Android 游戏案例分析

周坤[1], 潘振忠[2], and 张昱[3]

中国科学技术大学 计算机科学与技术学院

January 27, 2015

[1]984040887@qq.com
[2]pzz2011@mail.ustc.edu.cn
[3]yuzhang@ustc.edu.cn, 0551-63603804

目录

1 APK 结构 **1**

1.1 dex 文件结构 2

1.2 dex 文件反编译工具 2

2 Android 游戏软件案例分析概述 **5**

3 游戏软件案例分析: dex 文件解析 **6**

3.1 游戏软件案例介绍 6

3.2 数据分析 6

3.3 总结 10

4 基于 cocos2d-x 的游戏软件案例分析：so 文件解析 **10**

4.1 cocos2d-x 游戏库函数 11

4.2 Lua VM 提供的 API 12

4.3 系统级 JNI 本地函数 12

4.3.1 以 JNI 结尾的函数的具体分析 14

4.4 C/C++ 层与 Lua 之间的绑定函数 16

4.5 应用级 JNI 本地函数 16

4.6 总结 17

5 问题 **18**

1 APK 结构

APK 文件：Android application package 文件。每个要安装到 Android 平台的应用都要被编译打包为一个单独的文件，后缀名为.apk，其中包含了应用的二进制代码、资源、配置文件等。

apk 文件实际是一个 zip 压缩包，可以通过解压缩工具解开。一般解压后的结构如表 1所示：

表 1: apk 组成

文件或目录	作用
assets/	存放资源文件的目录
META-INF/	从 java jar 文件引入的描述包信息的目录
res/	存放资源文件的目录
libs/	如果存在的话，存放的是 ndk 编出来的 so 库
AndroidManifest.xml	程序全局配置文件
classes.dex	最终生成的 dalvik 字节码
resources.ars	编译后的二进制资源文件

- assets 目录
  assets 目录可以存放一些配置文件，这些文件的内容在程序运行过程中可以通过相关的 API 获得。
- META-INF 目录
  META-INF 目录下存放的是签名信息，用来保证 apk 包的完整性和系统的安全。在 eclipse 编译生成一个 apk 包时，会对所有要打包的文件做一个校验计算，并把计算结果放在 META-INF 目录下，这就保证了 apk 包里的文件不能被随意替换。
- res 目录
  res 目录存放资源文件，包括图片，字符串等等。
- libs 目录
  lib 目录下的子目录 armeabi 存放的是一些 so 文件。eclipse 在打包的时候会根据文件名的命名规则（lib****.so）去打包 so 文件，开头和结尾必须分别为”lib” 和”.so”，否则是不会打包到 apk 文件中的。
- AndroidManifest.xml
  该文件是每个应用都必须定义和包含的，它描述了应用的名字、版本、权限、引用的库文件等等信息，如要把 apk 上传到 Google Market 上，也要对这个 xml 做一些配置。在 apk 中的 AndroidManifest.xml

是经过压缩的，可以通过 AXMLPrinter2 工具解开，具体命令为：java -jar AXMLPrinter2.jar AndroidManifest.xml。

- classes.dex 文件
  classes.dex 是 java 源码编译后生成的 java 字节码文件。但由于 Android 使用的 dalvik 虚拟机与标准的 java 虚拟机是不兼容的，dex 文件与 class 文件相比，不论是文件结构还是 opcode 都不一样。目前常见的 java 反编译工具都不能处理 dex 文件。Android 模拟器中提供了一个 dex 文件的反编译工具，dexdump。
- resources.ars 文件
  编译后的二进制资源文件。

补充说明：

res 和 assets 文件夹来存放不需要系统编译成二进制的文件，例如字体文件等，两者的区别：

1.assets: 不会在 R.java 文件下生成相应的标记，assets 文件夹可以自己创建文件夹，必须使用 AssetsManager 类进行访问，存放到这里的资源在运行打包的时候都会打入程序安装包中。

2.res：会在 R.java 文件下生成标记，这里的资源会在运行打包操作的时候判断哪些被使用到了，没有被使用到的文件资源是不会打包到安装包中的。

1.1 dex 文件结构

dex 文件格式如表 2所示, 具体参考 dalvik 虚拟机源代码介绍官网 https://source.android.com/devices/tech/dalvik/dex-format.html。

1.2 dex 文件反编译工具

表 3列出了几种关于 dex 文件反编译的工具。

- dex2jar
  将 classes.dex 拷贝到 dex2jar 目录下，使用命令：
  ./dex2jar.sh classes.dex
  得到 classes_dex2jar.jar 文件
  打开 jdgui，导入文件 classes_dex2jar.jar，就可以查看反编译的 jar 文件。

表 2: dex 文件结构

名称	格式	描述
header	header_item	文件头
string_ids	string_id_item[]	字符串标识符列表。这里标志了当前 DEX 文件所有字符串使用的标志，也有一些内部名称（例如., 类型描述）或者代码段引用的一些常量对象。这个表按照字符串常量进行排序，字符串使用 UTF-16 进行编码（不在本地敏感方式），并且这个表不包含任何复制项。
type_ids	type_id_item[]	类型标识项。这里是此文件中所有类型的标识（类，队列，或者主类型），无论在文件中是否定义。这个表按照字符串标识索引进行排序，它不包含任何复制项。
proto_ids	proto_id_item[]	函数原型标识表。这里定义了此文件中所用引用的原型标识。这个表按照类型标识索引进行排序，并且参数也是通过类型标识索引进行排序。此表不包含复制项。
field_ids	field_id_item[]	区域标识符列表。这里定义了在此文件中的所有区域定义，此表按照类型标识索引进行主排序，并且按照区域名称作为中排序，按照类型作为子排序。并不包含复制项。
method_ids	method_id_item[]	函数标识表。定义了此文件引用的方法标识。按照类型标识索引作为主排序，按照方法名作为辅排序，按照方法原型作为子排序。不包含复制项。
class_defs	class_def_item[]	类标识表。引用这些类的父类以及接口前必须要经过排序。并且此列表中不能出现重复的类名。
data	ubyte[]	以上表列出的数据，在此进行保存。
link_data	ubyte[]	静态链接数据段。如果此节为空则没有静态链接文件。

表 3: dex 反编译工具

工具	描述
dex2jar	将 classes.dex 反编译出 jar 文件，即 apk 的源程序文件的 java 字节码，然后可以使用 jdgui 查看 dex2jar 反编译出的 jar 文件
Dedexer	可以读取 dex 格式的文件，生成一种类似于汇编语言的输出。这种输出与 jasmin 的输出相似，但包含的是 Dalvik 的字节码
androguard	androguard 是基于 python 的，将 apk 文件中的 dex 文件，类，方法等都映射为 python 的对象
dexdump	Android 自身提供的一个 dex 文件的反编译工具
DexAnalyzer	基于 dexdump 修改之后的 dex 文件解析工具，能够提取 dex 文件中的类和方法，并作相应的分类

- Dedexer

  Dedexer 的网站：http://dedexer.sourceforge.net。
- androguard

  网站：http://code.google.com/p/androguard/wiki/Installation

  安装完成后 androguard 目录下的所有 py 文件都是一个工具，然后可以利用这些工具分析 apk 文件获取相应的信息，比如 androdd.py 用来生成 apk 文件中每个类的方法的调用流程图；androlyze.py 是一个强大的静态分析工具，它提供的一个独立的 Shell 环境来辅助分析人员执行分析工作。在终端提示符下执行“./androlyze.py -s”会进入 androlyze 的 Shell 交互环境，分析人员可以在其中执行不同的命令，来满足不同情况下的分析需求。androlyze.py 通过访问对象的字段与方法的方式来提供反馈结果，分析过程中可能会用到 3 个对象：apk 文件对象、dex 文件对象、分析结果对象。这 3 个对象是通过 androlyze.py 的 Shell 环境（以下简称 Shell 环境）来获取的。
- dexdump

  在 android SDK 中提供了一个名为 dexdump 的工具，可以打印 DEX 文件的信息, 它的源代码在 dalvik 目录中。dexdump 工具提供了一些命令行参数来帮助用户输出相应的 dex 文件信息，相应的参数如下：

  -c：验证 DEX 文件的校验和

  -d：反汇编代码段

-f：显示文件头摘要
-h：显示文件头详细信息
-i：忽略文件校验
-l：输出格式，可以是'plain' 或者'xml' 格式
-m：打印出寄存器图
-t：临时文件名称

- DexAnalyzer
  DexAnalyzer 是我们在 dexdump 基础上做修改之后的 dex 文件解析工具，除了原有的 dexdump 所能输出的信息之外新增了以下信息的输出：
  -a ：表示要进行方法的输出
  -s：输出程序自定义的方法以及类
  -k：输出程序中的库方法及对应的类
  -n：输出 native 方法及对应的类

2 Android 游戏软件案例分析概述

本文档中对于 Android 游戏软件的分析主要是基于以下两个方向进行，一个是从 dex 文件本身入手，利用 dex 文件解析结果进行案例分析，参见第3节；另外一个是从游戏软件包含的动态链接库入手，分析动态链接库包含的信息，参见第4节。

我们对游戏软件分析的主要过程是：首先对需要进行分析的游戏软件进行信息收集，包括游戏软件 apk 文件的大小，其中包含的 dex 文件的大小、动态链接库及其大小，参见第3.1节；然后基于 DexAnalyzer 对游戏软件的 dex 文件进行解析，统计游戏软件中自定义方法、库方法和自定义的 native 方法以及这些方法所属的类信息，并作相应的总结，参见第3.2节。

dex 文件解析是使用我们自己编写的 DexAnalyzer 工具对 dex 文件进行解析。DexAnalyzer 能统计游戏软件的 dex 文件中调用的各类方法及其从属的类信息，各类方法主要分为游戏软件自身定义的方法和使用的库方法，对于自定义方法进一步识别出其中的 native method。

动态链接库分析是利用 objdump 工具对游戏软件的动态链接库文件 (.so 文件) 所包含的函数等进行深入分析。我们将其中的函数分成五类：游戏引擎库函数、Lua 运行时函数、系统级 JNI 本地函数、C++ 与 Lua 之间的绑定函数、应用级 JNI 本地函数，参见第4.1节到第4.5节；然后就每一类函数进行分析总结。

本文档中以 MoonWarriors 游戏软件为例，对其 classes.dex 文件和动态链接库 libgame.so 进行分析。依据 dex 文件解析数据，我们发现有关 lua 的函数没有出现在 dex 字节码中；随后从动态链接库文件中查看，这些 lua 函数的实现均包含在动态链接库中，参见第4.2节和第4.4节。而对于 native 方法，在 dex 字节中出现但没有具体的实现，其实现也是包含在动态链接库中，参见4.5。

通过第3节和第4节对 MoonWarriors 游戏软件的分析使得我们对 Android 游戏软件中调用的类和方法的信息有了初步的了解，在第3.3节和第4.6节给出初步的总结。

3 游戏软件案例分析：dex 文件解析

3.1 游戏软件案例介绍

表 4: 游戏软件信息

软件名称	apk 大小	dex 文件大小	lib 目录	
			so 文件名	so 文件大小
MoonWarriors	10.5 MB	83.9 KB	libgame.so	8.5 MB
Temple Run	116.9 MB	1.1 MB	libmono.so	84.0 KB
			libunity.so	43.0 KB

我们对表 4中列的两个游戏软件进行案例分析。

- MoonWarriors
  MoonWarriors 是一个使用 Cocos2d-x LUA 开发的类似雷电战机的游戏 Demo。源代码发布在 Cocos2d-x 官网的引擎示例当中：http://cn.cocos2d-x.org/tutorial/show?id=2254。
- Temple Run
  神庙逃亡（Temple Run）是由 Imangi Studios 开发的没有终点的动作类视频游戏, 在 Android 平台采用统一的 Unity 游戏引擎。源代码位于http://temple-run.cn.uptodown.com/android。

3.2 数据分析

我们使用 DexAnalyzer 对 2 个游戏软件的 classes.dex 文件进行解析，解析得到的文件位于本调研报告所在目录的 result 子目录下。游戏软件的 dex 文件的解析结果保存在对应游戏名字的目录下面，其中：

- libMethod.txt 是库方法及其对应类的信息；
- userMethod.txt 是程序自定义方法及其对应类的信息;
- nativeMethod.txt 是 native 方法对应的类及其对应类的信息。

根据解析结果文件，统计得到如表 5所示的数据。

表 5: 游戏软件 dex 解析数据

游戏软件	方法信息		类信息	
	方法	数目	类	数目
MoonWarriors	lib method	299	lib class	87
	self-define method	385	self-define class	61
	native method	21	native class	6
	methods sum	684	classes sum	148
Temple Run	lib method	1748	lib class	412
	self-define method	7214	self-define class	1190
	native method	99	native class	10
	methods sum	8962	classes sum	1602

下面以 MoonWarriors 为例对表 5中的数据进行说明，MoonWarriors 的 dex 字节码调用的库方法有 299 个，这些库方法分别属于 87 个类；程序自定义的方法有 385 个，分别归属 61 个类；在这些自定义方法中有 21 个 native 方法，归属到 6 个类。

我们进一步对 MoonWarriors 调用的库方法进行总结，将其所属的类进行归类，结果如表 6所示。

Android 核心库中主要有 4 类 API：Java 标准 API(java 包)、Java 扩展 API(javax 包)、Android 包以及企业和组织提供的 Java 类库 (org 包等)。依据表 **??**的分类，MoonWarriors 游戏软件调用的库方法中没有涉及到对 java 扩展包的引用，第三方的包中只调用了 org/json/JSONObject 的 toString() 方法。JSON(JavaScript Object Notation) 是一种轻量级的数据交换格式, 介绍 JSON 的网站：http://www.json.org/json-zh.html。库方法的调用主要集中在 Android 包以及 java 包中方法的调用。

表 6: 库方法分类

类库	包路径 (所含类数)	包含的类
android(58)	android/app (4)	Activity, AlertDialog$Builder, AlertDialog, Dialog
	android/content (4)	Context, Intent, SharedPreferences$Editor, SharedPreferences
	android/content/res (3)	AssetFileDescriptor, AssetManager, Resources
	android/database (1)	Cursor
	android/database/sqlite (2)	SQLiteDatabase, SQLiteOpenHelper
	android/graphics (5)	Bitmap, Canvas, Paint, Rect, Typeface
	android/graphics/drawable (1)	ColorDrawable
	android/hardware (2)	Sensor, SensorManager
	android/media (2)	MediaPlayer, SoundPool
	android/net (5)	ConnectivityManager, NetworkInfo, Uri, wifi/WifiInfo, wifi/WifiManager
	android/opengl (3)	ETC1Util$ETC1Texture, ETC1Util, GLSurfaceView
	android/os (4)	Handler, Message, Process, Vibrator
	android/telephony (1)	TelephonyManager
	android/text (4)	Editable, InputFilter$LengthFilter, TextPaint, TextUtils
	android/util (3)	DisplayMetrics, FloatMath, Log
	android/view (7)	Display, KeyEvent, MotionEvent, ViewGroup$LayoutParams, Window, WindowManager, inputmethod/InputMethodManager
	android/widget (7)	EditText, FrameLayout, LinearLayout$LayoutParams, LinearLayout, ProgressBar, RelativeLayout$LayoutParams, TextView
java 包 (28)	java/io (4)	File, FileInputStream, InputStream, PrintStream
	java/lang (10)	CharSequence, Class, Exception, Integer, Math, Object, String, StringBuilder, System, ref/WeakReference
	java/net (2)	HttpURLConnection, URL
	java/nio (2)	ByteBuffer, ByteOrder
	java/util (10)	ArrayList, HashMap, Iterator, LinkedList, Locale, Map$Entry, Map, Set, Vector, concurrent/Semaphore
第三方 (1)	org/json (1)	JSONObject

MoonWarriors 游戏软件中程序自定义方法调用的 native method 较少，只有 21 个 native 方法，具体如下所示：

```
1  Lorg/cocos2dx/lib/Cocos2dxAccelerometer;.onSensorChanged:(FFFJ)V
2  Lorg/cocos2dx/lib/Cocos2dxBitmap;.nativeInitBitmapDC:(II[B)V
3  Lorg/cocos2dx/lib/Cocos2dxETCLoader;.nativeSetTextureInfo:(II[BI)V
4  Lorg/cocos2dx/lib/Cocos2dxHelper;.nativeSetApkPath:(Ljava/lang/String
       ;)V
5  Lorg/cocos2dx/lib/Cocos2dxHelper;.nativeSetEditTextDialogResult:([B)V
6  Lorg/cocos2dx/lib/Cocos2dxLuaJavaBridge;.callLuaFunctionWithString:(
       ILjava/lang/String;)I
7  Lorg/cocos2dx/lib/Cocos2dxLuaJavaBridge;.
       callLuaGlobalFunctionWithString:(Ljava/lang/String;Ljava/lang/
       String;)I
8  Lorg/cocos2dx/lib/Cocos2dxLuaJavaBridge;.releaseLuaFunction:(I)I
9  Lorg/cocos2dx/lib/Cocos2dxLuaJavaBridge;.retainLuaFunction:(I)I
10 Lorg/cocos2dx/lib/Cocos2dxRenderer;.nativeDeleteBackward:()V
11 Lorg/cocos2dx/lib/Cocos2dxRenderer;.nativeGetContentText:()Ljava/lang/
       String;
12 Lorg/cocos2dx/lib/Cocos2dxRenderer;.nativeInit:(II)V
13 Lorg/cocos2dx/lib/Cocos2dxRenderer;.nativeInsertText:(Ljava/lang/
       String;)V
14 Lorg/cocos2dx/lib/Cocos2dxRenderer;.nativeKeyDown:(I)Z
15 Lorg/cocos2dx/lib/Cocos2dxRenderer;.nativeOnPause:()V
16 Lorg/cocos2dx/lib/Cocos2dxRenderer;.nativeOnResume:()V
17 Lorg/cocos2dx/lib/Cocos2dxRenderer;.nativeRender:()V
18 Lorg/cocos2dx/lib/Cocos2dxRenderer;.nativeTouchesBegin:(IFF)V
19 Lorg/cocos2dx/lib/Cocos2dxRenderer;.nativeTouchesCancel:([I[F[F)V
20 Lorg/cocos2dx/lib/Cocos2dxRenderer;.nativeTouchesEnd:(IFF)V
21 Lorg/cocos2dx/lib/Cocos2dxRenderer;.nativeTouchesMove:([I[F[F)V
```

这些 native 方法均对应于与游戏引擎 cocos2dx 相关的 6 个类, 包括加速器、位图、渲染相关的类：

- org/cocos2dx/lib/Cocos2dxAccelerometer: 主要是提供 sensor 的监测
- org/cocos2dx/lib/Cocos2dxBitmap：主要是创建 TextBitmap
- org/cocos2dx/lib/Cocos2dxETCLoader: cocos2d-x 对 etc1 图片显示的支持，etc1 图片是 android 下通用的压缩纹理
- org/cocos2dx/lib/Cocos2dxHelper：被 ndk 层创建 Cocos2dxAccelerometer、Cocos2dxMusic、Cocos2dxSound，并提供相关的操作，同时提供一些 Cocos2dxPrefsFile.Xml 的配置信息
- org/cocos2dx/lib/Cocos2dxLuaJavaBridge: LUA 与 java 互调
- org/cocos2dx/lib/Cocos2dxRenderer：利用 androidGLSurfaceview 的生命周期创造 cocos2d 的生命周期，并画图

http://blog.csdn.net/jacklam200/article/details/8965457中介绍 android 程序和 cocos2dx 协作，比如 Java_org_cocos2dx_lib_Cocos2dxHelper 用于衔接 org.cocos2dx.lib.Cocos2dxHelper 类。

3.3 总结

通过对比两个游戏的统计数据，对于大型一点的游戏，比如 Temle Run，其中涉及到的库方法与自定义方法都有几千个，人工从其中辨别哪些方法可以优化是很困难的。要想从库方法入手进行优化还需要进行其他方面的调研来总结是否有可以优化的地方。

游戏软件中，自定义方法中调用的 native method 的方法相对来说比较少，基本与游戏引擎以及第三方提供的游戏插件有关，从这些方法中进行人工分析的难度低得多，但仍然需要对这些 native method 方法特征进行进一步分析。

另外需要考虑的一个方面是，游戏软件中方法的调用之间是否存在一定的模式，从调用模式入手思考可以优化的点。

4 基于 cocos2d-x 的游戏软件案例分析：so 文件解析

我们以 MoonWarriors 软件包含的动态链接库 libgame.so 为例，对该动态链接库包含的函数等进行深入分析。我们用 objdump 命令，从 libgame.so 中导出符号表。符号表中的信息可以参见 https://ring0.me/2014/12/linkers-loaders-library-notes/文章最后的表格。我们将其中的函数分成如下五类，同时结合 cocos2d-x 3.3 版本的源码分析这几类函数在源码中的位置，依次在下面各小节中分别说明:

1. **游戏引擎库函数**：指由 cocos2d-x 引擎提供的游戏相关的库所包含的函数，参见第4.1节；
2. Lua **运行时函数**：Lua 运行时环境（Lua 虚拟机）提供的库函数，参见第4.2节；
3. **系统级 JNI 本地函数**：cocos2d-x 中针对 android 系统中的 JNI 函数给出的本地函数实现，参见第4.3节；
4. C++ **与 Lua 之间的绑定函数**：cocos2d-x 用于方便沟通 C/C++ 层与 Lua vm 之间绑定 (binding) 所使用的、函数名以 tolua 开头的函数（一个自动 bind 工具），参见第4.4节；
5. **应用级 JNI 本地函数**：以 Java_org_cocos2dx_lib 打头的函数，这些是针对 cocos2d 以及游戏应用的 JNI 函数给出的本地函数实现，参见

第4.5节；

表 7列出了 libgame.so 中涉及的 5 类函数的个数及所占比例。

表 7: 各类符号的数目及比例

类别	该类符号的数目	100%
cocos2d-x 游戏库函数	8431	52.69%
lua vm 提供的 API	74	0.45%
C/C++ 与 lua 之间的绑定函数	79	0.4808%
与 JNI 相关的函数 (不含接口 JNI)	69	0.42%
接口 JNI	21	0.128%
总函数数目	16431	100%

4.1 cocos2d-x 游戏库函数

这类函数是指由 cocos2d-x 引擎提供的游戏相关的库所包含的函数，主要位于 cocos2d-x-3.3/cocos/base 目录下。

首先根据 objdump -T libgame.so|grep ".text" > _text.txt
命令得到文件_text.txt，此文件中的每一行的最后一项均表示一个游戏在运行过程中可能用到的函数名。然后我们查看该文件，发现共有 16431 行，也就是说游戏运行过程中可能用到 16431 个函数。

而通过执行:

cat _text.txt | grep "_Z" > _z.txt

cat _z.txt | grep "cocos2dx" > cocos2dx.txt

查看得到的 cocos2dx.txt，发现共有 8431 行，可以推出 cocos2d-x 游戏引擎提供了 8431 个库函数，这些函数都是经过编译器去除面向对象特性后得到的函数。例如：

```
1 _ZTSN7cocos2d9extension9CCBReaderE
2 _ZN7cocos2d10CCDirector13setProjectionENS_20ccDirectorProjectionE
3 _ZTVN7cocos2d8CCActionE
4 _ZTIN7cocos2d8CCLiquidE
5 _ZNK7cocos2d13CCDictElement9getStrKeyEv
6 _ZN7_JNIEnv16CallObjectMethodEP8_jobjectP10_jmethodIDz
```

这些函数的函数名特点如下:

以_ZN7_JNIEnv16CallObjectMethodEP8_jobjectP10_jmethodIDz 为例，它就是编译器将名字空间 JNIEnv 中的 CallObjectMethod(unsigned char，

_jobject* ,_jmethodID*) 函数去除面向对象特征转换后得到的能唯一标识的符号名字。

1. _Z 是所有改名后的符号的开始符号；
2. N 表示其后跟的是 namespace 或者类名；
3. 7_JNIEnv 表示长度为 7 个字符的类名或名字空间_JNIEnv；
4. 16CallObectMethod 表示由 16 个字符组成的函数名；
5. 再接下去就是函数参数类型列表，依次描述每个参数的类型：
   - E 表示第 1 个参数为 unsigned char 类型；
   - P 表示指针类型，由于其后是 8_jobject，故第 2 个参数类型为_jobject *；
   - 同理有第 3 个参数类型为_jmethodID *；
   - z 表示参数类型列表的结束。

如果想了解这类函数名的具体信息，可以查看我们提供的 cocos.txt 文件(位于本调研文档所在目录中的 result/MoonWarriors 文件夹中)。这类函数应该不止这些，还有一些未包含 cocos 关键字，这里没有列举出来。

4.2 Lua VM 提供的 API

Lua VM 提供的 API 的实现源码位于 cocos2d-x-3.3/external/lua/lua 目录下, 这些函数是 Lua 提供的接口，帮助 lua 成为一门小巧方便的嵌入语言。举例如下:

```
1  luaL_ref
2  luaL_unref
3  luaL_checkany
4  luaopen_table
5  luaopen_ffi
6  luaopen_os
7  luaopen_io
8  ...
```

该类函数名形式为:luaL_* 或 luaopen_*，在 libgame.so 中出现的这类函数的函数名信息可参见 luaopen_prefix.txt 和 lua_prefix.txt 文件 (位于本调研文档所在目录中的 result/MoonWarriors 文件夹中)。

4.3 系统级 JNI 本地函数

系统级 JNI 本地函数是 cocos2d-x 中针对 android 系统中的 JNI 函数给出的本地函数实现，这类函数的实现源码位于 cocos2d-x-3.3/cocos/

platform/android 目录下。利用 objdump 查看 libgame.so 中可得到相关的这类函数的部分符号信息如下：

```
001f12d0 g    DF .text  00000068 openKeyboardJNI
001f1338 g    DF .text  00000068 closeKeyboardJNI
0021e540 g    DF .text  00000054 endJNI
0021e0d4 g    DF .text  00000080 preloadBackgroundMusicJNI
0021e154 g    DF .text  00000088 playBackgroundMusicJNI


001efe48  w   DF .text  00000018 _ZN7_JNIEnv12NewStringUTFEPKc
001f007c g    DF .text  000000f0
        _Z21showEditTextDialogJNIPKcS0_iiiiPFvS0_PvES1_
001f01cc g    DF .text  000000ac _Z17getPackageNameJNIv
001f0598 g    DF .text  0000009c _Z16getBoolForKeyJNIPKcb
001f0634 g    DF .text  00000094 _Z19getIntegerForKeyJNIPKci
001f06c8 g    DF .text  0000009c _Z17getFloatForKeyJNIPKcf
001f0764 g    DF .text  0000009c _Z18getDoubleForKeyJNIPKcd
001f0800 g    DF .text  00000130 _Z18getStringForKeyJNIPKcS0_
001f0930 g    DF .text  0000008c _Z16setBoolForKeyJNIPKcb
001f09bc g    DF .text  0000008c _Z19setIntegerForKeyJNIPKci
001f0a48 g    DF .text  00000094 _Z17setFloatForKeyJNIPKcf
001f0adc g    DF .text  00000090 _Z18setDoubleForKeyJNIPKcd
001f0b6c g    DF .text  000000a8 _Z18setStringForKeyJNIPKcS0_
001efe10  w   DF .text  00000038
        _ZN7_JNIEnv22CallStaticDoubleMethodEP7_jclassP10\_jmethodIDz
0018cf44 g    DF .text  00000018 JNI_OnLoad
```

上面的符号可以进一步分成如下 3 类:

1. 以 JNI 结尾的函数, 这些函数我们目前推测是由 cocos2d-x 引擎对 android 平台提供的 Java 接口的 C++/C 实现，并且是基于 JNI 机制实现的。
2. 经过编译器为去除面向对象特征而转换得到的函数，暂时还不清楚这些函数的具体作用。例如：

   ```
   _Z19terminateProcessJNIv
   _Z22enableAccelerometerJNIv
   ...
   ```

3. JNI_Onload：当 Android 的 VM(Virtual Machine) 执行到应用软件里的 System.loadLibrary("libgame.so") 函数时，首先会去执行应用软件的动态链接库 libgame.so 里的 JNI_OnLoad() 函数。它的用途如下：1）告诉 VM 此 C 组件使用哪一个 JNI 版本。如果你的 *.so 文件没有提供 JNI_OnLoad() 函数，VM 会默认该 *.so 是使用最老的 JNI 1.1

版本。由于新版的 JNI 做了许多扩充，如果需要使用 JNI 的新版功能，例如 JNI 1.4 的 java.nio.ByteBuffer, 就必须借由 JNI_OnLoad() 函数来告知 VM。

2）由于 VM 执行到 System.loadLibrary() 函数时，就会立即先呼叫 JNI_OnLoad()，所以 C 组件的开发者可以借由 JNI_OnLoad() 来进行 C 组件内的初期值之设定 (Initialization)。

下面我们仅以那些以 JNI 结尾的函数为例来进行进一步的分析。

4.3.1 以 JNI 结尾的函数的具体分析

首先, 我们将这些函数全部罗列如下:

```
getDPIJNI
openKeyboardJNI
closeKeyboardJNI
endJNI
preloadBackgroundMusicJNI
playBackgroundMusicJNI
stopBackgroundMusicJNI
pauseBackgroundMusicJNI
resumeBackgroundMusicJNI
rewindBackgroundMusicJNI
isBackgroundMusicPlayingJNI
getBackgroundMusicVolumeJNI
setBackgroundMusicVolumeJNI
getEffectsVolumeJNI
setEffectsVolumeJNI
playEffectJNI
stopEffectJNI
preloadEffectJNI
unloadEffectJNI
pauseEffectJNI
pauseAllEffectsJNI
resumeEffectJNI
resumeAllEffectsJNI
```

通过对.so 利用 objudmp -D 反汇编，我们可以查看到 libgame.so 文件导出表的所有汇编代码。以 playBackgroundMusicJNI 为例子, 为了得到该函数所调用的函数，从反汇编得到的汇编代码中, 我们只需查看其中的跳转指令。最终，通过查看被调用函数的汇编代码及其跳转指令，我们得到了函数 playBackgroundMusicJNI 调用关系如下：

- playBackgroundMusicJNI
  - _ZN13CocosDenshion17SimpleAudioEngine14stopAllEffectsEv
    - _ZN7cocos2d7CCPointC1Eff
    - _ZN7cocos2d6CCSizeC1Eff
    - _ZN7cocos2d6CCRect7setRectEffff
    - _ZN13CocosDenshion17SimpleAudioEngine14stopAllEffectsEv：即调用自身
    - <android_setCpuArm+0x94c>
    - _ZN7cocos2d9JniHelper9getJavaVMEv
      - pthread_key_create@plt
      - __android_log_print@plt
      - pthread_getspecific@plt
      - _ZN7cocos2d9JniHelper9getJavaVMEv 自身递归
    - __android_log_print@plt
    - 其他一些xxx@plt 函数
  - _ZN7_JNIEnv14DeleteLocalRefEP8_jobject
  - _ZN7_JNIEnv20Cal33lStaticVoidMethodEP7_jclassP10_jmethodIDz
    - .....

总的来说, 函数 playBackgroundMusicJNI 会调用如下三个函数和其自身。

```
1._ZN13CocosDenshion17SimpleAudioEngine14stopAllEffectsEv
2._ZN7_JNIEnv20Cal33lStaticVoidMethodEP7_jclassP10_jmethodID
3._ZN7_JNIEnv14DeleteLocalRefEP8_jobject
```

其中 1、2、3 存在严格的顺序关系，即会先调用 1，然后 2，然后才是 3；而对自身的调用会被插入第 1 个函数与第 2 个函数之间, 或第 2 个函数与第 3 个函数之间或第 3 个函数以后。

深入函数 1._ZN13CocosDenshion17SimpleAudioEngine14stopAllEffectsE, 我们发现该函数调用的函数包括如下 5 个函数以及其自身。

```
_ZN7cocos2d7CCPointC1Eff
_ZN7cocos2d6CCSizeC1Eff
_ZN7cocos2d6CCRect7setRectEffff>
android_setCpuArm+0x94c
_ZN7cocos2d9JniHelper9getJavaVMEv
```

同样的，我们分析 5._ZN7cocos2d9JniHelper9getJavaVMEv, 可以发现该函数会对如下函数及其自身进行调用。

```
pthread_key_create@plt
__android_log_print@plt
pthread_getspecific@plt
```

4.4 C/C++ 层与 Lua 之间的绑定函数

这些函数名的形式为 tolua_XXX. 这些函数是用于将 C/C++ 写的函数自动注册到 Lua VM 中, 供 Lua 脚本语言调用。这些函数的实现源码位于 cocos2d-x-3.3/external/lua/tolua 目录下。

而针对这类函数, 我们也许可以在保持接口不变的情况下, 从逃逸分析的角度给出优化方法。之所以认为这是一种可能的优化策略, 主要是因为在那些用 Lua 编写的代码 (基于 cocos2d-x 引擎) 中存在对 C++ 层对象的调用, 而这类函数会通过 malloc 将其放在堆上。

tolua_XXX 函数例如:

```
tolua_isnoobj
tolua_typename
tolua_classevents
```

4.5 应用级 JNI 本地函数

这类函数是游戏软件中本地函数的具体实现，即是游戏软件的 dex 文件中具有 native 属性的自定义方法的具体实现。而且我们暂时估计这些方法是一些由 cocos2d-x 提供的、用于 cocos2d-x 与 android 平台建立联系的接口函数，所有与引擎相关的 native 函数或者第三方提供的 sqlite，web，socket，ssl，openssl 等 native 函数均通过这些接口函数, 最终作用于 android 平台。

这类函数位于 cocos2d-x-3.3/cocos/platform/android 目录下。

以我们分析的 case 为例，在 dex 文件中，有如下 native function:

```
onSensorChanged
nativeInitBitmapDC
nativeSetTextureInfo
nativeSetApkPath
callLuaFunctionWithString
callLuaGlobalFunctionWithString
releaseLuaFunction
retainLuaFunction
```

```
nativeDeleteBackward
nativeGetContentText
nativeInit
nativeInsertText
nativeKeyDown
nativeOnPause
nativeOnResume
nativeRender
nativeTouchesBegin
nativeTouchesCancel
nativeTouchesEnd
nativeTouchesMove
```

其中的 nativeDeleteBackward 函数对应的 C++ 层函数为:

Java_org_cocos2dx_lib_Cocos2dxRenderer_nativeDeleteBackward 该函数位于 cocos2d-x/cocos/platform/android/jni 目录下的 Java_org_cocos2dx_lib_Cocos2dxRenderer.cpp 文件中

更直观的展示如下:

通过命令 dexdump -d xxx.apk 输出的文件中包含如下一个 native method 的信息

```
    #2              : (in Lorg/cocos2dx/lib/Cocos2dxRenderer;)
      name          : 'nativeDeleteBackward'
      type          : '()V'
      access        : 0x010a (PRIVATE STATIC NATIVE)
      code          : (none)
```

通过如下命令:

```
objdump -T libgame.so |grep "
        Java_org_cocos2dx_lib_Cocos2dxRenderer_nativeDeleteBackward"
```

可以输出

```
001f0d48 <Java_org_cocos2dx_lib_Cocos2dxRenderer_nativeDeleteBackward
        >:
```

因此要使用的 native function 需要在.so 文件和.dex 文件中一一对应。

4.6 总结

通过以上对.so 文件中的符号表进行分类，我们推测，如果要对基于 cocos2d-x 的游戏进行优化，可以只通过对 Lua VM 做优化，保持其对 cocos2d-x 的使用接口不变。对 Lua VM 进行优化可能存在多种方案; 当然

基于整体 cocos2d-x 游戏的优化也未必一定是针对 lua vm 的优化，一切暂时还是未知。

另一方面，为了将优化后的 Lua VM 能快速替换游戏软件中所使用的 Lua VM 版本，我们可以尝试从一个或若干个通过 NDK 编译得到的.so 单独剥离出 liblua.so, 然后替换为我们修改 Lua vm 得到的那份新的 liblua.so。这里提到的 liblua.so 指的都是 Lua vm 编译得到的.so 文件，只是可能需要进行修改。而对其进行剥离也可能会在我们的考虑之内，甚至可能成为要开展的工作之一。

5 问题

dex 文件解析问题：

1.Android 核心库中主要有 4 类 API：Java 标准 API(java 包)、Java 扩展 API(javax 包)、Android 包以及企业和组织提供的 Java 类库 (org 包等)。从 2 个例子解析结果来看游戏软件中调用的库方法在这 4 类 API 都有覆盖，不可能对所有的核心库都做优化。

2. 通过对比两个游戏的统计数据，对于大型一点的游戏，比如 Temle Run，其中涉及到的库方法与自定义方法都有几千个，人工从其中辨别哪些方法可以优化是很困难的。要想从库方法入手进行优化还需要进行其他方面的调研来总结是否有可以优化的地方。

3. 因为我们有一个想法是探寻 Android 游戏软件中 I/O 角度的并行性，所以打算对 dex 文件解析的结果做进一步筛选，只提取输入输出相关的类方法，这个暂时还没有进行，目前还不能给出结论。希望能够了解商业应用中是否针对游戏软件中调用核心库方法有一些关注的焦点或者相关的内容。

cocos2d-x 中 lua 相关的问题:

1. 就目前基于 cocos2d-x 开发的游戏而言，是否有在 linux+android studio 上开发的呢？而新版的 android studio 在 linux 下存在各种问题，在导入工程后，没法将 cocos2d-x 游戏中的需要的一些文件包含进工程中。不知道是不是和 gradle build system 有关？

在 android studio 推出之前，是有 linux+eclipse 的，此 eclipse 当时也可以在 http://developer.android.com/sdk/index.html#Other 找到，但是自从 android studio 推出后，之前的 google 版本的 eclipse 就下架了。作为例证:http://www.cnblogs.com/lonkiss/archive/2012/11/17/2775440.html 该网页可以说明此事。

2. cocos2d-x 中的 lua 解释器代码是否就是 http://www.lua.org/download.html上提供的？如果经过修改，不知道修改过哪些东西？该代码在 cocos2d-x 源码中的路径为 cocos2d-x/external/lua/lua。