完了，但dalvik是执行代码，我怎么执行你的类方法呢？记得上面说的Method结构(ClassObject结构里指向它的指针，ClassObject里当然包括方法咯)，它才是dalvik的执行单元，ClassObject只是描述，类似于odex和dexFile吧。具体可以看：[android dalvik浅析一：解释器及其执行](http://www.cnblogs.com/vendanner/p/4784381.html" \t "http://www.cnblogs.com/vendanner/p/_blank)

void dvmCallMethodV(Thread\* self, const Method\* method, Object\* obj,

bool fromJni, JValue\* pResult, va\_list args)

 　　下面我们来看下在dalvik在加载过程中使用到的数据结构(Method、ClassObject的结构体见源码Object.h)：

　　Method:　　　　java方法在dalvik中的结构体，是dalvik的执行单位；拥有指向ClassObject的指针

　　ClassObject:　　java类在dalvik中的结构体，包含多个Method结构；拥有指向DvmDex的指针；拥有指向加载该类的ClassLoader指针

　　DvmDex:　　　 拥有指向dexFile的指针

　　dexFile:　　　　dex在dalvik的描述结构

 　　一目了然，我们可以通过Method追踪到odex文件。