第五讲

Android程序的逆向分析

鲁宏伟

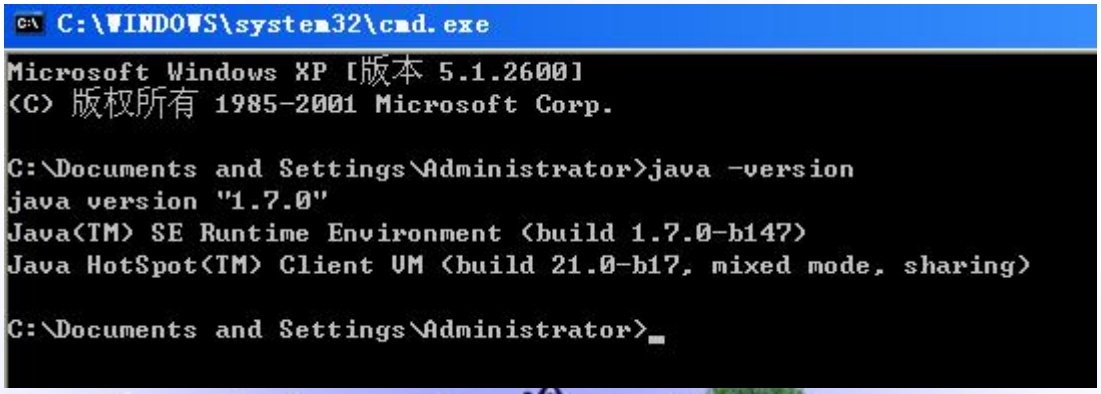
luhw@hust.edu.cn

Android程序分析环境搭建

- 安装JDK
- 安装Android SDK
- 安装Android NDK
- Eclipse集成开发环境
- 安装CDT、ADT插件
- 创建Android Virtual Device
- 使用到的工具

安装JDK

- Android开发语言使用的是Java，所以我们要安装JDK（Java Development Kit）Java 开发工具包
- 安装好了JDK后，，在dos下输入java -version，如图显示，表示我们安装正确，这只是表明成功安装了JDK，但是能不能编译java文件还要靠后面的系统环境变量的配置



安装Android SDK

- Android SDK 指的是Android专属的软件开发工具包
- Android SDK不用安装，下载后，直接解压即可， 将下载后的SDK的压缩包解压到适当的位置
- 安装成功后需要将相关的目录(包含“\android-sdk\tools”和“\android-sdk\platform-tools”)添加到系统的PATH环境变量中
- 安装成功后需要通过SDK管理器下载具体版本的SDK，可以根据自己的需要下载一个或多个版本下载

安装Android NDK

- NDK提供了一系列的工具，帮助开发者快速开发C（或C++）的动态库，并能自动将so和java应用一起打包成apk
- NDK集成了交叉编译器，并提供了相应的mk文件隔离平台、CPU、API等差异，开发人员只需要简单修改mk文件（指出“哪些文件需要编译”、“编译特性要求”等），就可以创建出so

Eclipse集成开发环境

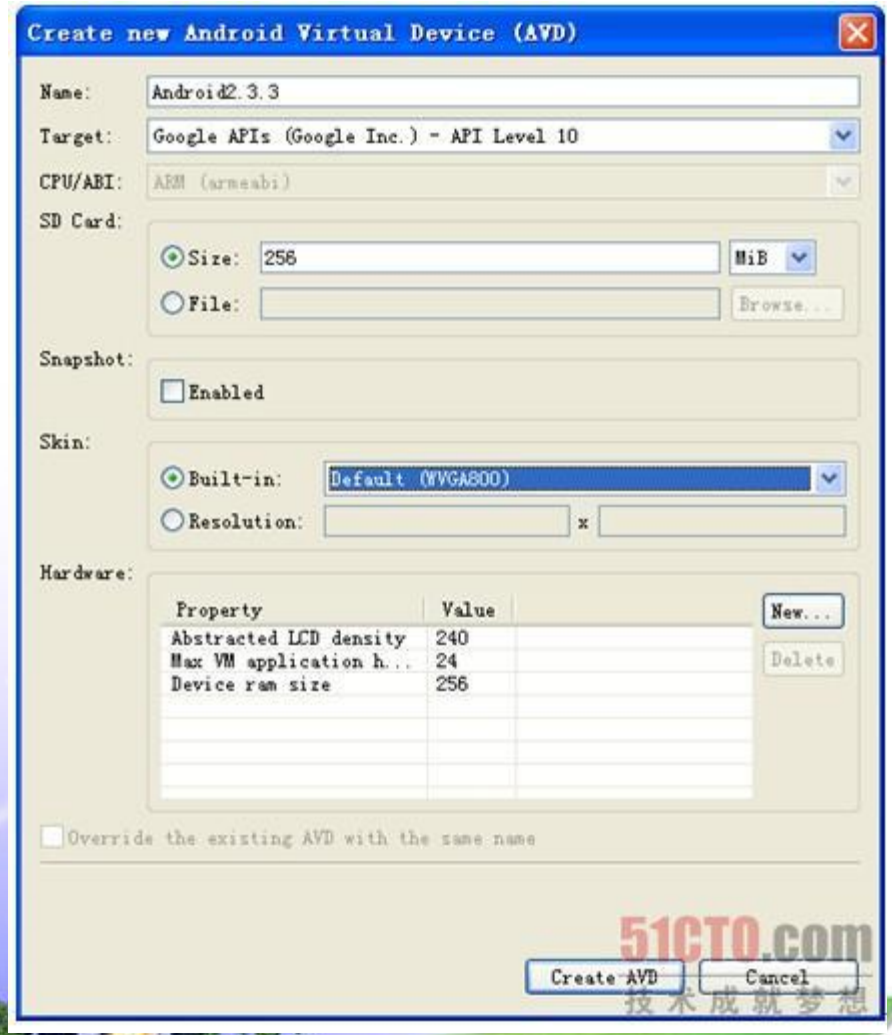
- Eclipse 是一个开放源代码的、基于Java的可扩展开发平台。就其本身而言，它只是一个框架和一组服务，用于通过插件组件构建开发环境。
- Eclipse的插件机制是轻型软件组件化架构。在客户机平台上，Eclipse使用插件来提供所有的附加功能，例如支持Java以外的其他语言。
- 已有的分离的插件已经能够支持C/C++（CDT）、Perl、Ruby，Python、telnet和数据库开发。

安装CDT、ADT插件

- 如果使用的Eclipse是For Mobile Developers版本或自带CDT插件，可以跳过CDT插件的安装；否则需要手动安装CDT插件。
- ADT插件是Google为Android开发提供的Eclipse插件，方便在Eclipse开发环境中创建、编辑、调试Android程序。

创建Android Virtual Device

- Android SDK中提供了"Android Virtual Device Manager"工具，方便在没有真实Android设备环境的情况下调试运行Android程序。
- 双击运行“D:\android-sdk-windows\ AVD Manager.exe”，，结果如图所示。



相关工具

- 分析Android程序需要用到很多工具，包括：
  - ✓ **反编译工具**
  - ✓ **静态分析工具**
  - ✓ **动态调试工具**
  - ✓ **集成分析环境等。**
- 所有工具中，大多数是开源或者免费的软件。

如何分析Android程序

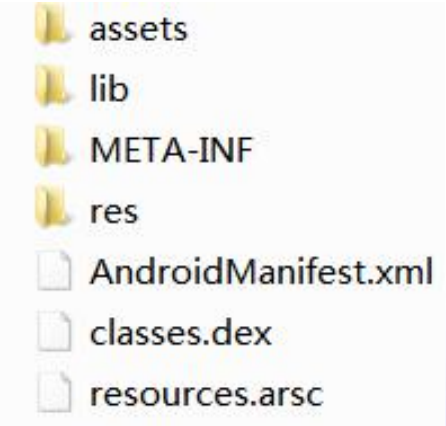
- 如何动手?
- 反编译APK文件
- 分析APK文件
- 修改Smali文件代码
- 重新编译APK文件并签名
- 安装测试

如何动手？

- 将apk文件利用ApkTool反编译，生成smali格式的反汇编代码，然后阅读smali文件的代码来理解程序的运行机制，找到程序的突破口进行修改，最后生成使用ApkTool 重新编译生成apk文件并签名，最后进行测试，如此循环，直至程序被成功破解
- 在实际分析过程中，借助dex2jar和jd-gui等工具配合来进行Java源码级的分析。

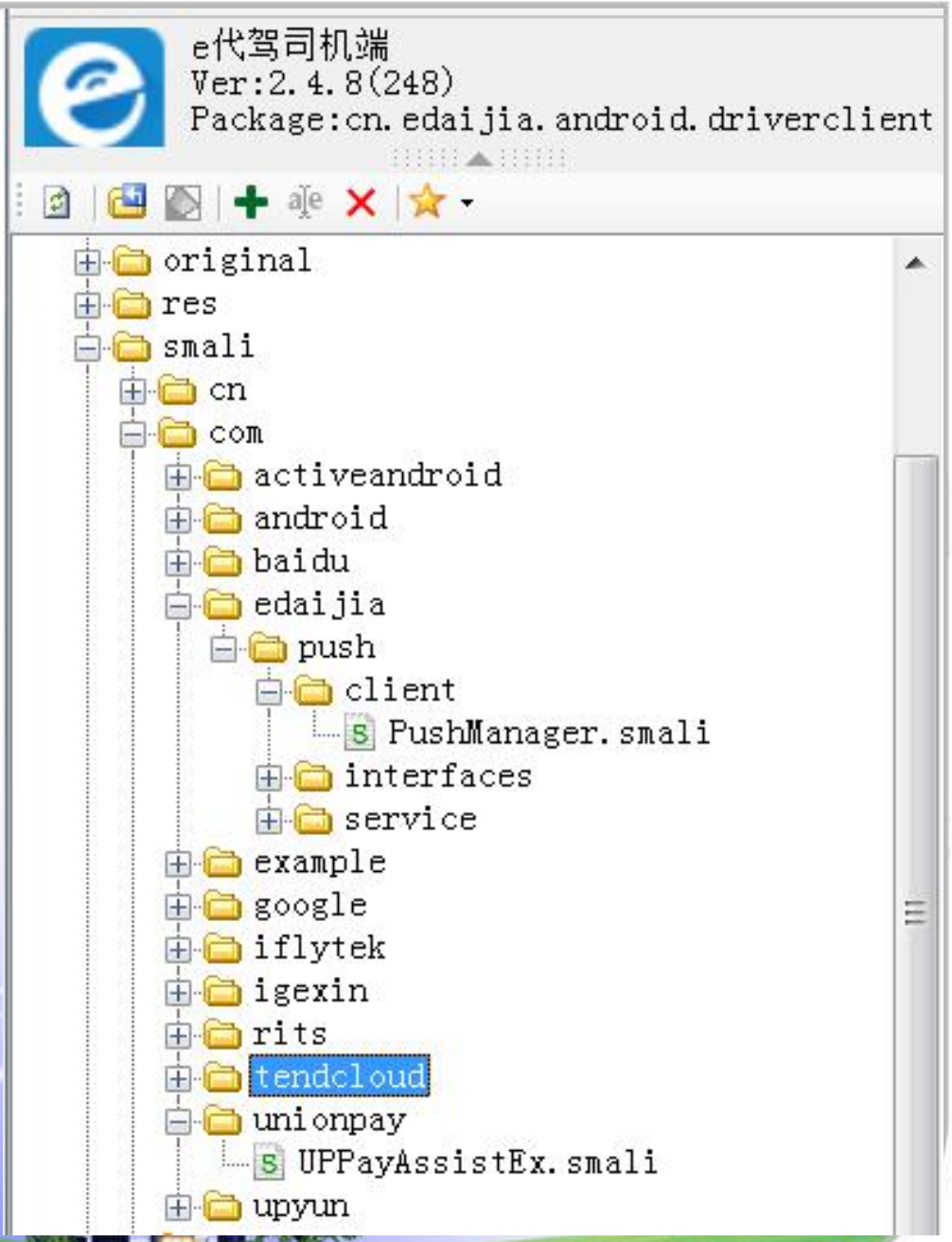
反编译APK文件

- APK文件其实就是一个ZIP格式的压缩包，其核心是dex文件和lib目录下的so文件。



- 利用ApkTool反编译，生成smali格式
- 利用dex2jar可以将dex文件转换成jar文件

反编译后的结果


e代驾司机端
Ver:2.4.8(248)
Package:cn.edaijia.android.driverclient
original
res
smali
cn
com
activeandroid
android
baidu
edaijia
push
client
PushManager.smali
interfaces
service
example
google
iflytek
igexin
rits
tendcloud
unionpay
UPPayAssistEx.smali
upyun

cn.edaijia.android.driverclient

- cn.edaijia.android
- com
  - activeandroid
  - android.internal.telephony
  - baidu
  - edaijia.push
    - client
      - PushManager
    - interfaces
    - service
  - example.android_base
  - google
  - iflytek
    - a
    - b
    - c
    - msc
    - record
    - speech
    - ui
      - a
      - b
      - RecognizerDialog
      - RecognizerDialogListener
      - SynthesizerDialog
      - SynthesizerDialogListener
      - UploadDialog
      - a
      - b
      - c
      - d
      - e
      - f
      - g
      - h
      - i
      - j
    - util
    - Setting
    - Version

PushManager.class

```
package com.edaijia.push.client;

import android.app.Application;

public enum PushManager
  implements ServiceConnection
{
  public static final String ACTION_CLIENT_ID = "cn.edaijia.android.push.client_id";
  public static final String ACTION_PAUSE = "cn.edaijia.android.push.pause";
  public static final String ACTION_PUSH = "cn.edaijia.android.push.push";
  public static final String ACTION_RESUME = "cn.edaijia.android.push.resume";
  public static final String ACTION_START = "cn.edaijia.android.push.start";
  public static final String ACTION_STOP = "cn.edaijia.android.push.stop";
  public static final String PARAM_CLIENT_ID = "client_id";
  public static final String PARAM_DATA = "data";
  public static final String PARAM_MSG_ID = "msg_id";
  public static final String SP_FILE = "push";
  public static final String SP_KEY_STATUS = "status";
  public static Application sApp;
  public static String sAppId;
  public static String sAppVer;
  static SharedPreferences sSp;
  PushCommand mPushCommand = null;

  static
  {
    PushManager[] arrayOfPushManager = new PushManager[1];
    arrayOfPushManager[0] = INSTANCE;
    $VALUES = arrayOfPushManager;
  }

  private PushManager() {}

  public static boolean getStatusFromSp()
  {
    return sSp.getBoolean("status", true);
  }

  public static void init(Application paramApplication, String paramString1, String
  {
    sApp = paramApplication;
    sSp = paramApplication.getSharedPreferences("push", 0);
    if (getStatusFromSp()) {
      INSTANCE.start();
    }
    sAppId = paramString1;
    sAppVer = paramString2;
  }
```

分析APK文件

- 反编译apk 文件成功后，会在当前的outdir 目录下生成一系列目录与文件。其中 smali目录下存放了程序所有的反汇编代码，res 目录则是程序中所有的资源文件，这些目录的子目录和文件与开发时的源码目录组织结构是一致的
- 如何寻找突破口是分析一个程序的关键。对于一般的Android来说，错误提示信息通常是指引关键代码的风向标，在错误提示附近一般是程序的核心验证代码，分析人员需要阅读这些代码来理解软件的注册流程。

分析APK文件

- 错误提示是Android程序中的字符串资源，开发 Android程序时，这些字符串可能硬编码到源码中，也可能引用自“res\values ”目录下的strings.xml 文件，apk 文件在打包时，strings.xml 中的字符串被加密存储为resources.arsc 文件保存到apk 程序包中，apk 被成功反编译后这个文件也被解密出来了。
- 开发Android 程序时，String.xml 文件中的所有字符串资源都在“gen/<packagename>/ R.java”文件的 String 类中被标识，每个字符串都有唯一的 int 类型索引值，使用Apktool反编译apk 文件后，所有的索引值保存在 string.xml 文件同目录下的public.xml 文件中。

修改Smali文件代码

- 用任意一款文本编辑器打开相应的smali 文件，根据需要修改代码，保存后退出，代码就算修改完成了。

重新编译APK文件并签名

- 修改完Smali 文件代码后，需要将修改后的文件重新进行编译打包成 apk 文件。
- 编译 apk 文件的命令格式为
- **apktool b[uild] [OPTS] [<app_path>] [<out_file>]**
- 编译生成的apk没有签名，还不能安装测试，接下来需要使用signapk.jar 工具对apk 文件进行签名。
- signapk.jar 是Android源码包中的一个签名工具。代码位于 Android源码目录下的 /build/tools/signapk/SignApk.java 文件中

安装测试

- 启动Android AVD，或者使用数据线连接手机与电脑，然后在命令提示符下执行以下命令安装破解后的程序：
  adb install signed.apk
- 不出意外会得到以下输出提示：

```
adb install signed.apk
822 KB/s (39472 bytes in 0.046s)
        pkg: /data/local/tmp/signed.apk
Success
```

安装测试

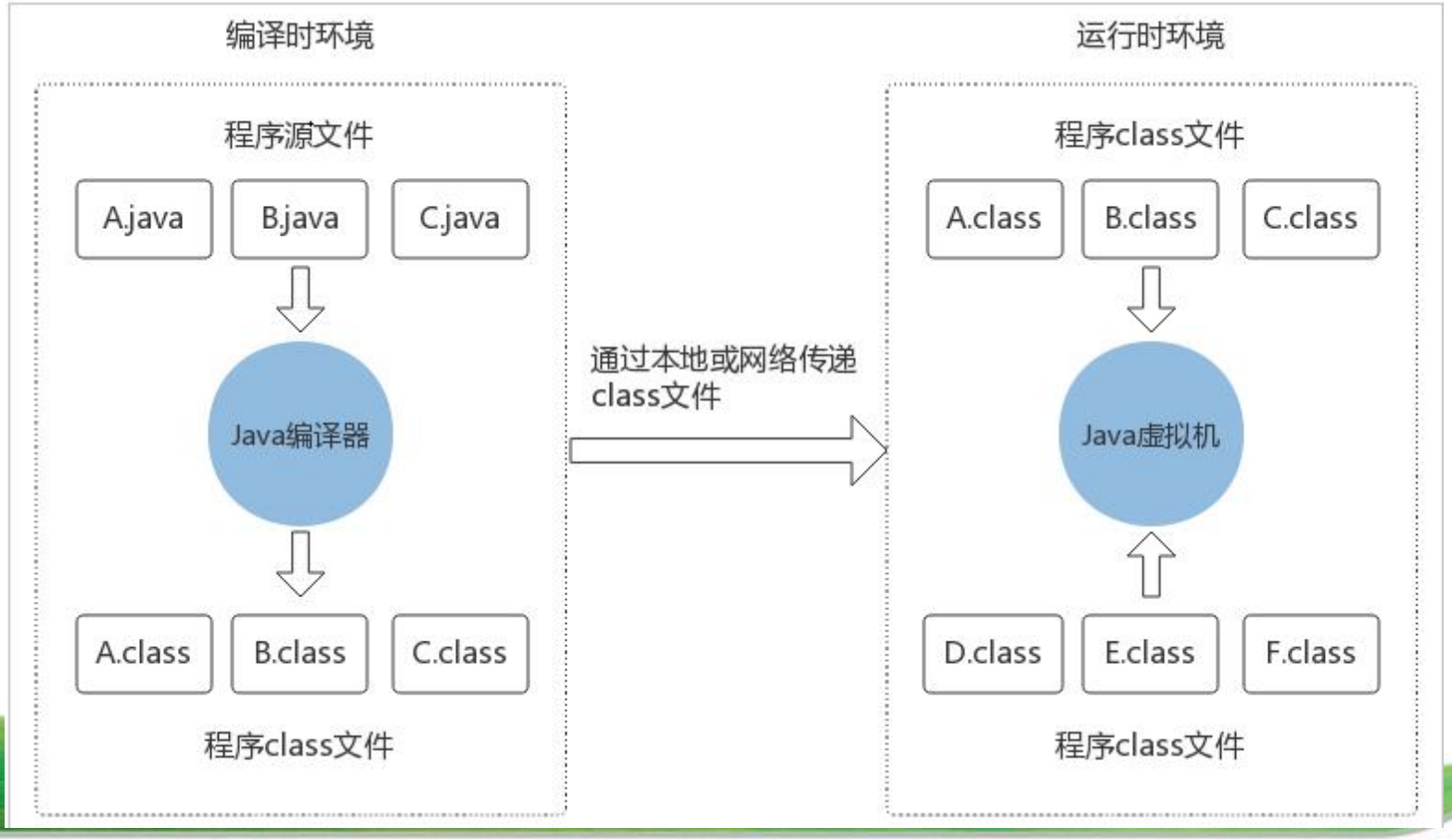
- Android Debug Bridge（adb）：该工具可将其他工具接入模拟器和设备，它除了可以让其他工具还可以使命令行上传或下载文件，安装或卸载程序包，通过进入设备或模拟器的shell环境访问许多其他功能

理解Java虚拟机体系结构

- 众所周知，Java支持平台无关性、安全性和网络移动性。而Java平台由Java虚拟机和Java核心类所构成，它为纯Java程序提供了统一的编程接口，而不管下层操作系统是什么。正是得益于Java虚拟机，它号称的“一次编译，到处运行”才能有所保障。

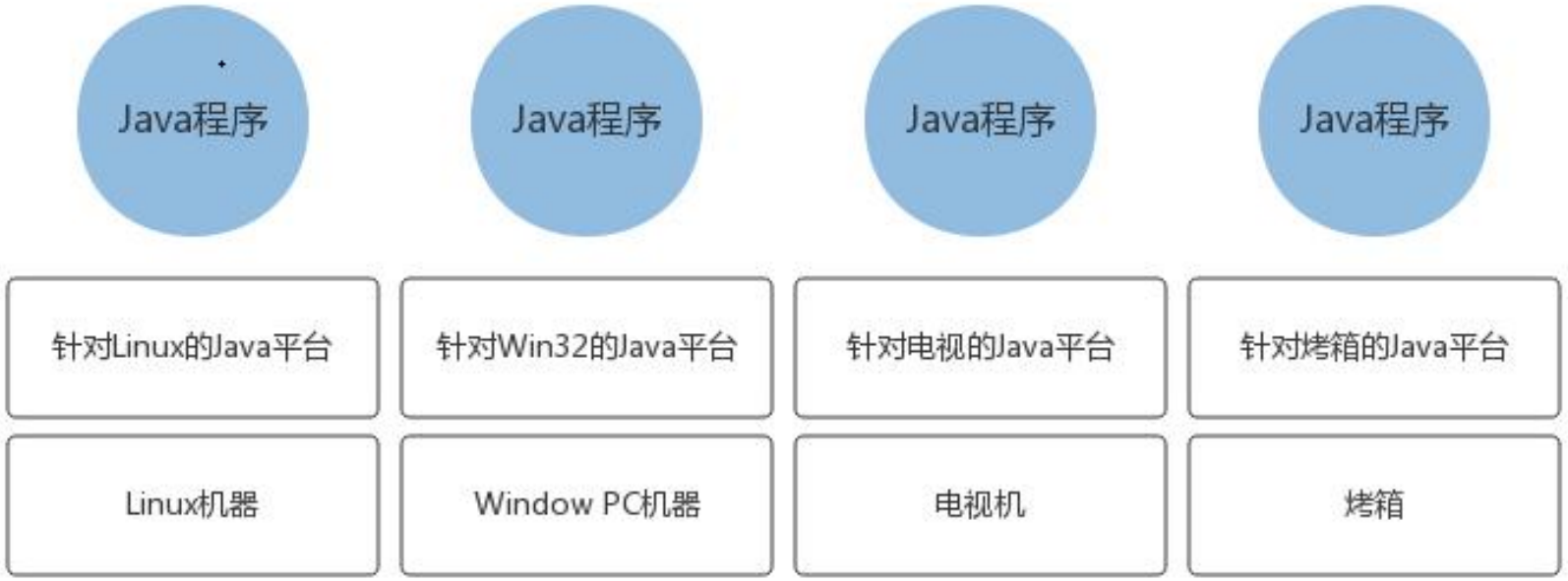
Java程序执行流程

- Java程序的执行依赖于编译环境和运行环境。源码代码转变成可执行的机器代码，由下面的流程完成



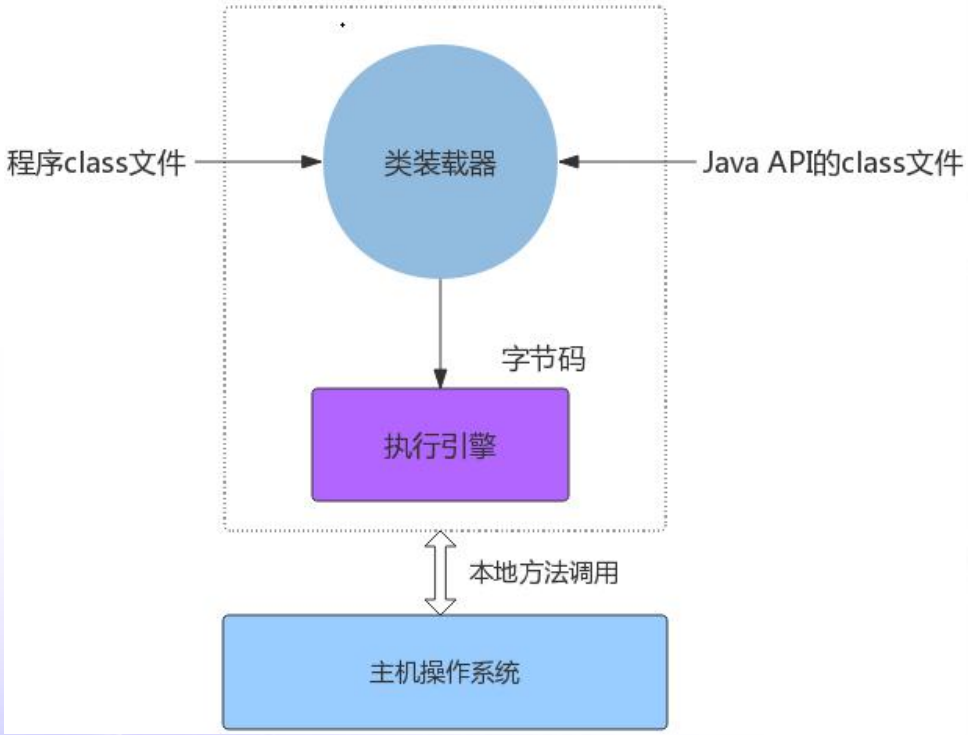
Java程序执行流程

- Java技术的核心就是Java虚拟机，因为所有的Java程序都在虚拟机上运行。当启动一个Java程序时，一个虚拟机实例就诞生了。当程序结束，这个虚拟机实例也就消亡



Java虚拟机

- Java虚拟机的主要任务是装载class文件并且执行其中的字节码。由右图可以看出，Java虚拟机包含一个类装载器（class loader），它可以从程序和API中装载class文件，Java API中只有程序执行时需要的类才会被装载，字节码由执行引擎来执行

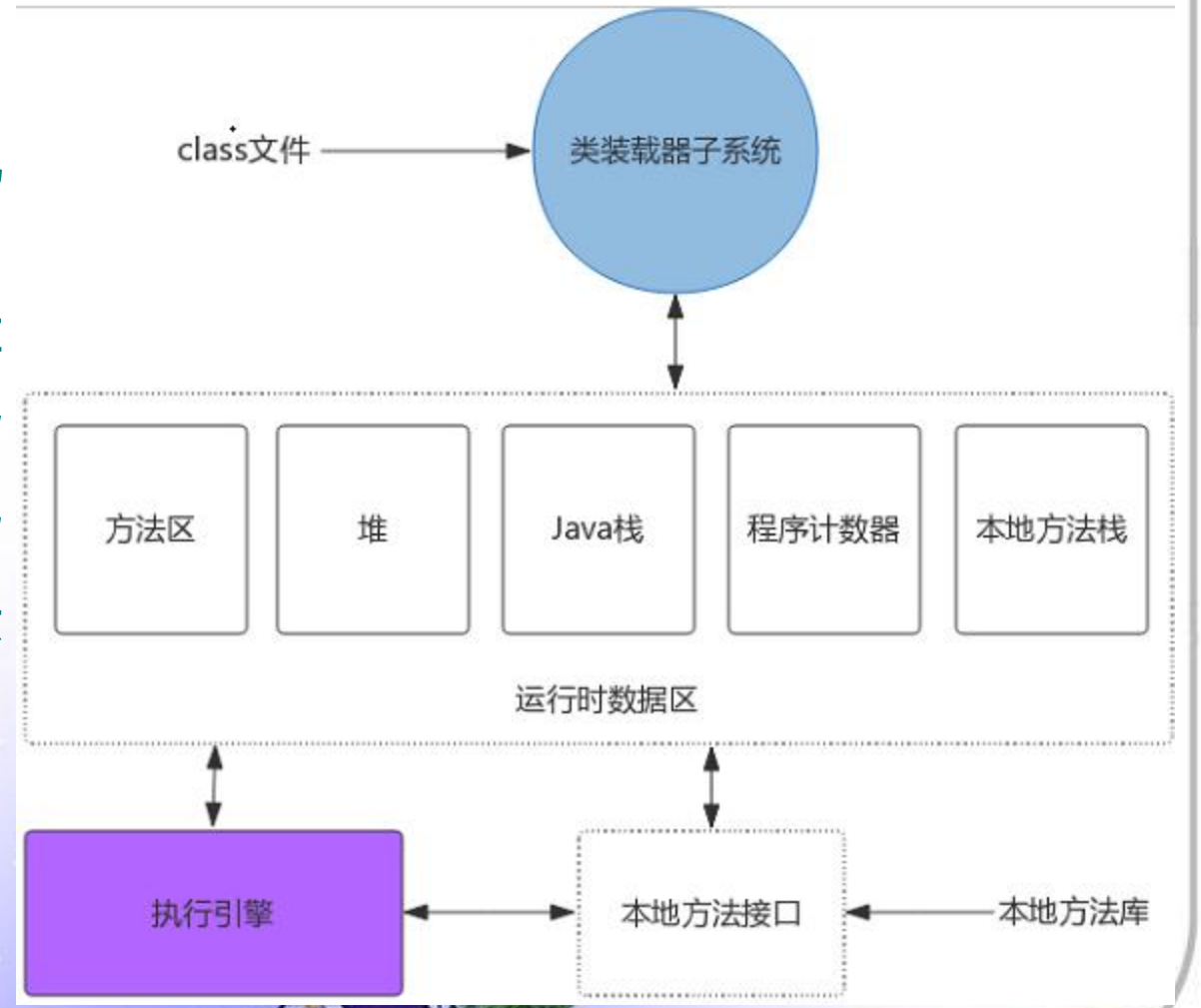


Java虚拟机

- 当Java虚拟机由主机操作系统上的软件实现时，Java程序通过调用本地方法和主机进行交互。
- Java方法由Java语言编写，编译成字节码，存储在class文件中。
- 本地方法由C/C++/汇编语言编写，编译成和处理器相关的机器代码，存储在动态链接库中，格式是各个平台专有。所以本地方法是联系Java程序和底层主机操作系统的连接方式。

Java虚拟机体系结构

- 在 Java虚拟机规范中，一个虚拟机实例的行为是分别按照子系统、内存区、数据类型和指令来描述的，这些组成部分一起展示了抽象的虚拟机的内部体系结构。



Java虚拟机体系结构-class文件

- class文件包含的内容：

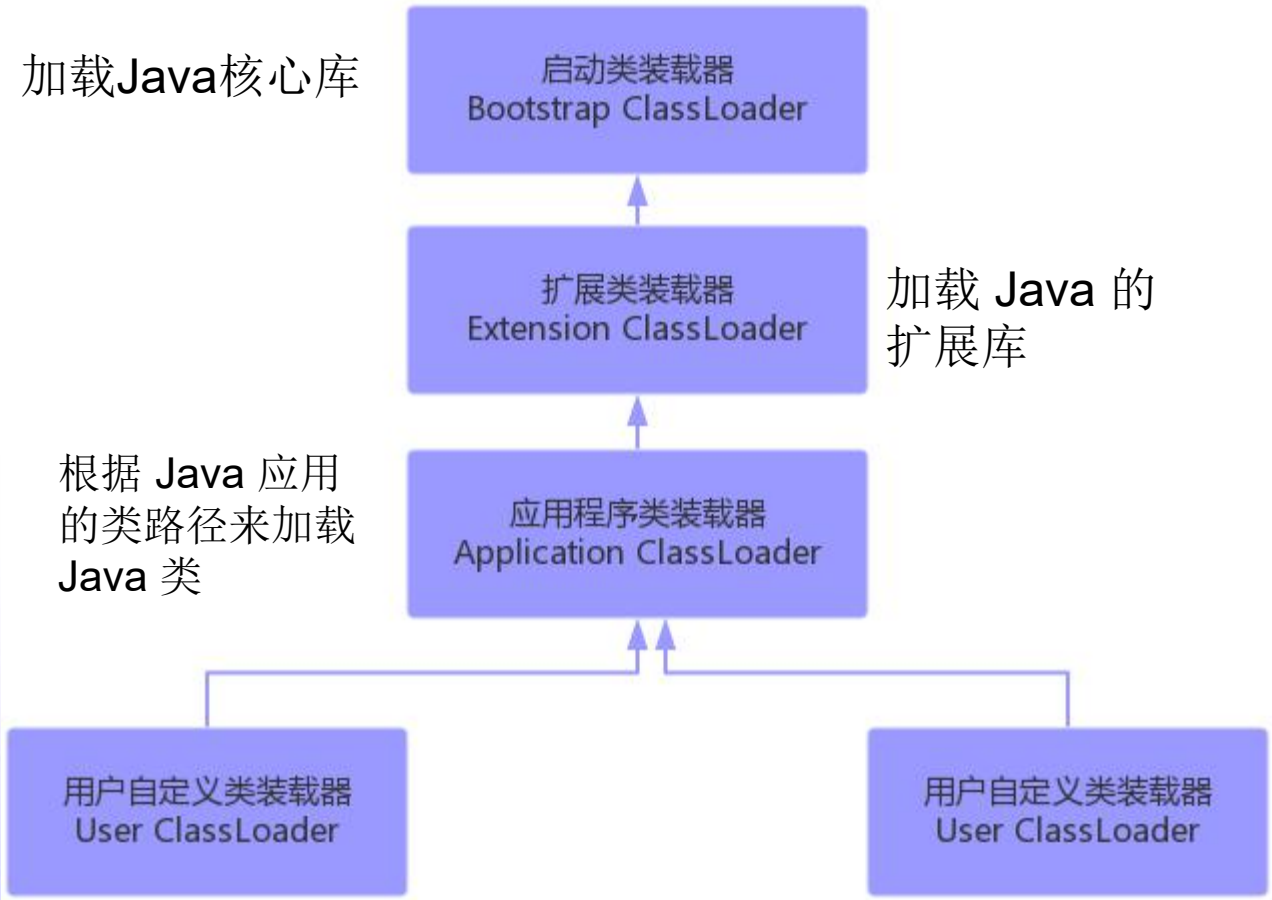
```
ClassFile {

    u4 magic;                                   //魔数：0xCAFEBABE，用来判断是否是Java class文件
    u2 minor_version;                           //次版本号
    u2 major_version;                           //主版本号
    u2 constant_pool_count;                     //常量池大小
    cp_info constant_pool[constant_pool_count-1]; //常量池
    u2 access_flags;                            //类和接口层次的访问标志（通过|运算得到）
    u2 this_class;                              //类索引（指向常量池中的类常量）
    u2 super_class;                             //父类索引（指向常量池中的类常量）
    u2 interfaces_count;                        //接口索引计数器
    u2 interfaces[interfaces_count];            //接口索引集合
    u2 fields_count;                            //字段数量计数器
    field_info fields[fields_count];            //字段表集合
    u2 methods_count;                           //方法数量计数器
    method_info methods[methods_count];         //方法表集合
    u2 attributes_count;                        //属性个数
    attribute_info attributes[attributes_count];  //属性表

}
```

Java虚拟机体系结构-类装载器子系统

- 类装载器子系统负责查找并装载类型信息。其实Java虚拟机有两种类装载器：系统装载器和用户自定义装载器。前者是Java虚拟机实现的一部分，后者则是Java程序的一部分

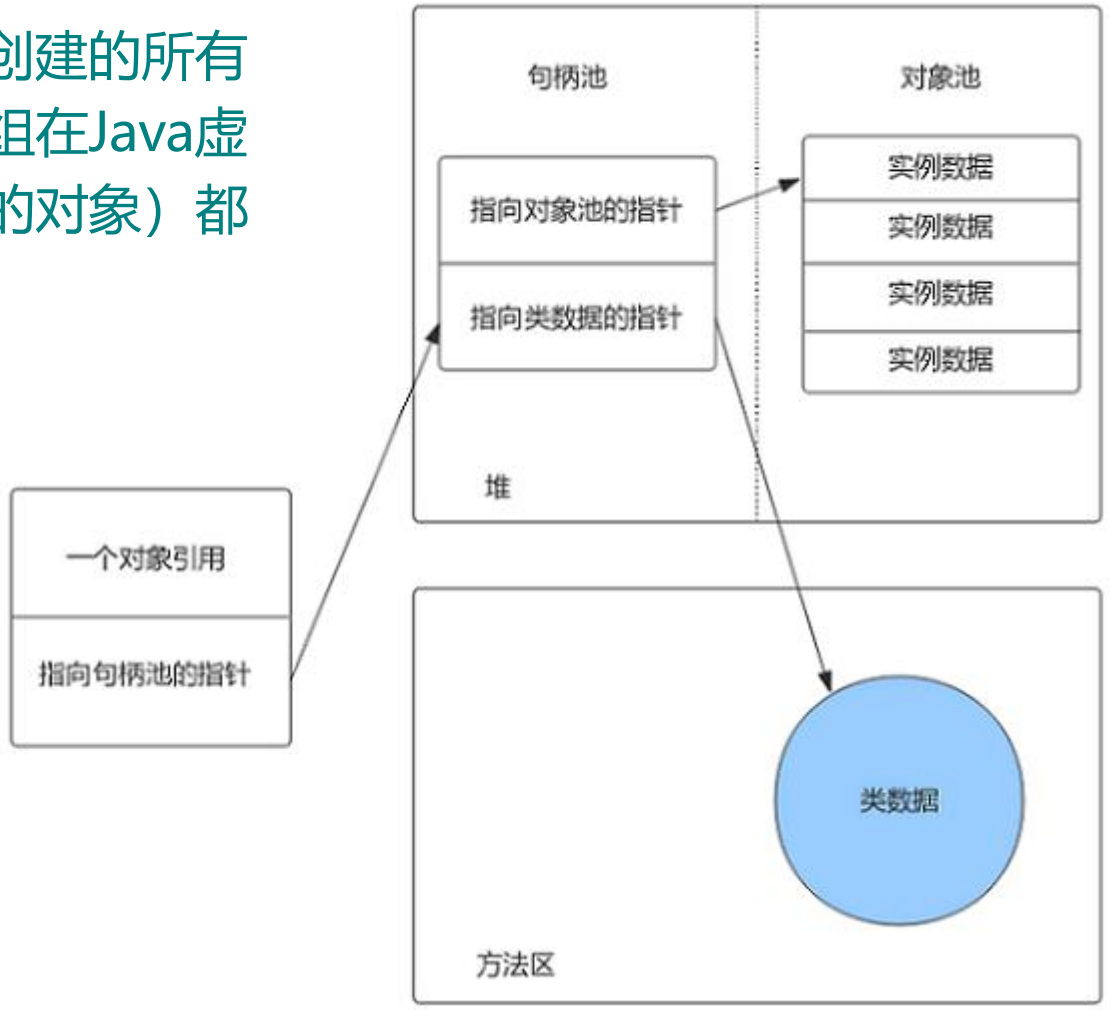


Java虚拟机体系结构-方法区

- 在Java虚拟机中，关于被装载的类型信息存储在一个方法区的内存中。
- 当虚拟机装载某个类型时，它使用类装载器定位相应的class文件，然后读入这个class文件并将它传输到虚拟机中，接着虚拟机提取其中的类型信息，并将这些信息存储到方法区。
- 方法区也可以被垃圾回收器收集，因为虚拟机允许通过用户定义的类装载器来动态扩展Java程序。

Java虚拟机体系结构-堆

- Java程序在运行时创建的所有类实例或数组（数组在Java虚拟机中是一个真正的对象）都放在同一个堆中
- 需要注意的是，Java虚拟机有一条在堆中分配对象的指令，却没有释放内存的指令，因为虚拟机把这个任务交给垃圾收集器处理



Java虚拟机体系结构-Java栈

- 每当启动给一个线程时，Java虚拟机会为它分配一个Java栈。
- Java栈由许多栈帧组成，一个栈帧包含一个Java方法调用的状态。
- 当线程调用一个Java方法时，虚拟机压入一个新的栈帧到该线程的Java栈中，当该方法返回时，这个栈帧就从Java栈中弹出。
- Java栈存储线程中Java方法调用的状态--包括局部变量、参数、返回值以及运算的中间结果等。
- Java虚拟机没有寄存器，其指令集使用Java栈来存储中间数据。这样设计的原因是为了保持Java虚拟机的指令集尽量紧凑，同时也便于Java虚拟机在只有很少通用寄存器的平台上实现。
- 另外，基于栈的体系结构，也有助于运行时某些虚拟机实现的动态编译器和即时编译器的代码优化。

Dalvik和ART

- Android系统是以Linux为内核构建的。
- Google为了降低应用的开发难度，并将其适配到不同硬件配置的设备上，在Linux内核之上构建了一个虚拟机，Android应用使用 Java开发，运行在虚拟机之上。

Dalvik和ART

- Dalvik就是Android4.4及之前使用的虚拟机，它使用的是JIT（Just-In-Time）技术来进行代码转译，每次执行应用的时候，Dalvik将程序的代码编译为机器语言执行。
- 随着硬件水平的不断发展以及人们对更高性能的需求，Dalvik虚拟机的不足日益突出。
- 应运而生的ART(Android RunTime)虚拟机，其处理机制根本上的区别是：它采用AOT(Ahead-Of-Time)技术，会在应用程序安装时就转换成机器语言，不再在执行时解释，从而优化了应用运行的速度。
- 在内存管理方面，ART也有比较大的改进，对内存分配和回收都做了算法优化，降低了内存碎片化程度，回收时间也得以缩短。

Dalvik虚拟机

- Java虚拟机运行的是Java字节码，而Dalvik虚拟机运行的则是其专有的文件格式DEX。
- 在Java SE程序中的Java类会被编译成一个或者多个字节码文件（.class）然后打包到JAR文件，而后Java虚拟机会从相应的CLASS文件和JAR文件中获取相应的字节码；Android应用虽然也是使用Java语言进行编程，但是在编译成CLASS文件后，还会通过一个工具（dx）将所有的CLASS文件转换成一个.dex (Dalvik Executable)文件。
- .dex格式是专为Dalvik设计的一种压缩格式，适合内存和处理器速度有限的系统。

Dalvik虚拟机

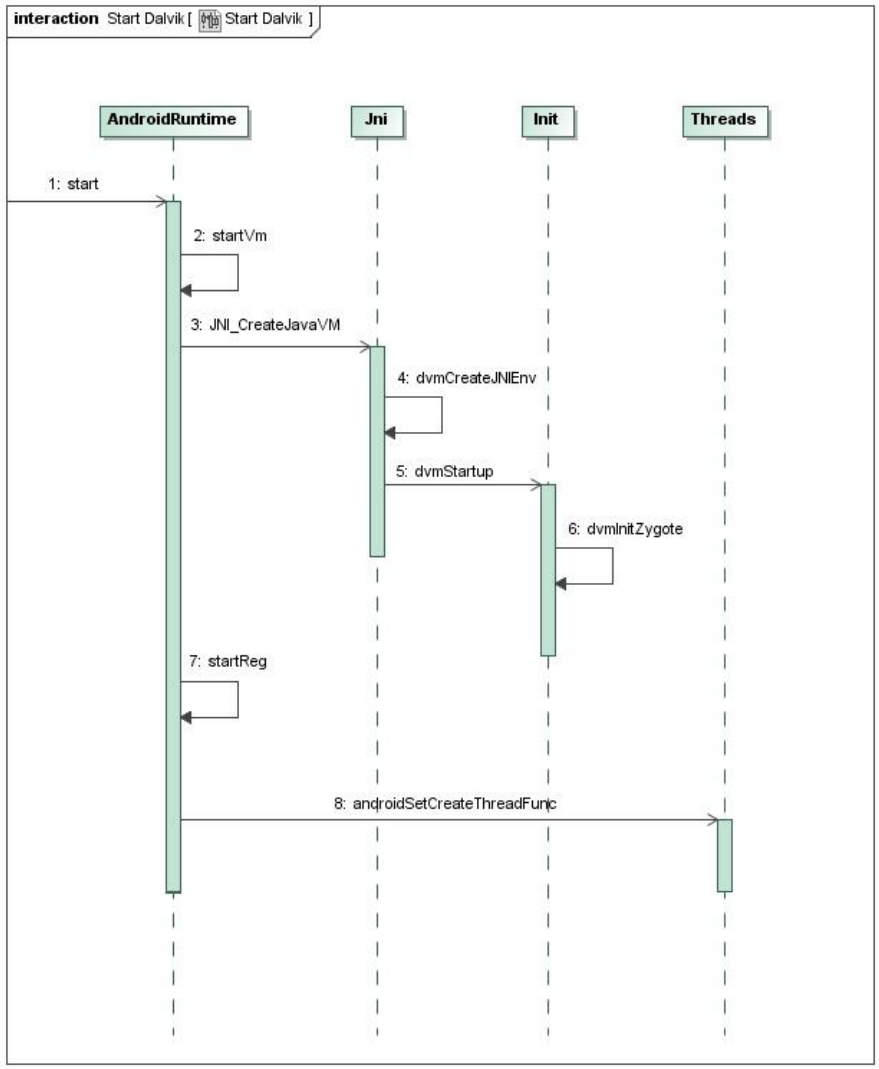
- Dalvik经过优化，允许在有限的内存中同时运行多个虚拟机的实例，并且每一个Dalvik应用作为一个独立的Linux 进程执行。独立的进程可以防止在虚拟机崩溃的时候所有程序都被关闭。
- 相对于一般Java虚拟机（JVM），Dalvik虚拟机中的可执行文件体积更小；
- Dalvik基于寄存器，而JVM基于栈

Dalvik虚拟机是如何执行程序的

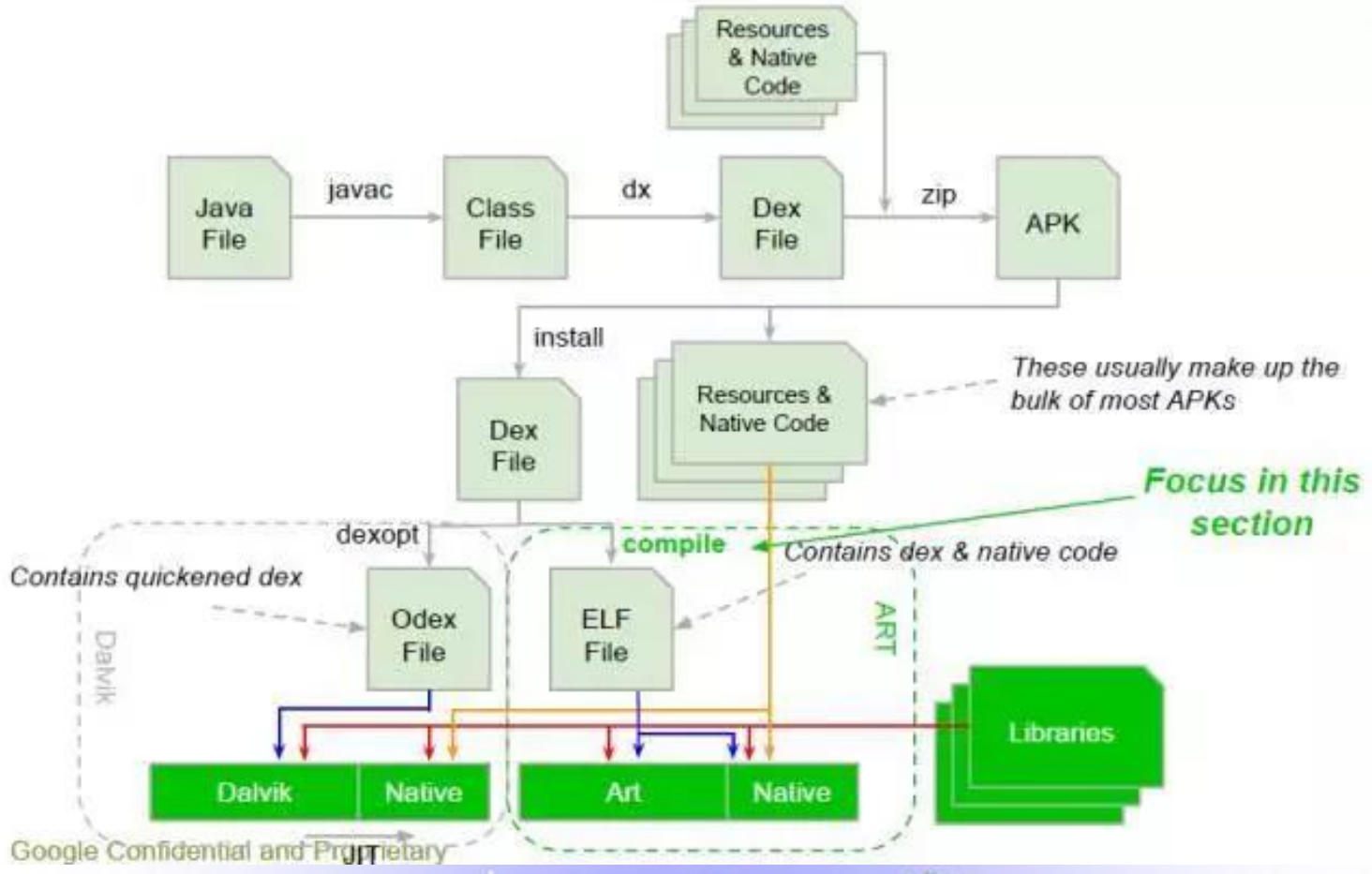
- 每一个Android应用都运行在一个Dalvik虚拟机实例里，而每一个虚拟机实例都是一个独立的进程空间。
- Zygote是虚拟机实例的孵化器。每当系统要求执行一个Android应用程序，Zygote就会fork出一个子进程来执行该应用程序。
- 这样做的好处显而易见：Zygote进程是在系统启动时产生的，它会完成虚拟机的初始化、库的加载、预置类库的加载和初始化等等操作，而在系统需要一个新的虚拟机实例时，Zygote通过复制自身，最快速的提供一个系统。另外，对于一些只读的系统库，所有虚拟机实例都和Zygote共享一块内存区域，大大节省了内存开销。

Dalvik虚拟机是如何执行程序的

Dalvik虚拟机在Zygote进程中启动完成之后，就会获得一个JavaVM实例和一个JNIEnv实例。其中，获得的JavaVM实例就是用来描述Zygote进程的Dalvik虚拟机实例，而获得的JNIEnv实例描述的是Zygote进程的主线程的JNI环境。



Android对apk的执行流程



Android对apk的执行流程

- Java文件在编译成class文件，然后经过Android平台的dx工具转换为Dex文件后，同Native code（JNI）和资源一起打包成apk，apk安装到手机后解压出Dex文件。
- Dalvik会通过dexopt工具将Dex优化，成为Odex文件，Odex文件的效率比Dex高，但其中大部分代码仍然需要每次执行时编译；
- 而ART则会将Dex通过dex2oat工具编译得到一个ELF文件，它是一个可执行的文件。

Android对apk的执行流程

- 在执行这段Java代码时，Dalvik虚拟机先要把test()方法的每句代码转译成Dex代码，对其中的① ② 两句赋值语句，执行时需要在虚拟机中进行“指令读取—识别指令—跳转—实例操作”的解析过程；
- 而ART中Java代码都被以方法为单位编译成汇编指令，执行上面这个方法的时候，① ② 两句代码只需要直接拷贝两个寄存器的值，各需要一条汇编指令就可以完成，省去了跳转、指令读取的过程，执行效率也就大大提高了。

```
int a = 1;
int b = 2;
public int test() {
    int x = a; //①
    int y = b; //②
    int z = a + b;
    return z;
}
```

smali文件

- 我们用工具反编译一些App的时候，会看到一个smali文件夹，里面其实就是每个Java类所对应的smali文件。
- Android虚拟机Dalvik并不是执行Java虚拟机JVM编译后生成的class文件，而是执行再重新整合打包后生成的dex文件，dex文件反编译之后就是smali代码，可以说，smali语言是Dalvik的反汇编语言

smali文件

- Smali 语言起初是由一个名叫 JesusFreke 的 hacker 对 Dalvik 字节码的翻译，并非一种官方标准语言，因为 Dalvik 虚拟机名字来源于冰岛一个小渔村的名字，JesusFreke 便把 smali和 baksmali 取自了冰岛语中的“汇编器”和“反编器”。
- 目前 Smali 是在 Google Code 上的一个开源项目。虽然主流的 DEX 可执行文件反汇编工具不少，如 Dedexer、IDA Pro 和 dex2jar+jd-gui，但 Smali 提供反汇编功能的同时，也提供了打包反汇编代码重新生成 dex 的功能，因此 Smali被广泛地用于 App 广告注入、汉化和破解，ROM 定制等方面。

smali文件

- Java和Smali数据类型对比

JAVA	Smali
V	void
Z	boolean
B	byte
S	short
C	char
I	int
J	long
F	float
D	double

引用类型

L	对象
[	数组

Java代码

```
private boolean show(){
        boolean tempFlag = ((3-2)==1)? true : false;
        if (tempFlag) {
            return true;
        }else{
            return false;
        }
    }
```

转换smali代码

```
.method private show()Z
    .locals 2

    .prologue            //方法开始
    .line 22
    const/4 v0, 0x1      // v0赋值为1

    .line 24
    .local v0, tempFlag:Z
// 判断v0是否等于0，不符合条件向下走，符合条件执行cond_0分支
    if-eqz v0, :cond_0

    .line 25
    const/4 v1, 0x1      // 符合条件分支

    .line 27
    :goto_0
    return v1

    :cond_0
    const/4 v1, 0x0  // cond_0分支

    goto :goto_0
.end method
```

smali文件

- 每个 Dalvik 寄存器都是 32 位大小，对于小于或者等于 32 位长度的类型来说，一个寄存器就可以存放该类型的值，而像 J、D 等 64 位的类型，它们的值是使用相邻两个寄存器来存储的，如 v0 与 v1、v3 与 v4 等。
- Java 中的对象在 smali 中以 Lpackage/name/ObjectName;的形式表示。前面的 L 表示这是一个对象类型，package/name/表示该对象所在的包，ObjectName 是对象的名字，“;”表示对象名称的结束。相当于 java 中的 package.name.ObjectName。例 如：Ljava/lang/String;相当于 java.lang.String。

smali文件

- “[”类型可以表示所有基本类型的数组。[I 表示一个整型一维数组，相当于 java 中的int[]。
- 对于多维数组，只要增加[就行了，[[I 相当于 int[][]，[[[I 相当于 int[][][]。
- 注意每一维的最多 255 个。对象数组的表示：[Ljava/lang/String;表示一个 String 对象数组。

smali文件

- 方法
- 方法调用的表示格式：Lpackage/name/ObjectName;->MethodName(III)Z。
- Lpackage/name/ObjectName;表示类型，MethodName 是方法名，III 为参数（在此是 3 个整型参数），Z 是返回类型（bool 型）。函数的参数是一个接一个的，中间没有隔开。
- 一个更复杂的例子：

method(I[[IILjava/lang/String;[Ljava/lang/Object;)Ljava/lang/String;

- 在 java 中则为：String method(int, int[][], int, String, Object[]).

smali文件

- 字段，即 java 中类的成员变量，表示格式：Lpackage/name/ObjectName;->FieldName:Ljava/lang/String;
- 即包名，字段名和字段类型
- 字段名与字段类型是以冒号“:”分隔

两种不同的寄存器表示法

- 有两种方式——v命名方式和p命名方式。
- p命名方式中的第一个寄存器就是方法中的第一个参数寄存器。
- 在下表中用这两种命名方式来表示上一个例子中有5个寄存器和3个参数的方法。

v0		the first local register
v1		the second local register
v2	p0	the first parameter register
v3	p1	the second parameter register
v4	p2	the third parameter register

两种不同的寄存器表示法

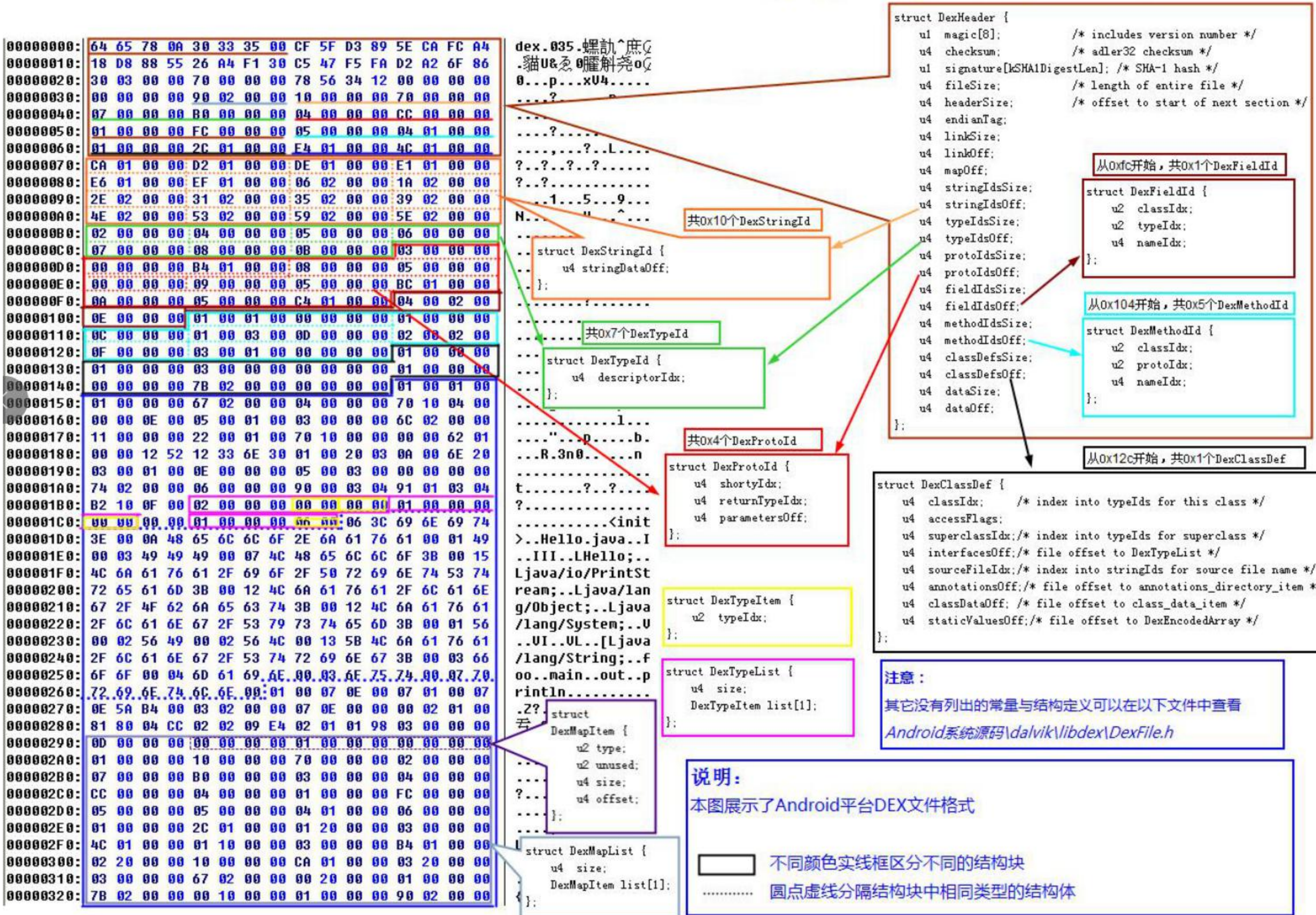
- 假设一个函数使用到 M 个寄存器，并且该函数有 N 个入参，根据 Dalvik 虚拟机参数传递方式中的规定：入参使用最后的 N 个寄存器，局部变量使用从 v0 开始的前 M-N 个寄存器。
- 比如，某函数 A 使用了 5 个寄存器，2 个显式的整形参数，如果函数 A 是非静态方法，函数被调用时会传入一个隐式的对象引用，因此实际传入的参数个数是 3 个。根据传参规则，局部变量将使用前 2 个寄存器，参数会使用后 3 个寄存器。

dex文件格式

- Dex文件头主要包括校验和以及其他结构的偏移地址和长度信息。

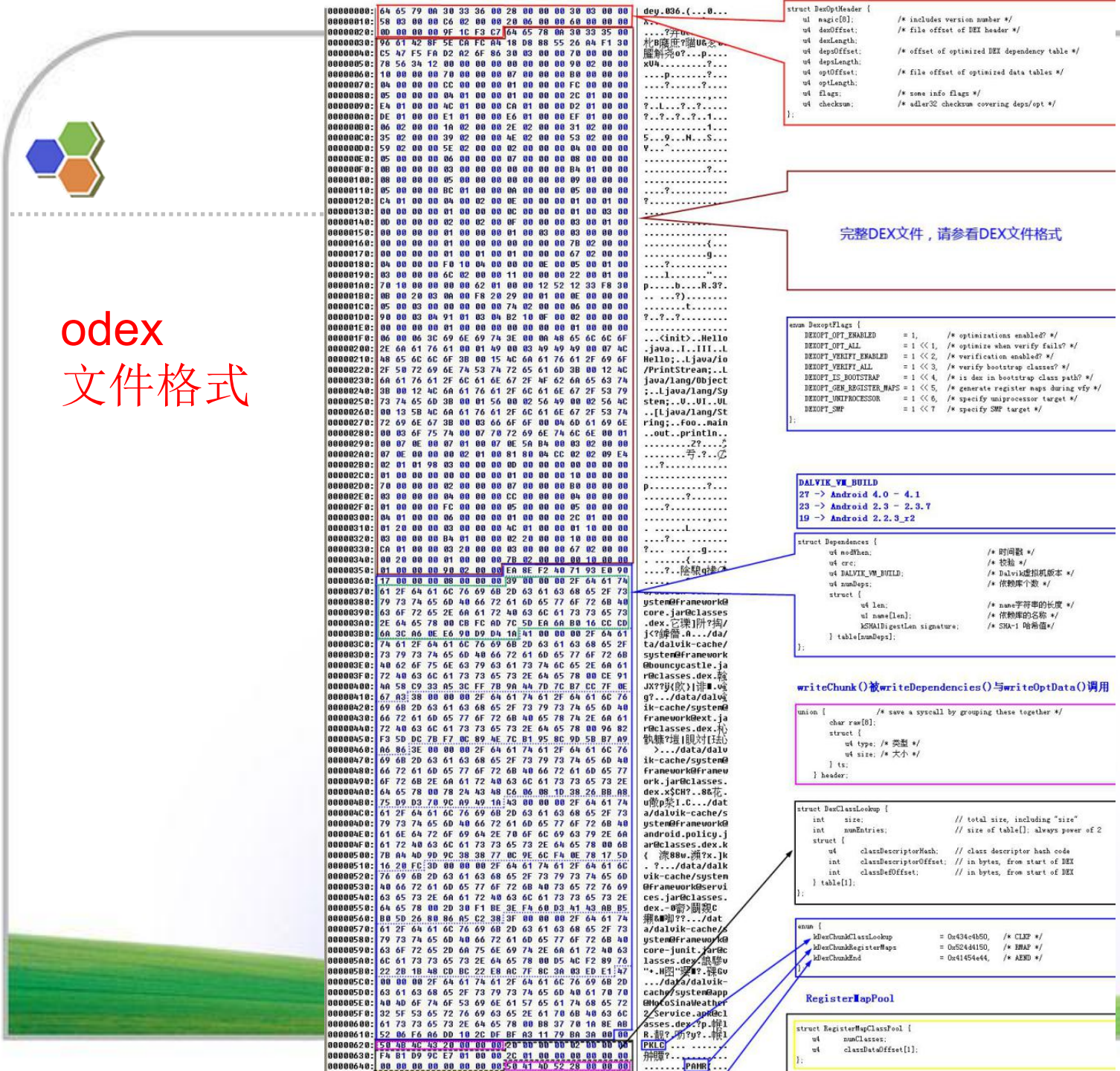
字段名称	偏移值	长度	描述
magic	0x0	8	'Magic'值，即魔数字段，格式如”dex/n035/0”，其中的035表示结构的版本。
checksum	0x8	4	校验码。采用adler32算法
signature	0xC	20	SHA-1签名。
file_size	0x20	4	Dex文件的总长度。
header_size	0x24	4	文件头长度，009版本=0x5C,035版本=0x70。
endian_tag	0x28	4	标识字节顺序的常量,根据这个常量可以判断文件是否交换了字节顺序,缺省情况下=0x78563412。

dex文件格式



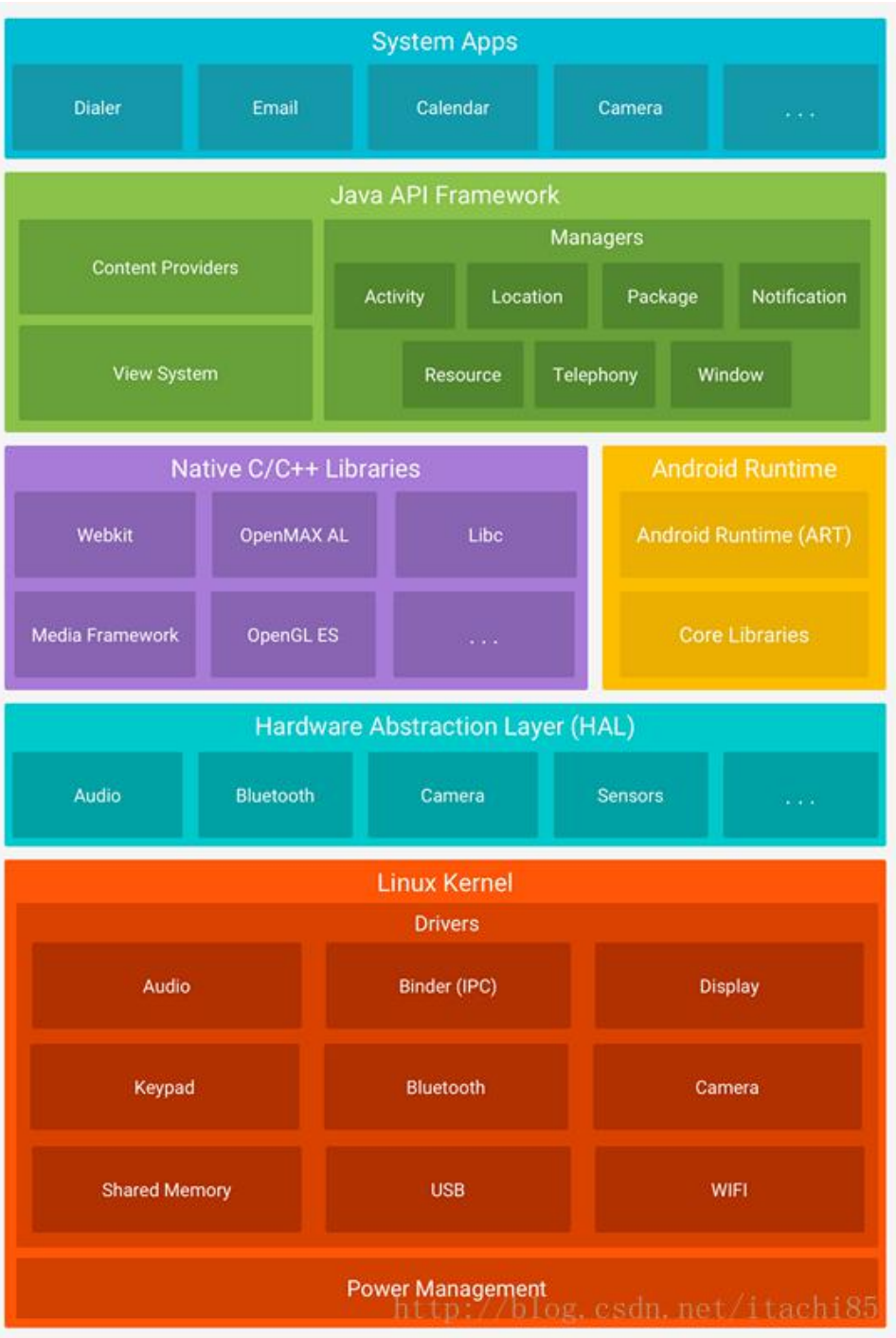
00000000: 64 65 78 0A 30 33 35 00 CF 5F D3 89 5E CA FC A4
00000010: 18 D8 88 55 26 A4 F1 30 C5 47 F5 FA D2 A2 6F 86
00000020: 30 03 00 00 70 00 00 00 78 56 34 12 00 00 00 00
00000030: 00 00 00 00 90 02 00 00 10 00 00 00 70 00 00 00
00000040: 07 00 00 00 B0 00 00 00 04 00 00 00 CC 00 00 00
00000050: 01 00 00 00 FC 00 00 00 05 00 00 00 04 01 00 00
00000060: 01 00 00 00 2C 01 00 00 E4 01 00 00 4C 01 00 00
00000070: CA 01 00 00 D2 01 00 00 DE 01 00 00 E1 01 00 00
00000080: E6 01 00 00 EF 01 00 00 06 02 00 00 1A 02 00 00
00000090: 2E 02 00 00 31 02 00 00 35 02 00 00 39 02 00 00
000000A0: 4E 02 00 00 53 02 00 00 59 02 00 00 5E 02 00 00
000000B0: 02 00 00 00 04 00 00 00 05 00 00 00 06 00 00 00
000000C0: 07 00 00 00 08 00 00 00 0B 00 00 00 03 00 00 00
000000D0: 00 00 00 00 B4 01 00 00 08 00 00 00 05 00 00 00
000000E0: 00 00 00 00 09 00 00 00 05 00 00 00 BC 01 00 00
000000F0: 0A 00 00 00 05 00 00 00 C4 01 00 00 04 00 02 00
00000100: 0E 00 00 00 01 00 01 00 00 00 00 00 01 00 00 00
00000110: 0C 00 00 00 01 00 03 00 0D 00 00 00 02 00 02 00
00000120: 0F 00 00 00 03 00 01 00 00 00 00 00 01 00 00 00
00000130: 01 00 00 00 03 00 00 00 00 00 00 00 01 00 00 00
00000140: 00 00 00 00 7B 02 00 00 00 00 00 00 01 00 01 00
00000150: 01 00 00 00 67 02 00 00 04 00 00 00 70 10 04 00
00000160: 00 00 0E 00 05 00 01 00 03 00 00 00 6C 02 00 00
00000170: 11 00 00 00 22 00 01 00 70 10 00 00 00 00 62 01
00000180: 00 00 12 52 12 33 6E 30 01 00 20 03 0A 00 6E 20
00000190: 03 00 01 00 0E 00 00 00 05 00 03 00 00 00 00 00
000001A0: 74 02 00 00 06 00 00 00 90 00 03 04 91 01 03 04
000001B0: B2 10 0F 00 02 00 00 00 00 00 00 00 01 00 00 00
000001C0: 00 00 00 00 01 00 00 00 00 00 06 3C 69 6E 69 74
000001D0: 3E 00 0A 48 65 6C 6C 6F 2E 6A 61 76 61 00 01 49
000001E0: 00 03 49 49 49 00 07 4C 48 65 6C 6C 6F 3B 00 15
000001F0: 4C 6A 61 76 61 2F 69 6F 2F 50 72 69 6E 74 53 74
00000200: 72 65 61 6D 3B 00 12 4C 6A 61 76 61 2F 6C 61 6E
00000210: 67 2F 4F 62 6A 65 63 74 3B 00 12 4C 6A 61 76 61
00000220: 2F 6C 61 6E 67 2F 53 79 73 74 65 6D 3B 00 01 56
00000230: 00 02 56 49 00 02 56 4C 00 13 5B 4C 6A 61 76 61
00000240: 2F 6C 61 6E 67 2F 53 74 72 69 6E 67 3B 00 03 66
00000250: 6F 6F 00 04 6D 61 69 6E 00 03 6F 75 74 00 07 70
00000260: 72 69 6E 74 6C 6E 00 01 00 07 0E 00 07 01 00 07
00000270: 0E 5A B4 00 03 02 00 00 07 0E 00 00 00 02 01 00
00000280: 81 80 04 CC 02 02 09 E4 02 01 01 98 03 00 00 00
00000290: 0D 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00
000002A0: 01 00 00 00 10 00 00 00 70 00 00 00 02 00 00 00
000002B0: 07 00 00 00 B0 00 00 00 03 00 00 00 04 00 00 00
000002C0: CC 00 00 00 04 00 00 00 01 00 00 00 FC 00 00 00
000002D0: 05 00 00 00 05 00 00 00 04 01 00 00 06 00 00 00
000002E0: 01 00 00 00 2C 01 00 00 01 20 00 00 03 00 00 00
000002F0: 4C 01 00 00 01 10 00 00 03 00 00 00 B4 01 00 00
00000300: 02 20 00 00 10 00 00 00 CA 01 00 00 03 20 00 00
00000310: 03 00 00 00 67 02 00 00 20 00 00 00 01 00 00 00
00000320: 7B 02 00 00 00 10 00 00 01 00 00 00 90 02 00 00
>..Hello.java..I
..III..LHello;..
Ljava/io/PrintSt
ream;..Ljava/lan
g/Object;..Ljava
/lang/System;..V
..VI..VL..[Ljava
/lang/String;..f
oo..main..out..p
rintln..........
struct DexHeader {
u1 magic[8]; /* includes version number */
u4 checksum; /* adler32 checksum */
u1 signature[kSHA1DigestLen]; /* SHA-1 hash */
u4 fileSize; /* length of entire file */
u4 headerSize; /* offset to start of next section */
u4 endianTag;
u4 linkSize;
u4 linkOff;
u4 mapOff;
u4 stringIdsSize;
u4 stringIdsOff;
u4 typeIdsSize;
u4 typeIdsOff;
u4 protoIdsSize;
u4 protoIdsOff;
u4 fieldIdsSize;
u4 fieldIdsOff;
u4 methodIdsSize;
u4 methodIdsOff;
u4 classDefsSize;
u4 classDefsOff;
u4 dataSize;
u4 dataOff;
};
共0x10个DexStringId
struct DexStringId {
u4 stringDataOff;
};
共0x7个DexTypeId
struct DexTypeId {
u4 descriptorIdx;
};
共0x4个DexProtoId
struct DexProtoId {
u4 shortyIdx;
u4 returnTypeIdx;
u4 parametersOff;
};
从0xfc开始，共0x1个DexFieldId
struct DexFieldId {
u2 classIdx;
u2 typeIdx;
u4 nameIdx;
};
从0x104开始，共0x5个DexMethodId
struct DexMethodId {
u2 classIdx;
u2 protoIdx;
u4 nameIdx;
};
从0x12c开始，共0x1个DexClassDef
struct DexClassDef {
u4 classIdx; /* index into typeIds for this class */
u4 accessFlags;
u4 superclassIdx;/* index into typeIds for superclass */
u4 interfacesOff;/* file offset to DexTypeList */
u4 sourceFileIdx;/* index into stringIds for source file name */
u4 annotationsOff;/* file offset to annotations_directory_item */
u4 classDataOff; /* file offset to class_data_item */
u4 staticValuesOff;/* file offset to DexEncodedArray */
};
struct DexTypeItem {
u2 typeIdx;
};
struct DexTypeList {
u4 size;
DexTypeItem list[1];
};
struct DexMapItem {
u2 type;
u2 unused;
u4 size;
u4 offset;
};
struct DexMapList {
u4 size;
DexMapItem list[1];
};
注意：
其它没有列出的常量与结构定义可以在以下文件中查看
Android系统源码\dalvik\libdex\DexFile.h
说明：
本图展示了Android平台DEX文件格式
不同颜色实线框区分不同的结构块
圆点虚线分隔结构块中相同类型的结构体

odex
文件格式


struct DexOptHeader {
u1 magic[8]; /* includes version number */
u4 dexOffset; /* file offset of DEX header */
u4 dexLength;
u4 depsOffset; /* offset of optimized DEX dependency table */
u4 depsLength;
u4 optOffset; /* file offset of optimized data tables */
u4 optLength;
u4 flags; /* some info flags */
u4 checksum; /* adler32 checksum covering deps/opt */
};
完整DEX文件，请参看DEX文件格式
enum DexoptFlags {
DEXOPT_OPT_ENABLED = 1, /* optimizations enabled? */
DEXOPT_OPT_ALL = 1 << 1, /* optimize when verify fails? */
DEXOPT_VERIFY_ENABLED = 1 << 2, /* verification enabled? */
DEXOPT_VERIFY_ALL = 1 << 3, /* verify bootstrap classes? */
DEXOPT_IS_BOOTSTRAP = 1 << 4, /* is dex in bootstrap class path? */
DEXOPT_GEN_REGISTER_MAPS = 1 << 5, /* generate register maps during vfy */
DEXOPT_UNIPROCESSOR = 1 << 6, /* specify uniprocessor target */
DEXOPT_SMP = 1 << 7 /* specify SMP target */
};
DALVIK_VM_BUILD
27 -> Android 4.0 - 4.1
23 -> Android 2.3 - 2.3.7
19 -> Android 2.2.3_r2
struct Dependences {
u4 modWhen; /* 时间戳 */
u4 crc; /* 校验 */
u4 DALVIK_VM_BUILD; /* Dalvik虚拟机版本 */
u4 numDeps; /* 依赖库个数 */
struct {
u4 len; /* name字符串的长度 */
u1 name[len]; /* 依赖库的名称 */
kSHA1DigestLen signature; /* SHA-1 哈希值*/
} table[numDeps];
};
writeChunk()被writeDependencies()与writeOptData()调用
union { /* save a syscall by grouping these together */
char raw[8];
struct {
u4 type; /* 类型 */
u4 size; /* 大小 */
} ts;
} header;
struct DexClassLookup {
int size; // total size, including "size"
int numEntries; // size of table[]; always power of 2
struct {
u4 classDescriptorHash; // class descriptor hash code
int classDescriptorOffset; // in bytes, from start of DEX
int classDefOffset; // in bytes, from start of DEX
} table[1];
};
enum {
kDexChunkClassLookup = 0x434c4b50, /* CLKP */
kDexChunkRegisterMaps = 0x524d4150, /* RMAP */
kDexChunkEnd = 0x41454e44, /* AEND */
};
RegisterMapPool
struct RegisterMapClassPool {
u4 numClasses;
u4 classDataOffset[1];
};

Android系统架构

- Android系统架构分为五层，从上到下依次是应用层、应用框架层、系统运行库层、硬件抽象层和Linux内核层。



Android系统启动流程

- 启动电源以及系统启动
- ✓ 当电源按下时引导芯片代码开始从预定义的地方（固化在ROM）开始执行。加载引导程序Bootloader到RAM，然后执行。
- 引导程序Bootloader
- ✓ 引导程序是在Android操作系统开始运行前的一个小程序，它的主要作用是把系统OS拉起来并运行。
- linux内核启动
- ✓ 内核启动时，设置缓存、被保护存储器、计划列表，加载驱动。当内核完成系统设置，它首先在系统文件中寻找”init”文件，然后启动root进程或者系统的第一个进程。
- init进程启动

静态分析Android程序

- 快速定位Android 程序的关键代码
- Smali文件格式
- Android 程序中的类
- 阅读反编译的smali代码
- 使用IDA Pro 静态分析Android 程序
- 阅读反编译的Java代码

快速定位Android 程序的关键代码

- 在逆向一个Android软件时，如果盲目的分析，可能需要阅读成千上万行的反汇编代码才能找到程序的关键点，这无疑是浪费时间的表现

程序的主Activity

- 一个Android程序由一个或多个 Activity以及其它组件组成，每个Activity都是相同级别的，不同的Activity实现不同的功能。每个Activity都是Android程序的一个显示“页面”，主要负责数据的处理及展示工作，在Android 程序的开发过程中，程序员很多时候是在编写用户与Activity之间的交互代码。
- 每个Android程序有且只有一个主 Activity（隐藏程序除外，它没有主Activity），它是程序启动的第一个Activity。

程序的主Activity

- 在反编译出的AndroidManifest.xml 中找到主Activity后，可以直接去查看其所在类的OnCreate()方法的反汇编代码，对于大多数软件来说，这里就是程序的代码入口处，所有的功能都从这里开始得到执行，我们可以沿着这里一直向下查看，追踪软件的执行流程。

查找程序入口

AndroidManifest.xml* | DoctorAnimationActivity.smali | TccTeaEncryptDecrypt.smali | ap.smali

```
197                 <category android:name="android.intent.category.DEFAULT" />
198             </intent-filter>
199             <intent-filter>
200                 <action android:name="com.tencent.qqtransfer.intent.action.OPEN" />
201                 <category android:name="com.tencent.transfer" />
202                 <category android:name="android.intent.category.DEFAULT" />
203             </intent-filter>
204         </activity>
205         <activity android:theme="@android:style/Theme.NoTitleBar" android:name=".ui.packcontact.PackContactActivi
206         <activity android:theme="@android:style/Theme.NoTitleBar" android:name=".ui.packcontact.PackContactPrevie
207         <activity android:theme="@android:style/Theme.NoTitleBar" android:name=".ui.packcontact.SelectContactsAct
208         <activity android:theme="@android:style/Theme.NoTitleBar" android:name=".ui.IntroToTransferContactActivit
209         <activity android:theme="@android:style/Theme.NoTitleBar" android:name=".ui.JustGetContactNumActivity" an
210         <activity android:theme="@style/dialog" android:name=".ui.Guide33003Activity" android:launchMode="singleT
211         <activity android:theme="@style/dialog" android:name=".ui.WhyLoginQQActivity" android:launchMode="singleT
212         <activity android:theme="@android:style/Theme.NoTitleBar" android:name=".ui.transfer.TransferMainActivity
213         <activity android:theme="@style/NoTitleTransparentTheme" android:name=".apps.doctor.ui.DoctorAnimationAct
214             <intent-filter>
215                 <action android:name="android.intent.action.MAIN" />
216                 <category android:name="android.intent.category.DEFAULT" />
217             </intent-filter>
```

需重点关注的Application类

- 如果需要在程序的组件之间传递全局变量，或者在Activity启动之前做一些初始化工作，就可以考虑使用Application 类了。
- 鉴于 Application 类比程序中其它的类启动得都要早，一些商业软件将授权验证的代码都转移到了该类中。例如，在OnCreate()方法中检测软件的购买状态，如果状态异常则拒绝程序继续运行。
- 因此，在分析 Android程序过程中，我们需要先查看该程序是否具有Application 类，如果有，就要看看它的OnCreate()方法中是否做了一些影响到逆向分析的初始化工作。

如何定位关键代码——信息反馈法

- 所谓信息反馈法，是指先运行目标程序，然后根据程序运行时给出的反馈信息作为突破口寻找关键代码。
- 例如，运行目标程序并输入错误的注册码时，会弹出提示“无效用户名或注册码”，这就是程序反馈给我们的信息
- 通常情况下，程序中用到的字符串会存储在String.xml文件或者硬编码到程序代码中，如果是前者的话，字符串在程序中会以id 的形式访问，只需在反汇编代码中搜索字符串的id 值即可找到调用代码处；如果是后者的话，在反汇编代码中直接搜索字符串即可。

如何定位关键代码——特征函数法

- 这种定位代码的方法与信息反馈法类似。在信息反馈法中，无论程序给出什么样的反馈信息，终究是需要调用Android SDK 中提供的相关API 函数来完成的。
- 比如弹出注册码错误的提示信息就需要调用Toast.MakeText().Show()方法，在反汇编代码中直接搜索Toast应该很快就能定位到调用代码，如果 Toast在程序中有多处的话，可能需要分析人员逐个甄别。

如何定位关键代码——顺序查看法

- 顺序查看法是指从软件的启动代码开始，逐行的向下分析，掌握软件的执行流程，这种分析方法在病毒分析时经常用到。

如何定位关键代码——代码注入法

- 代码注入法属于动态调试方法，它的原理是手动修改 apk 文件的反汇编代码，加入Log 输出，配合 LogCat查看程序执行到特定点时的状态数据。
- 这种方法在解密程序数据时经常使用

如何定位关键代码——栈跟踪法

- 栈跟踪法属于动态调试方法，它的原理是输出运行时的栈跟踪信息，然后查看栈上的函数调用序列来理解方法的执行流程

如何定位关键代码——Method Profiling

- Method Profiling（方法剖析）属于动态调试方法，它主要用于热点分析和性能优化。
- 该功能除了可以记录每个函数占用的CPU 时间外，还能够跟踪所有的函数调用关系，并提供比栈跟踪法更详细的函数调用序列报告，这种方法在实践中可帮助分析人员节省很多时间，也被广泛使用

使用IDA Pro静态分析Android程序

- IDA Pro 从6.1 版本开始，提供了对 Android的静态分析与动态调试支持。包括 Dalvik指令集的反汇编、原生库（ARM/Thumb 代码）的反汇编、原生库（ARM/Thumb 代码）的动态调试等。
- 具体可查看IDA Pro 官方的更新日志，链接如下：http://www.hex-rays.com/products/ida/6.1/index.shtml

阅读反编译的Java代码

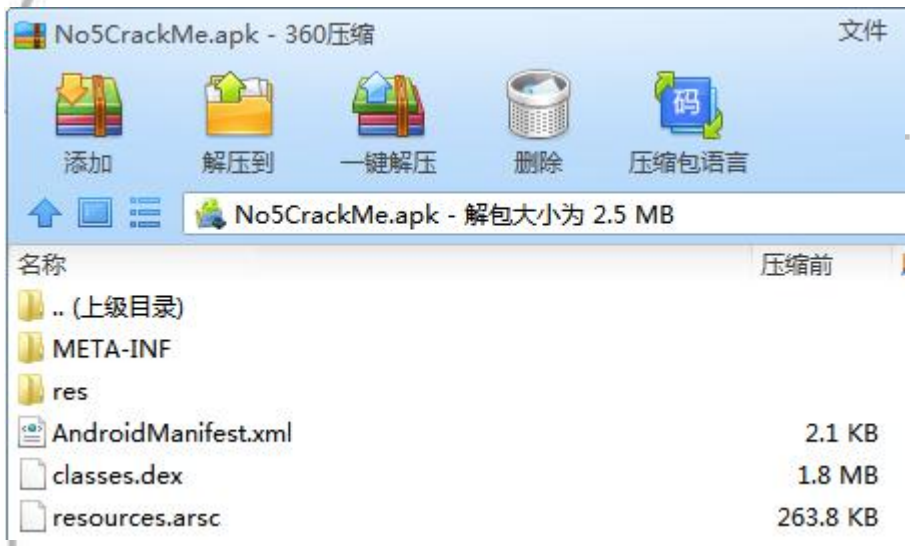
- 使用dex2jar生成jar文件
- 使用jd-gui查看jar文件的源码

实例

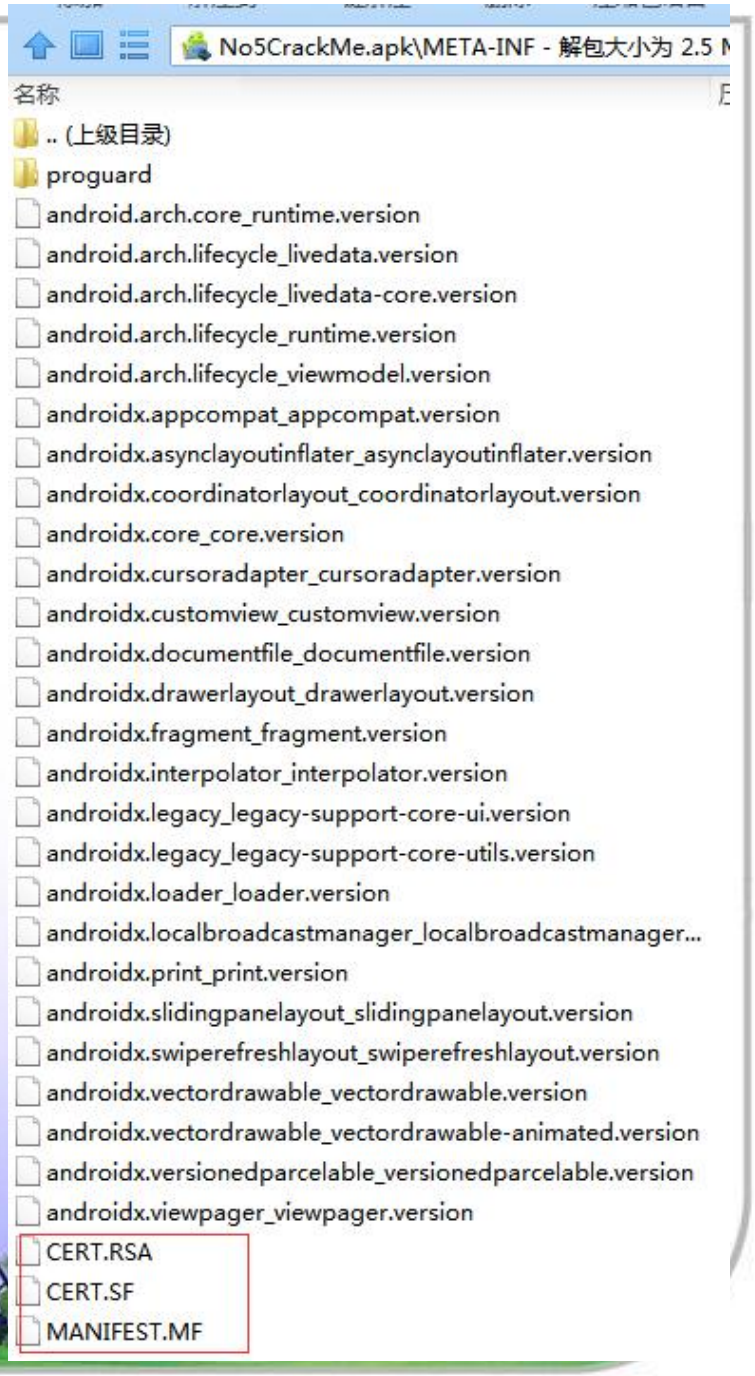

驗證
看雪CTF.TSRC 2018 團隊賽.kwaiching


1234567890
驗證
看雪CTF.TSRC 2018 團隊賽.kwaiching
f a i l e d ! ! !

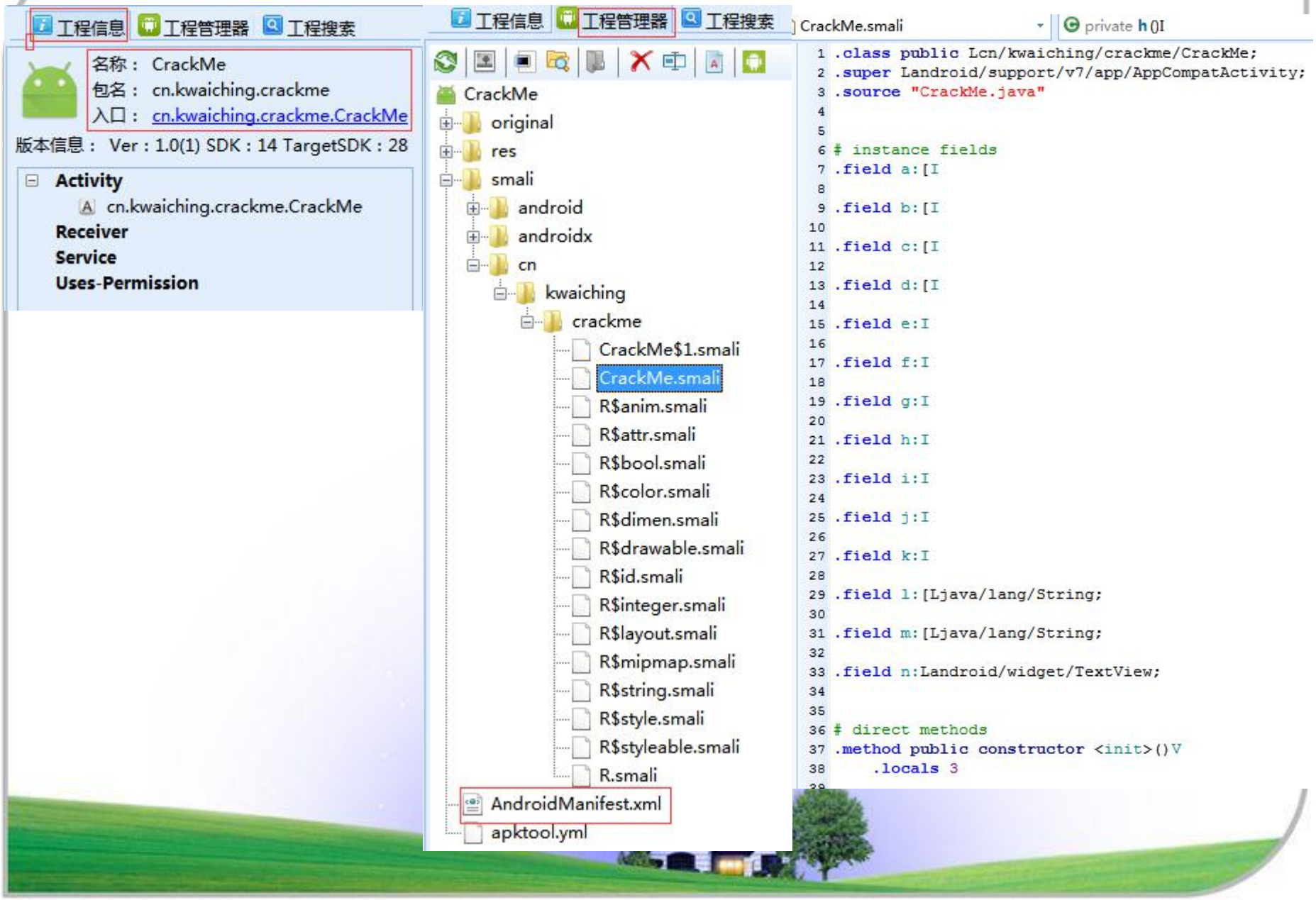
实例


No5CrackMe.apk - 360压缩
文件
添加
解压到
一键解压
删除
压缩包语言
No5CrackMe.apk - 解包大小为 2.5 MB
名称
压缩前
.. (上级目录)
META-INF
res
AndroidManifest.xml
2.1 KB
classes.dex
1.8 MB
resources.arsc
263.8 KB


No5CrackMe.apk\res
名称
.. (上级目录)
anim
color
color-v21
color-v23
drawable
drawable-hdpi-v4
drawable-ldrtl-hdpi-v17
drawable-ldrtl-mdpi-v17
drawable-ldrtl-xhdpi-v17
drawable-ldrtl-xxhdpi-v17
drawable-ldrtl-xxxhdpi-v17


No5CrackMe.apk\META-INF - 解包大小为 2.5 M
名称
.. (上级目录)
proguard
android.arch.core_runtime.version
android.arch.lifecycle_livedata.version
android.arch.lifecycle_livedata-core.version
android.arch.lifecycle_runtime.version
android.arch.lifecycle_viewmodel.version
androidx.appcompat_appcompat.version
androidx.asynclayoutinflater_asynclayoutinflater.version
androidx.coordinatorlayout_coordinatorlayout.version
androidx.core_core.version
androidx.cursoradapter_cursoradapter.version
androidx.customview_customview.version
androidx.documentfile_documentfile.version
androidx.drawerlayout_drawerlayout.version
androidx.fragment_fragment.version
androidx.interpolator_interpolator.version
androidx.legacy_legacy-support-core-ui.version
androidx.legacy_legacy-support-core-utils.version
androidx.loader_loader.version
androidx.localbroadcastmanager_localbroadcastmanager...
androidx.print_print.version
androidx.slidingpanelayout_slidingpanelayout.version
androidx.swiperefreshlayout_swiperefreshlayout.version
androidx.vectordrawable_vectordrawable.version
androidx.vectordrawable_vectordrawable-animated.version
androidx.versionedparcelable_versionedparcelable.version
androidx.viewpager_viewpager.version
CERT.RSA
CERT.SF
MANIFEST.MF

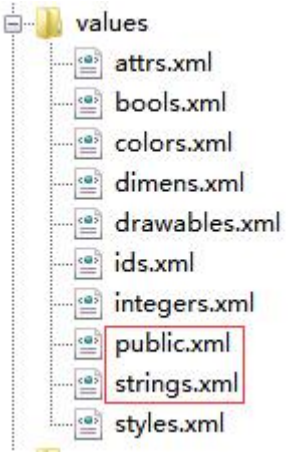
实例


工程信息
工程管理器
工程搜索
名称： CrackMe
包名： cn.kwaiching.crackme
入口： cn.kwaiching.crackme.CrackMe
版本信息： Ver : 1.0(1) SDK : 14 TargetSDK : 28
Activity
cn.kwaiching.crackme.CrackMe
Receiver
Service
Uses-Permission
工程信息
工程管理器
工程搜索
CrackMe
original
res
smali
android
androidx
cn
kwaiching
crackme
CrackMe$1.smali
CrackMe.smali
R$anim.smali
R$attr.smali
R$bool.smali
R$color.smali
R$dimen.smali
R$drawable.smali
R$id.smali
R$integer.smali
R$layout.smali
R$mipmap.smali
R$string.smali
R$style.smali
R$styleable.smali
R.smali
AndroidManifest.xml
apktool.yml
CrackMe.smali
private h()I
1 .class public Lcn/kwaiching/crackme/CrackMe;
2 .super Landroid/support/v7/app/AppCompatActivity;
3 .source "CrackMe.java"
4
5
6 # instance fields
7 .field a:[I
8
9 .field b:[I
10
11 .field c:[I
12
13 .field d:[I
14
15 .field e:I
16
17 .field f:I
18
19 .field g:I
20
21 .field h:I
22
23 .field i:I
24
25 .field j:I
26
27 .field k:I
28
29 .field l:[Ljava/lang/String;
30
31 .field m:[Ljava/lang/String;
32
33 .field n:Landroid/widget/TextView;
34
35
36 # direct methods
37 .method public constructor <init>()V
38 .locals 3

实例

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8" standalone="no"?><manifest xmlns:androi
    <application android:allowBackup="true" android:appComponentFactory="and
        <activity android:name="cn.kwaiching.crackme.CrackMe">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
</manifest>
```




```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="abc_action_bar_home_description">Navigate home</string>
    <string name="abc_action_bar_up_description">Navigate up</string>
    <string name="abc_action_menu_overflow_description">More options</string>
    <string name="abc action mode done">Done</string>
```

```
    <string name="app_name">CrackMe</string>
    <string name="kwaiching">看雪CTF.TSRC 2018 團隊賽.kwaiching</string>
    <string name="me">"
s \u0009\u0009u \u0009\u0009\u0009c \u0009\u0009\u0009\u0009c \u0009\u0009
```

```
"</string>
    <string name="notMe">f \u0009a \u0009\u0009i \u0009\u0009\u00091 \u
    <string name="ok">驗證</string>
    <string name="search_menu_title">Search</string>
    <string name="status_bar_notification_info_overflow">999+</string>
    <string name="success00" />
    <string name="success34">"
財穀有餘主得內助富貴之命
```

实例

droidManifest.xml | public.xml | strings.xml

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3     <public type="anim" name="abc_fade_in" id="0x7f010000" />
4     <public type="anim" name="abc_fade_out" id="0x7f010001" />
5     <public type="anim" name="abc_grow_fade_in_from_bottom" id="0x7f010002" />
6     <public type="anim" name="abc_popup_enter" id="0x7f010003" />

884     <public type="string" name="me" id="0x7f0b0029" />
885     <public type="string" name="notMe" id="0x7f0b002a" />
886     <public type="string" name="ok" id="0x7f0b002b" />
```

- values
  - attrs.xml
  - bools.xml
  - colors.xml
  - dimens.xml
  - drawables.xml
  - ids.xml
  - integers.xml
  - public.xml
  - strings.xml
  - styles.xml

```
private void a(int arg8) {
    int v0 = 0x7F0B002A;
    int v1 = 34;
    if(arg8 <= v1) {
        if(arg8 < v1) {
        }
        else {
            try {
                this.n.setText(String.format("%s%s", this.getString(0x7F0B0029), this.l[arg8]));
                this.findViewById(0x7F070059).setEnabled(false);
                return;
            label_23:
                this.n.setText(this.getString(v0));
            }
            catch(Exception ) {
                this.n.setText(this.getString(v0));
            }

            return;
        }
    }

    goto label_23;
}
```

实例

Unicode编码 | UTF-8编码 | URL编码/解码 | Unix时间戳 | Ascii/N

f \u0009a \u0009\u0009i \u0009\u0009\u0009l \u0009\u0009\u0009\u0009e
\u0009\u0009\u0009\u0009\u0009d \u0009\u0009\u0009\u0009\u0009\u0009!
\u0009\u0009\u0009\u0009\u0009\u0009\u0009!
\u0009\u0009\u0009\u0009\u0009\u0009\u0009\u0009!

f a i l e d
! ! !

<string name="notMe">

<public type="string" name="notMe" id="0x7f0b002a" />

Unicode编码 | UTF-8编码 | URL编码/解码 | Unix时间戳 | Ascii/N

s \u0009\u0009u \u0009\u0009\u0009c \u0009\u0009\u0009\u0009c
\u0009\u0009\u0009\u0009\u0009e \u0009\u0009\u0009\u0009\u0009\u0009s
\u0009\u0009\u0009\u0009\u0009\u0009\u0009s
\u0009\u0009\u0009\u0009\u0009\u0009\u0009\u0009!
\u0009\u0009\u0009\u0009\u0009\u0009\u0009\u0009!

<public type="string" name="me" id="0x7f0b0029" />

<string name="me">"

s u c c e
s s !
!

实例

```
56      <string name="success00" />
57      <string name="success34">"
58  財穀有餘主得內助富貴之命
59
60
61  此命福氣果如何，僧道門中衣祿多；
62  離祖出家方為妙，朝晚拜佛念彌陀。
63
64
65
66  此命推來為人性躁；
67  與人做事反為不美；
```

```
protected void onCreate(Bundle arg2) {
    super.onCreate(arg2);
    this.setContentView(0x7F09001C);
    this.b();
    this.n = this.findViewById(0x7F07003A);
    this.findViewById(0x7F070059).setOnClickListener(new View$OnClickListener() {
        public void onClick(View arg3) {
            try {
                CrackMe.a(this.a);
            }
            catch(Exception ) {
                this.a.n.setText(this.a.getString(0x7F0B002A));
            }
        }
    });
}
```

执行初始化

点击“验证”时执行的函数

```
889     <public type="string" name="success00" id="0x7f0b002e" />
890     <public type="string" name="success34" id="0x7f0b002f" />
```

```
private void b() {
    int v0;
    for(v0 = 0; v0 <= 72; ++v0) {
        this.l[v0] = v0 != 34 ? this.getResources().getString(0x7F0B002E) : this.getResources().getString(0x7F0B002F);
    }
}
```

```
static void a(CrackMe arg0) {
    arg0.a();
}
```

在onCreate中被调用

```
private void a() {
    int v0 = 0x7F0B002A;
    try {
        this.c();
        if(this.j != 0 && this.i != 0 && this.h != 0) {
            this.d();
            this.a(this.e() + this.f() + this.g() + this.h());
            return;
        }

        this.n.setText(this.getString(v0));
    }
    catch(Exception ) {
        this.n.setText(this.getString(v0));
    }
}
```

错误的话，显示Failed信息！

```
private void a(int arg8) {
    int v0 = 0x7F0B002A;
    int v1 = 34;
    if(arg8 <= v1) {
        if(arg8 < v1) {
        }
        else {
            try {
                this.n.setText(String.format("%s%s", this.getString(0x7F0B0029), this.l[arg8]));
                this.findViewById(0x7F070059).setEnabled(false);
                return;
            label_23:
                this.n.setText(this.getString(v0));
            }
            catch(Exception ) {
                this.n.setText(this.getString(v0));
            }

            return;
        }
    }

    goto label_23;
}
```

等于34时执行这段代码

success!!

一首诗

实例

```
private void a() {
    int v0 = 0x7F0B002A;
    try {
        this.c();
        if(this.j != 0 && this.i != 0 && this.h != 0) {
            this.d();
            this.a(this.e() + this.f() + this.g() + this.h());
            return;
        }

        this.n.setText(this.getString(v0));
    }
    catch(Exception ) {
        this.n.setText(this.getString(v0));
    }
}
```

错误的话，显示Failed信息！

```
private void c() {
    int v0 = 0x7F07002B;
    try {
        String v0_1 = this.findViewById(v0).getText().toString();
        this.j = 0;
        this.i = 0;
        this.h = 0;
        int v3 = 4;
        String v2 = v0_1.length() > v3 ? v0_1.substring(0, v3) : v0_1;
        this.j = Integer.parseInt(v2);
        if(this.j > 0 && this.j < 189) {
            this.j = 0;
        }

        if(this.j <= 1983 || this.j >= 2007) {
            this.j = 0;
        }

        int v4 = 6;
        v2 = v0_1.length() > v4 ? v0_1.substring(v3, v4) : v0_1;
        this.i = Integer.parseInt(v2);
        if(this.i < 1 || this.i > 12) {
            this.i = 0;
        }

        int v5 = 8;
        if(v0_1.length() > v5) {
            v0_1 = v0_1.substring(v4, v5);
        }

        this.h = Integer.parseInt(v0_1);
        if(this.h >= 1 && this.h <= 31) {
            return;
        }

        this.h = 0;
    }
    catch(Exception ) {
        this.n.setText(this.getString(0x7F0B002A));
    }
}
```

取前4个字符

在（0,189）或（189,1983]或>2007时j=0

取接下来的两个字符

<1或>12时，i=0

取第7-8两个字符

<1或>31时，h=0

基于Android的ARM汇编语言基础

- Android与ARM处理器
- ARM汇编语言程序结构
- ARM处理器寻址方式
- ARM与Thumb指令集

Android与ARM处理器

- ARM处理器是一种32位嵌入式RISC处理器。ARM（Advanced RISC Machines，更早称作Acorn RISC Machine），既可以认为是一个公司的名字，也可以认为是对微处理器的通称，还可以认为是一种技术的名字
  - ✓ 1、体积小、低功耗、低成本、高性能；
  - ✓ 2、支持Thumb（16位）/ARM（32位）双指令集，能很好的兼容8位/16位器件；
  - ✓ 3、大量使用寄存器，指令执行速度更快；
  - ✓ 4、大多数数据操作都在寄存器中完成；
  - ✓ 5、寻址方式灵活简单，执行效率高；
  - ✓ 6、指令长度固定。

Android与ARM处理器

- ARM处理器共有37个寄存器，被分为若干个组，这些寄存器包括
  - 31个通用寄存器，包括程序计数器，均为32位的寄存器
  - 6个状态寄存器，用以标识CPU的工作状态及程序的运行状态，均为32位，目前只使用了其中的一部分。
- ARM微处理器的在较新的体系结构中支持两种指令集：ARM指令集和Thumb指令集。
- ARM指令为32位的长度，Thumb指令为16位长度。Thumb指令集为ARM指令集的功能子集，但与等价的ARM代码相比较，可节省30%～40%以上的存储空间，同时具备32位代码的所有优点。

Android与ARM处理器

- Android最初选择ARM作为其主要支持的硬件平台，除了市场原因外，在技术层面上可能是更多的考虑到ARM处理器的高性能，及其使用的广泛性。可以吸引以前在ARM平台上开发的工程师，加入到Android的开发行列中。
- 从Android1.6开始,Dalvik虚拟机提供了x86架构的支持
- Android 4.1加入了对MIPS的支持

ARM汇编语言程序结构

- 在 ARM 汇编语言程序中，以程序段为单位组织代码。
- 段是相对独立的指令或数据序列，具有特定的名称。
- 段可以分为代码段和数据段，代码段的内容为执行代码，数据段存放代码运行时需要用到的数据。
- 一个汇编程序至少应该有一个代码段，当程序较长时，可以分割为多个代码段和数据段，多个段在程序编译链接时最终形成一个可执行的映象文件。

ARM汇编语言程序结构

- 可执行映象文件通常由以下几部分构成：
  - ✓ 一个或多个代码段，代码段的属性为只读。
  - ✓ 零个或多个包含初始化数据的数据段，数据段的属性为可读写。
  - ✓ 零个或多个不包含初始化数据的数据段，数据段的属性为可读写
- 链接器根据系统默认或用户设定的规则，将各个段安排在存储器中的相应位置。因此源程序中段之间的相对位置与可执行的映象文件中段的相对位置一般不会相同。

程序示例

```
EXPORT Java_com_tencent_mm_protocal_MMProtocalJni_pack
Java_com_tencent_mm_protocal_MMProtocalJni_pack

var_28= -0x28
var_24= -0x24
var_20= -0x20
var_1C= -0x1C
var_18= -0x18
var_14= -0x14
var_10= -0x10
var_C= -0xC
arg_0=  0
arg_4=  4
arg_8=  8
arg_C=  0xC
arg_10=  0x10
arg_14=  0x14
arg_18=  0x18
arg_1C=  0x1C

PUSH    {R4,LR}
SUB     SP, SP, #0x20
LDR     R4, [SP,#0x28+arg_0]
STR     R4, [SP,#0x28+var_28]
LDR     R4, [SP,#0x28+arg_4]
STR     R4, [SP,#0x28+var_24]
LDR     R4, [SP,#0x28+arg_8]
STR     R4, [SP,#0x28+var_20]
LDR     R4, [SP,#0x28+arg_C]
STR     R4, [SP,#0x28+var_1C]
LDR     R4, [SP,#0x28+arg_10]
STR     R4, [SP,#0x28+var_18]
LDR     R4, [SP,#0x28+arg_14]
STR     R4, [SP,#0x28+var_14]
LDR     R4, [SP,#0x28+arg_18]
STR     R4, [SP,#0x28+var_10]
LDR     R4, [SP,#0x28+arg_1C]
STR     R4, [SP,#0x28+var_C]
BL      sub_EEEC
ADD     SP, SP, #0x20
POP     {R4,PC}
```

ARM处理器寻址方式

- ARM处理器有9种基本寻址方式：寻址方式是根据指令中给出的地址码字段来实现寻找真实操作数地址的方式。
- 1．寄存器寻址
  - ✓操作数的值在寄存器中，指令中的地址码字段给出的是寄存器编号，寄存器的内容是操作数，指令执行时直接取出寄存器值操作。
  - ✓例如指令：
  - ✓**MOV R1,R2 ； R1←R2**
  - ✓**SUB R0,R1,R2 ； R0←R1- R2**

ARM处理器寻址方式

- 2．立即寻址

在立即寻址指令中数据就包含在指令当中，立即寻址指令的操作码字段后面的地址码部分就是操作数本身，取出指令也就取出了可以立即使用的操作数（也称为立即数）。立即数要以“＃”为前缀，表示16进制数值时以“0x”表示。

例如指令：

- ✓ **ADD R0,R0,#1 ； R0←R0 + 1**
- ✓ **MOV R0,#0xff00 ； R0←0xff00**

ARM处理器寻址方式

- 3．寄存器移位寻址

寄存器移位寻址是ARM指令集特有的寻址方式。第2个寄存器操作数在与第1个操作数结合之前，先进行移位操作。

例如指令：

- ✓ MOV R0,R2,LSL #3 ； R2的值左移3位，结果放入R0，即R0=R2 * 8
- ✓ ANDS R1,R1,R2,LSL R3 ； R2的值左移R3位，然后和R1相与操作，结果放入R1

ARM处理器寻址方式

- 4．寄存器间接寻址

  指令中的地址码给出的是一个通用寄存器编号，所需要的操作数保存在寄存器指定地址的存储单元中，即寄存器为操作数的地址指针，操作数存放在存储器中。

  例如指令：

  - ✓ LDR R0,[R1]；R0←[R1]（将R1中的数值作为地址，取出此地址中的数据保存在R0中）
  - ✓ STR R0,[R1]；[R1] ←R0

ARM处理器寻址方式

- 5．变址寻址

变址寻址是将基址寄存器的内容与指令中给出的偏移量相加，形成操作数的有效地址。变址寻址用于访问基址附近的存储单元，常用于查表，数组操作，功能部件寄存器访问等。

例如指令：

- ✓ LDR R2,[R3,#4]；R2←[R3 + 4]（将R3中的数值加4作为地
- ✓ 址，取出此地址的数值保存在R2 中）
- ✓ STR R1,[R0,#-2]；[R0-2] ← R1（将R0中的数值减2 作为地址，把R1中的内容保存到此地址位置）

ARM处理器寻址方式

- 6．多寄存器寻址

  采用多寄存器寻址方式，一条指令可以完成多个寄存器值的传送，这种寻址方式用一条指令最多可以完成16个寄存器值的传送。

  例如指令：

  ✓ **LDMIA R1, {R0,R2,R5}；R0<-mem32[R1];R2<-mem32[R1+4];R5<-mem32[R1+8]**

ARM处理器寻址方式

- 7.堆栈寻址

堆栈是一种数据结构，堆栈是特定顺序进行存取的存储区，操作顺序分为“后进先出”和“先进后出”，堆栈寻址时隐含的，它使用一个专门的寄存器（堆栈指针）指向一块存储区域（堆栈），指针所指向的存储单元就是堆栈的栈顶。存储器生长堆栈可分为两种：

- ✓ 向上生长：向高地址方向生长，称为递增堆栈（Ascending Stack）
- ✓ 向下生长：向低地址方向生长，称为递减堆栈（Decending Stack）

ARM处理器寻址方式

- 8．块复制寻址
  - ✓ 块复制寻址用于把一块从存储器的某一位置复制到另一位置，是一个多寄存器传送指令。例如指令：
  - ✓ **STMIA R0!,{R1-R7}**；将R1～R7的数据保存到存储器中，存储器指针在保存第一个值之后增加，增长方向为向上增长。
  - ✓ **STMDA R0!,{R1-R7}**；将R1～R7的数据保存到存储器中，存储器指针在保存第一个值之后增加，增长方向为向下增长。

ARM处理器寻址方式

- 9．相对寻址

相对寻址是变址寻址的一种变通，由程序计数器PC提供基准地址，指令中的地址码字段作为偏移量，两者相加后得到的地址即为操作数的有效地址。

例如指令：

- ✓ BL ROUTE1 ； 调用到ROUTE1子程序
- ✓ BEQ LOOP ； 条件跳转到LOOP标号处

THUMB指令集

- Thumb指令集是ARM指令集的一个子集，是针对代码密度问题而提出的，它具有16位的代码宽度。与等价的32位代码相比较，Thumb指令集在保留32位代码优势的同时，大大的节省了系统的存储空间。
- Thumb不是一个完整的体系结构，不能指望处理器只执行Thumb指令集而不支持ARM指令集。
- 当处理器在执行ARM程序段时，称ARM处理器处于ARM工作状态，当处理器在执行Thumb程序段时，称ARM处理器处于Thumb工作状态。
- Thumb指令集并没有改变ARM体系底层的编程模型，只是在该模型上增加了一些限制条件，只要遵循一定的调用规则，Thumb子程序和ARM子程序就可以互相调用。

THUMB指令集

- 与ARM指令集相比较，Thumb指令集中的数据处理指令的操作数仍然是32位，指令地址也为32位，但Thumb指令集为实现16位的指令长度，舍弃了ARM指令集的一些特性，如大多数的Thumb指令是无条件执行的，而几乎所有的ARM指令都是有条件执行的，大多数的Thumb数据处理指令采用2地址格式。
- 由于Thumb指令的长度为16位，即只用ARM指令一半的位数来实现同样的功能，所以，要实现特定的程序功能，所需的Thumb指令的条数较ARM指令多。

THUMB指令集格式

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
1	0	0	0	Op		Offset5					Rs			Rd			Move Shifted register
2	0	0	0	1	1	I	Op	Rn/offset3			Rs			Rd			Add/subtract
3	0	0	1	Op		Rd			Offset8								Move/compare/add/ subtract immediate
4	0	1	0	0	0	0	Op				Rs			Rd			ALU operations
5	0	1	0	0	0	1	Op		H1	H2	Rs/Hs			Rd/Hd			Hi register operations /branch exchange
6	0	1	0	0	1	Rd			Word8								PC-relative load
7	0	1	0	1	L	B	0	Ro			Rb			Rd			Load/store with register offset
8	0	1	0	1	H	S	1	Ro			Rb			Rd			Load/store sign-extended byte/halfword
9	0	1	1	B	L	Offset5					Rb			Rd			Load/store with immediate offset
10	1	0	0	0	L	Offset5					Rb			Rd			Load/store halfword
11	1	0	0	1	L	Rd			Word8								SP-relative load/store
12	1	0	1	0	SP	Rd			Word8								Load address
13	1	0	1	1	0	0	0	0	S	SWord7							Add offset to stack pointer
14	1	0	1	1	L	1	0	R	Rlist								Push/pop register
15	1	1	0	0	L	Rb			Rlist								Multiple load/store
16	1	1	0	1	Cond				Softset8								Conditional branch
17	1	1	0	1	1	1	1	1	Value8								Software interrupt
18	1	1	1	0	0	Offset11											Unconditional branch
19	1	1	1	1	H	Offset											Long branch with link
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	

THUMB指令集格式

```
text:000E9A30                sub_E9A30                               ; CODE XREF: .text:000E9752↑p
text:000E9A30                ; __unwind {
text:000E9A30 38 B5                       PUSH    {R3-R5,LR}
text:000E9A32 0D 4C                       LDR     R4, =(byte_1445F4 - 0xE9A38)
text:000E9A34 7C 44                       ADD     R4, PC          ; byte_1445F4
text:000E9A36 25 78                       LDRB    R5, [R4]
text:000E9A38 19 F0 CA FA                 BL      sub_102FD0
text:000E9A3C EB 07                       LSLS    R3, R5, #0x1F
text:000E9A3E 02 D5                       BPL     loc_E9A46
text:000E9A40
text:000E9A40                loc_E9A40                               ; CODE XREF: sub_E9A30+1E↓j
text:000E9A40                                                        ; sub_E9A30+36↓j
text:000E9A40 0A 48                       LDR     R0, =(off_13E0C4 - 0xE9A46)
text:000E9A42 78 44                       ADD     R0, PC          ; off_13E0C4
text:000E9A44 38 BD                       POP     {R3-R5,PC}
text:000E9A46                ; ---------------------------------------------------------------------------
text:000E9A46
text:000E9A46                loc_E9A46                               ; CODE XREF: sub_E9A30+E↑j
text:000E9A46 20 1C                       MOVS    R0, R4
text:000E9A48 D6 F7 54 F8                 BL      __cxa_guard_acquire
text:000E9A4C 00 28                       CMP     R0, #0
text:000E9A4E F7 D0                       BEQ     loc_E9A40
text:000E9A50 20 1C                       MOVS    R0, R4
text:000E9A52 D6 F7 6D F9                 BL      __cxa_guard_release
text:000E9A56 06 48                       LDR     R0, =(off_13E0C4 - 0xE9A64)
text:000E9A58 06 49                       LDR     R1, =(sub_E98A8+1 - 0xE9A60)
text:000E9A5A 07 4A                       LDR     R2, =(unk_13D000 - 0xE9A62)
text:000E9A5C 79 44                       ADD     R1, PC          ; sub_E98A8
text:000E9A5E 7A 44                       ADD     R2, PC          ; unk_13D000
text:000E9A60 78 44                       ADD     R0, PC          ; off_13E0C4
text:000E9A62 D4 F7 5D FE                 BL      sub_BE720
text:000E9A66 EB E7                       B       loc_E9A40
text:000E9A66                ; End of function sub_E9A30
```

C程序逆向分析

- for循环语句反汇编代码的特点
- if...else分支语句反汇编代码的特点
- while循环语句反汇编代码的特点
- switch分支语句反汇编代码的特点

Android NDK JNI API逆向分析

- JNI是Java调用Native机制，是Java语言自己的特性，全称为 Java Native Interface
- Jni中C++和Java的参数传递

Jni中C++和Java的参数传递

- 在JAVA程序中，需要在类中声明所调用的库名称：
  - ✓ **static {**

    System.loadLibrary(“locSDK6a”);

  - ✓ **}**

Jni中C++和Java的参数传递

- 在JAVA代码中声明native函数

public final class Jni{

```
private static native String a(byte[] paramArrayOfByte, int paramInt);

private static native String b(double paramDouble1, double paramDouble2, int paramInt1, int paramInt2);

private static native String c(byte[] paramArrayOfByte, int paramInt);
}
```

Jni中C++和Java的参数传递

- 在JAVA中调用C++函数

```
try
{
  String str = a(arrayOfByte2, 132456);
  return str;
}
```

```
private static native String a(byte[] paramArrayOfByte, int paramInt);
```

```
Java_com_      _location_Jni_a                    00004848
Java_com_      _location_Jni_b                    00004910
Java_com_      _location_Jni_c                    00004A5C
Java_com_      _location_Jni_ee                   000047D0
Java_com_      _location_Jni_encodeNotLimit       00004758
Java_com_      _location_Jni_encrypt              00004D38
Java_com_      _location_Jni_f                    00004C0C
Java_com_      _location_Jni_g                    00004B10
Java_com_      _location_Jni_ib                   00004DB4
Java_com_      _location_Jni_murmur               00004708
Java_com_      _location_Jni_sky                  000046B0
```

ARM汇编代码

这里传递进来的参数四个参数是通过R0-R3寄存器传递的

```
.text:00004848                 EXPORT Java_com_      _location_Jni_a
.text:00004848 Java_com_.      _location_Jni_a       ; CODE XREF: Java_com_      _se
.text:00004848                                       ; DATA XREF: LOAD:stru_564to
.text:00004848
.text:00004848 var_828         = -0x828
.text:00004848 var_824         = -0x824
.text:00004848 s               = -0x81C
.text:00004848 var_42C         = -0x42C
.text:00004848
.text:00004848 ; __unwind {
.text:00004848                 PUSH    {R4-R7,LR}
.text:0000484A                 LDR     R4, =0xFFFFF7EC
.text:0000484C                 LDR     R1, =stru_804.st_size
.text:0000484E                 MOVS    R6, R2
.text:00004850                 ADD     SP, R4
.text:00004852                 LDR     R4, =(__stack_chk_guard_ptr - 0x485C)
.text:00004854                 STR     R3, [SP,#0x828+var_828]
.text:00004856                 ADD     R1, SP
.text:00004858                 ADD     R4, PC          ; __stack_chk_guard_ptr
.text:0000485A                 LDR     R4, [R4]        ; __stack_chk_guard
.text:0000485C                 MOVS    R2, #0x400      ; n
.text:00004860                 LDR     R3, [R4]
.text:00004862                 MOVS    R5, R0
.text:00004864                 ADD     R0, SP, #0x828+s ; s
.text:00004866                 STR     R3, [R1]
.text:00004868                 MOVS    R1, #0          ; c
.text:0000486A                 BLX     memset
.text:0000486E                 ADD     R0, SP, #0x828+var_42C
.text:00004870                 MOVS    R2, #0x80
.text:00004872                 ADDS    R0, #0x10       ; s
.text:00004874                 MOVS    R1, #0          ; c
.text:00004876                 LSLS    R2, R2, #3
.text:00004878                 BLX     memset
.text:0000487C                 STR     R4, [SP,#0x828+var_824]
.text:0000487E                 CMP     R6, #0
.text:00004880                 BEQ     loc_48A6
.text:00004882                 LDR     R2, [R5]
```

函数名

ARM参数传递

```
int __fastcall Java_com_     _location_Jni_a(int a1, int a2, int a3, int a4)
{
  int v4; // r6
  int v5; // r5
  const void *v6; // r7
  size_t v7; // r2
  int v9; // [sp+0h] [bp-828h]
  char s; // [sp+Ch] [bp-81Ch]
  int v11; // [sp+40Ch] [bp-41Ch]

  v4 = a3;
  v9 = a4;
  v5 = a1;
  memset(&s, 0, 0x400u);
  memset(&v11, 0, 0x400u);
  if ( v4 )
  {
    v6 = (const void *)(*(int (__fastcall **)(int, int, _DWORD))(*(_DWORD *)v5 + 736))(v5, v4, 0);
    v7 = (*(int (__fastcall **)(int, int))(*(_DWORD *)v5 + 684))(v5, v4);
  }
  else
  {
    v7 = 0;
    v6 = 0;
  }
  memcpy(&s, v6, v7);
  encode((char *)&v11, &s, v9);
  (*(void (__fastcall **)(int, int, const void *, _DWORD))(*(_DWORD *)v5 + 768))(v5, v4, v6, 0);
  return (*(int (__fastcall **)(int, int *))(*(_DWORD *)v5 + 668))(v5, &v11);
}
```

```
int __fastcall Java_com_     _location_Jni_a(JNIEnv *a1, int a2, int a3, int a4)
{
  int v4; // r6
  JNIEnv *v5; // r5
  const void *v6; // r7
  size_t v7; // r2
  int v9; // [sp+0h] [bp-828h]
  char s; // [sp+Ch] [bp-81Ch]
  int v11; // [sp+40Ch] [bp-41Ch]

  v4 = a3;
  v9 = a4;
  v5 = a1;
  memset(&s, 0, 0x400u);
  memset(&v11, 0, 0x400u);
  if ( v4 )
  {
    v6 = (const void *)((int (__fastcall *)(JNIEnv *, int, _DWORD))(*v5)->GetByteArrayElements)(v5, v4, 0);
    v7 = ((int (__fastcall *)(JNIEnv *, int))(*v5)->GetArrayLength)(v5, v4);
  }
```

此处在C里面调用了Java函数

推荐两本书


Android
软件安全与逆向分析
国内第一本Android软件安全书
看雪论坛
安卓巴士
图灵经典


iOS
应用逆向工程
iOS App Reverse Engineering
Second Edition