* + 1. Dalvik类加载机制

对于Java虚拟机而言，将类class加载到虚拟机运行时环境的过程叫做类加载。类加载机制的主要功能是将应用程序中的操作码以及程序数据提取并加载到虚拟机内部，以保证程序的正确执行。具体到Dalvik虚拟机就是将Dex文件中的指令与数据提取出来，并加载到Dalvik虚拟机与一个ClassObject数据结构进行关联，实现在虚拟机中执行字节码。类加载具体流程如图【】所示：

* + - 1. Dex文件验证与优化

当虚拟机获取程序中的classes.dex文件后，首先将这个Dex文件进行初步的验证工作，主要包括：验证Magic，校验SHA-a签名，计算Dex校验和等几个方面，当校验完成后，则开始进行优化。Dex文件优化的目的是根据当前平台特性对程序中的字节码进行部分替换，并为Dex文件增加辅助信息以提高Dalvik运行效率，最后输出一个优化文件Odex，其结构如图[]所示。

为了更大程序的保证原dex文件的数据的安全性以及优化机制的独立性，优化机制并没有直接改写原Dex文件，而是从新创建了一个Odex文件并以严格的机制要求优化信息写入该文件，主要包括依赖库关系、寄存器映射关系以及类的索引关系，这些管理可以提高类加载机制的执行效率。同时在优化过程中还根据平台特性对原Dex文件进行部分字节码替换（如对字段的替换，把查找改为直接引用），最后将修改过后的Dex文件写入Odex文件中。

依赖库优化是指Dalvik虚拟机在程序执行前期通过优化机制将Dex的本地函数库整合到Odex文件中，可以在一定程度上提高程序的执行效率。类索引信息的建立是优化机制的重要工作之一，通过为Dex文件建立一个哈希表DexClassLookup，其中的每一项为table结构体对象，记录了一个类的描述符哈希值、类描述符在Dex文件中的偏移以及类定义区的偏移地址，使得类加载机制可以很快的定位类资源地址并完成类加载。DexClassLookup中数据结构定义如表[]所示：

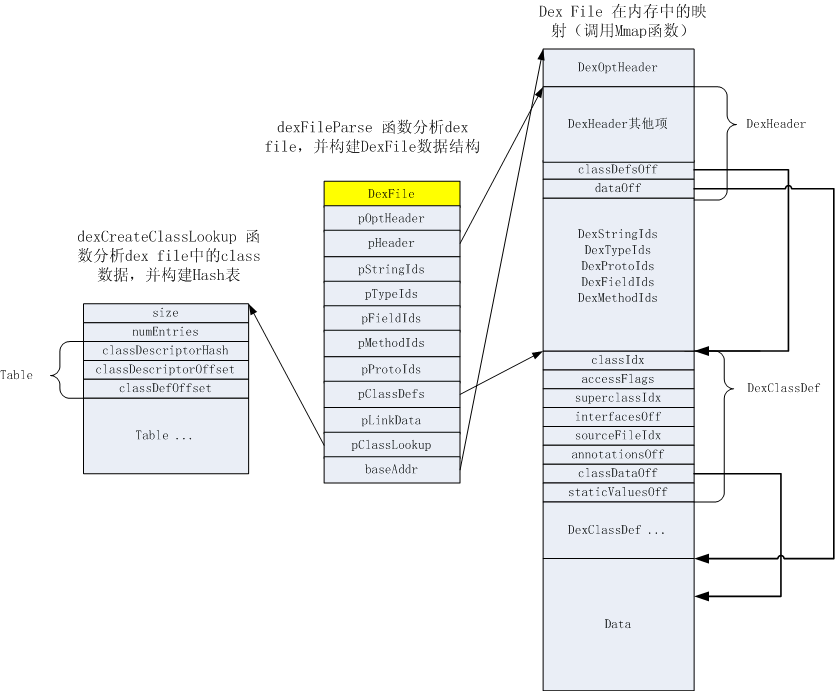
|  |  |  |
| --- | --- | --- |
| 字段名 | 字段类型类型 | 描述 |
| size | int | 表中table的个数(对应类个数) |
| numEntries | int | 表项入口地址 |
| classDescriptorHash | u4 | 类描述符哈希值 |
| classDescriptorOffset | u4 | 类描述符的偏移值 |
| classDefOffset | u4 | 类定义偏移值 |

Android设备在应用程序第一次启动的时候往往耗费较长的时间，在这期间虚拟机对目标Dex文件行了验证与优化并为之生成了Odex文件。当用户再次启动的时候，Odex文件会代替原有的Dex文件被虚拟机引用执行，不在需要进行优化，因而启动时间就要短很多。

* + - 1. Dex文件与解析

Dex文件与由多个Class文件整合而成，这是的Dalvik在加载一个目标类之前需要对Dex文件进行一系列的处理。Dex文件解析的主要目的就是对Dex文件进行读取分析，并建立一个DexFile结构体实例来描述该Dex文件，使得类加载函数可以通过该结构对目标类进行全部的数据进行索引并提取，完成类的实际加载工作。

Odex文件在解析过程中比较快的原因就是可以把类索引信息、依赖库信息、寄存器关系信息等提前与DexFile进行关联。完成优化数据的处理之后，将对写入Odex文件的Dex数据进行解析，将DexFile数据结构中各个成员变量与Dex文件的各个数据部分进行关联，是的虚拟机能够更加高效的对Dex文件中的数据进行查找并获取，关联效果如图[]所示。



* + - 1. 加载类数据

类加载机制的根本任务是根据运行程序的需要在已被虚拟机解析的Dex文件中查找并加载指定类。