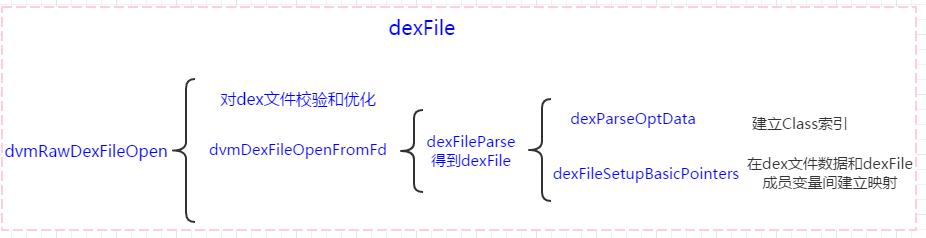
|  |  |  |
| --- | --- | --- |
| 名称 | 格式 | 描述 |
| header | DexHeader | 文件头 |
| string\_ids | DexStringId | 字符串索引表，记录了各个字符所在的偏移值。 |
| type\_ids | DexTypeId | 类型资源的索引信息 |
| proto\_ids | DexProtoId | 方法原型资源的索引信息 |
| field\_ids | DexFieldId | 代码中的字段的索引信息 |
| method\_ids | DexMothodId | Dex文件中类方法数据的索引信息 |
| class\_defs | DexClassDef | 类的定义和数据索引信息 |
| datas | ubyte | 数据区，保存着各个类的数据。 |
| Link\_data | ubyte | 静态连接数据。 |

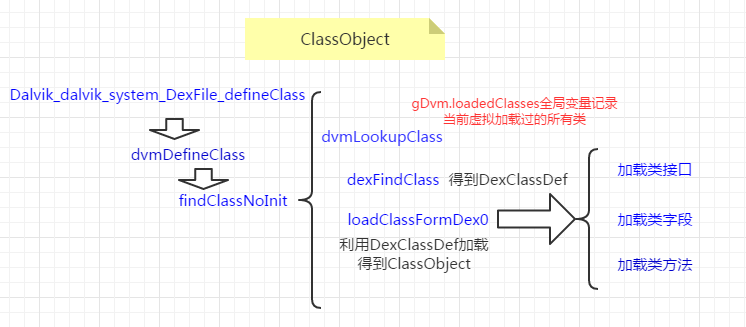
odex解析成dexFile结构体后，类加载器可以通过该结构体实现对Dex文件中各类数据的查找和获取(由此可知，dexFile和dex文件有对应关系)。看解析流程：



　　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。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。