**Android虚拟机调试器的原理与实现**

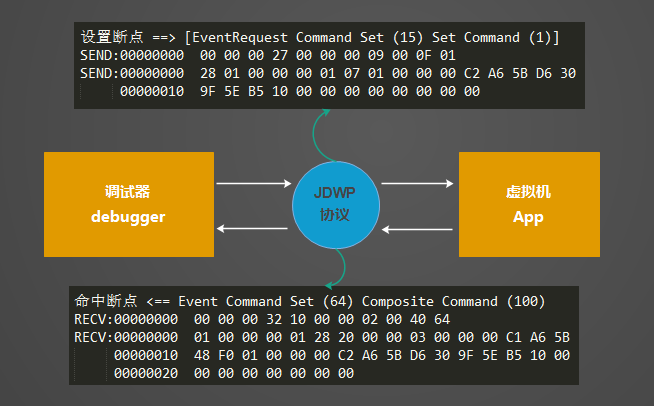
本文主要讲解Android虚拟机动态调试背后涉及到的技术原理，除了JDWP协议细节，还包括任意位置断点、堆栈输出、变量值获取等基础调试功能的具体实现。另外本文提供了一款新的android动态调试工具——AVMDBG，提供调试API接口，支持python脚本扩展。AVMDBG作为android调试技术研究过程中的实验项目，开源出来供学习参考。

1. **Android动态调试方案**

在讲解android动态调试实现之前，先回顾下针对apk进行动态调试常用的几种方法，比较下不同解决方案的特点和存在的问题：

1. smali插桩大法，反编译apk后在关键位置插入smali代码，通过打印日志的形式进行调试跟踪。但是这个方法非常繁琐，除了需要编写smali代码外，每次新增日志输出点都需要重新解包重打包，而且经常会遇到apk保护导致解包失败的问题需要解决。
2. IDA在6.6版本以后支持dex文件的动态调试，但是部分功能并不完善，比如监视寄存器变量的值，需要通过watch窗口手动添加并指定类型，而且单步过程中需要留意类型变化，可能导致虚拟机意外崩溃，导致这个问题的原因在后续文章中会分析到。
3. JEB，堪称apk静态分析的神器，在2.0版本以后开始加入动态调试的功能，最新版本同时支持smali代码和native代码的调试，功能可以说非常强大，如果说缺点可能只有一个，和IDA一样都是商业软件，收费不菲。
4. [andbug](https://github.com/swdunlop/AndBug)开源项目，5年前的项目原作者已经不再维护，原项目只支持linux环境，动态调试关键功能也有缺失，例如不支持任意代码位置断点等。使用过程中也碰到过一些bug，类似shell的交互方式使用起来并不是很顺手。后来有国内开发者[anbc](https://github.com/anbc/AndBug)对该项目进行了完善改造，新增函数调用监控功能支持配置文件。
5. android studio+[smalidea](https://bitbucket.org/JesusFreke/smali/src)插件，也是一个非常不错的调试解决方案，作者通过IDE调试器插件的形式支持smali代码的动态调试，使用过程中偶尔会遇到跳行的bug。smalidea插件同样是开源项目，作者的作品还有非常知名的dex反汇编工具[baksmali](https://bitbucket.org/JesusFreke/smali/src)等。
6. [xposed](http://repo.xposed.info/)类的安卓代码Hook框架，可以通过在函数入口和出口处hook进行调试输出，但同样无法对函数内部代码进行断点输出。类似的项目还有[cydia\_substrate](http://www.cydiasubstrate.com/)、[frida](http://www.frida.re/)、[ddi](https://github.com/crmulliner/ddi)等，也都是非常优秀的开源项目。
7. **JDWP协议简述**

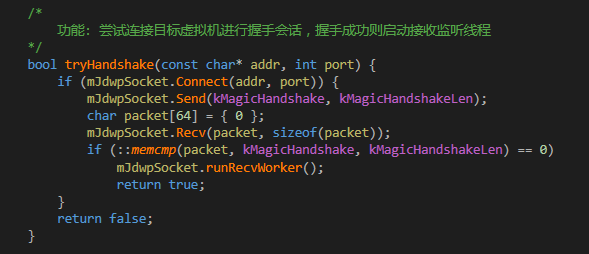
JDWP是Java Debug Wire Protocol (Java调试线协议)的缩写，简单来讲，就是调试器和目标虚拟机进行调试交互的通信协议，调试器通过JDWP协议来获取JAVA虚拟机中程序的信息或控制目标程序运行状态，比如函数断点设置，线程状态、变量值获取等，而目标虚拟机也在指定事件触发时通过JDWP协议通知调试器进行处理，比如断点事件消息、线程创建消息等。



[1] JDWP协议交互简图

Android虚拟机虽然和普通java虚拟机存在不少差异，但是它的调试接口同样是基于JDWP协议的，Dalvik虚拟机JDWP服务端的实现源码位于：[./android/dalvik/vm/jdwp](https://android.googlesource.com/platform/dalvik/+/eclair-release/vm/jdwp/)。android studio、eclipse 等IDE的调试功能和DDMS的监控功能也都是基于JDWP协议实现的。

JDWP协议通信要求首先进行握手会话来表明互相的身份，调试器端发送的与虚拟机返回的数据包内容一样，内容为“JDWP-Handshake”，通过验证以后才可以继续后续的会话。



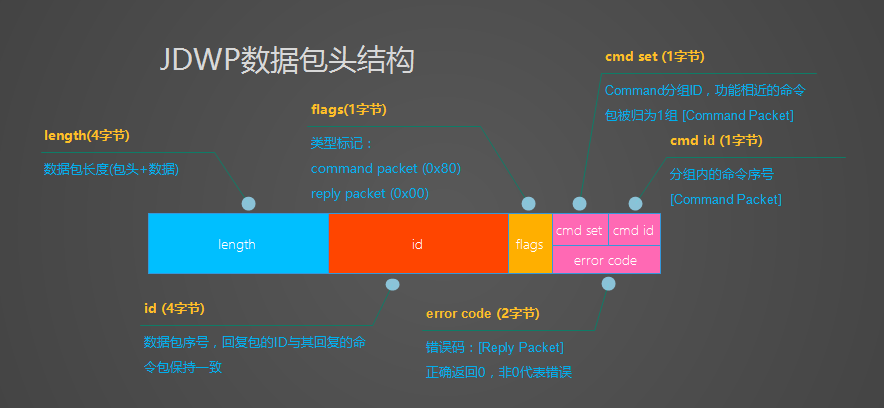
[2] JDWP协议通信前需要进行握手会话

Android虚拟机的JDWP实现支持adb和socket两种通信方式，可以使用adb的jdwp命令进行端口转发绑定，这样就可以通过socket和调试目标进程进行通讯，命令格式为"adb forward tcp:[port] jdwp:[pid]”。

JDWP协议的通信会话主要包含2类数据包，分别为Command packet(命令包)与Reply packet(回复包)：

1. Command packet：调试器发送给虚拟机用于获取程序状态信息或控制程序运行；虚拟机发送给调试器用于通知事件触发消息。
2. Reply packet：虚拟机发送给调试器用于回复命令包的请求或者执行结果。

JDWP数据包主要包含包头和和数据两部分，包头部分格式长度固定，为11字节，数据部分为可变长度，字段结构由其包头指定的类型决定。下面详解包头各字段的含义：

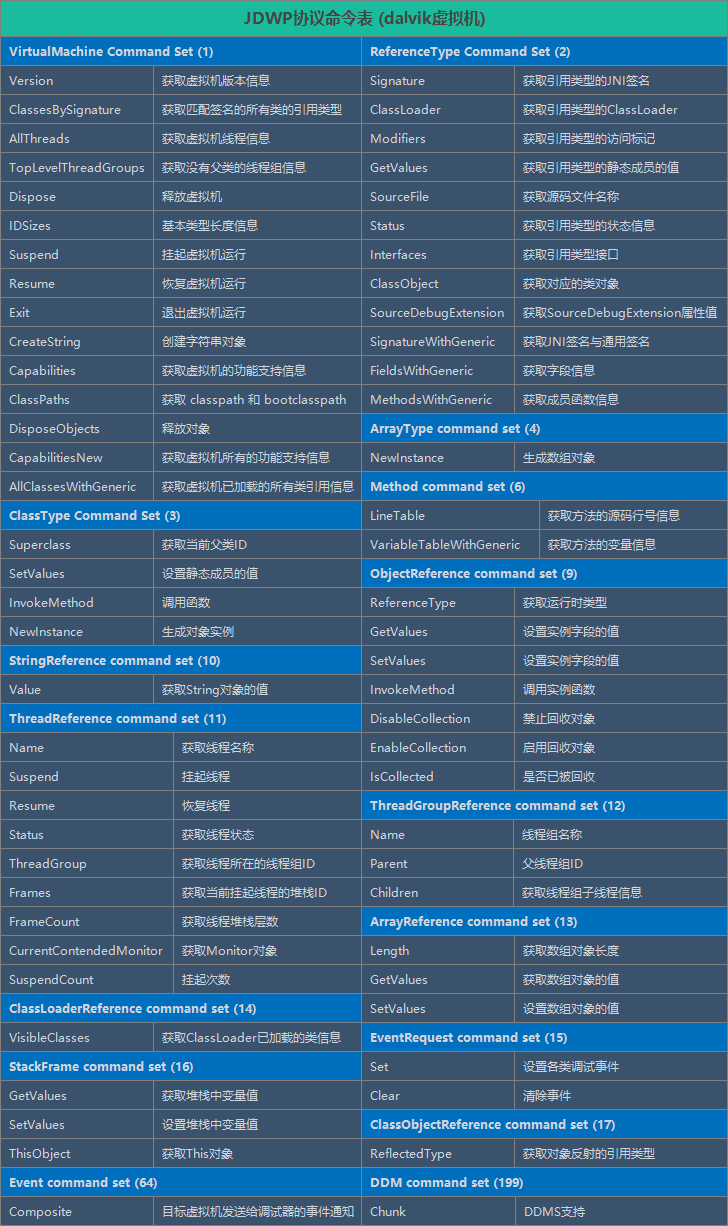


[3] JDWP数据包头结构简图

从上图可以看出，命令包和回复包的包头结构基本一致，区别在于最后2个字节，命令包拆分为2个单字节分别表示命令分组和命令序号，在回复包中则用于表示错误码，非0表示命令执行存在错误。

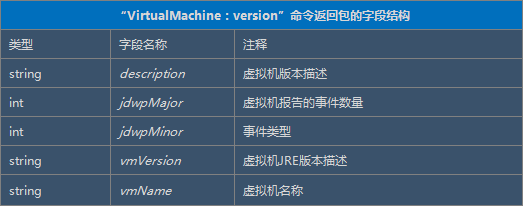
以java7为例说明，JDWP协议按功能划分为18组命令，总计91个不同命令请求，包含了虚拟机、引用类型、对象、线程、方法、堆栈、事件等不同类型的操作命令。JDWP协议支持的命令细节可以参考[官方文档](http://docs.oracle.com/javase/7/docs/platform/jpda/jdwp/jdwp-protocol.html)，这里不再赘述。

Android虚拟机对JDWP协议的支持实现并不完整，当然调试需要的绝大部分关键命令都是支持的，具体信息可以参考安卓dalvik虚拟机源码：[./android/dalvik/vm/jdwp/JdwpHandler.cpp](https://android.googlesource.com/platform/dalvik/+/eclair-release/vm/jdwp/JdwpHandler.c#1902) ，下图是dalvik虚拟机(以android4.2版本为准)支持的命令请求类型表：



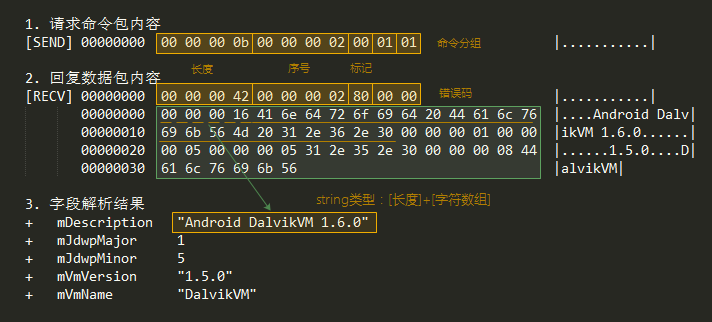
[4] JDWP协议命令表简图 (以android4.2版本支持为准)

虽然上面图表中的命令类型比较多，但JDWP协议本身并不算复杂，按协议标准进行组包请求即可，下面以获取目标虚拟机版本信息的命令“[VirtualMachine：version](http://docs.oracle.com/javase/7/docs/platform/jpda/jdwp/jdwp-protocol.html#JDWP_VirtualMachine_Version)”为例，演示一次JDWP协议的交互通信过程：



[5] “VirtualMachine：version”命令返回数据的注解

从上图的命令注解中可以看出，“VirtualMachine：version”请求命令没有附加数据，回复包数据中包含5个字段，通信数据包解析过程如下图：

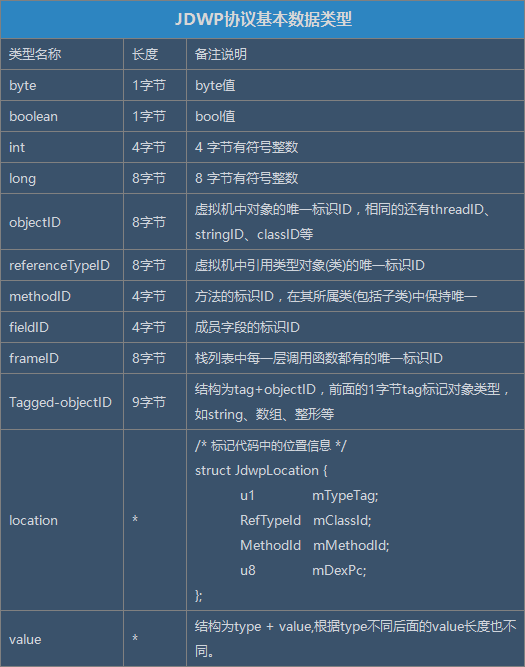


[6] “VirtualMachine：version”命令通信数据包解析

从上图的解析示意中，我们最终获取到目标虚拟机的版本信息，这中间有2点需要注意：

1. 数据包中数据使用大端模式；
2. 基本数据类型的内存结构，例如string，使用[长度]+[字符数据]的形式；

下面我们整理下JDWP协议中使用到的其他基本数据类型，在后续的命令请求与数据包解析中都会频繁使用到。其中有些数据类型的长度是由虚拟机实现定义的，比如ObjectID等，可以通过“VirtualMachine：IDSizes”命令进行获取。下图整理的数据类型说明以Android虚拟机的实现为标准：



[7] JDWP协议使用到基本数据类型

1. **动态调试的核心 —— 任意代码位置断点**

针对简单apk进行逆向时静态分析就足够胜任，但是碰到下面几种情况的时候，“任意代码位置断点”的动态调试分析则更加合适：

1. 需要对代码中的关键参数、变量值进行观察，例如加密后的字符串。
2. 对偏底层的公共函数库进行hook输出，快速筛选定位可疑调用，例如API调用监控等。
3. 在代码规模比较庞大、调用逻辑比较复杂的情况下，需要使用堆栈跟踪对可疑关键点的调用路径梳理确认。

“任意代码位置断点”功能不单指代码断点的调试事件通知，还会涉及到虚拟机栈结构、寄存器使用、参数变量值获取以及堆栈跟踪等方面的功能，但是其中很多知识点限于篇幅无法在本文中全部讲解透彻，这里推荐两篇关于Dalvik虚拟机的文章可以作为扩展阅读：《[深入理解Android之Java虚拟机Dalvik](http://blog.csdn.net/innost/article/details/50377905)》、《[Dalvik虚拟机进程和线程的创建过程分析](http://blog.csdn.net/luoshengyang/article/details/8923484)》。

下面重点讲解断点功能的实现，其中的关键点可以归结为以下几个问题：

1. **如何设置和处理断点事件？**

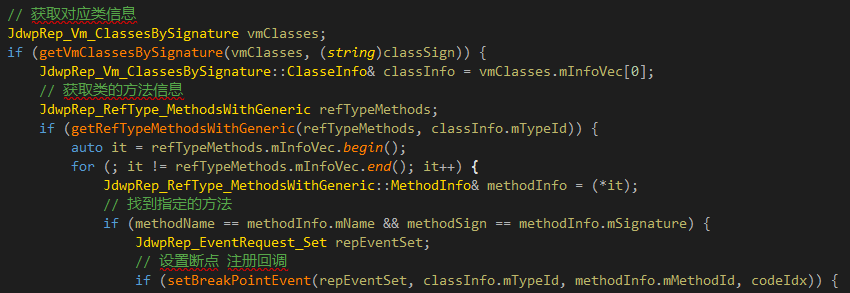
断点设置需要用到的命令是“EventRequest：Set”，该命令支持多种事件请求，包括断点、单步、类加载、方法进出、字段访问、线程、异常等多种事件，设置成功以后目标虚拟机会返回RequestID，并且在事件触发时会发送相应的事件信息给调试器请求处理(Event：Composite)，各类事件命令的具体格式可以参考[官方文档](http://docs.oracle.com/javase/8/docs/platform/jpda/jdwp/jdwp-protocol.html#JDWP_EventRequest_Set)。下面以断点为例进行讲解事件设置，首先看下事件命令请求的结构字段：



[8] 断点命令请求的数据包字段结构

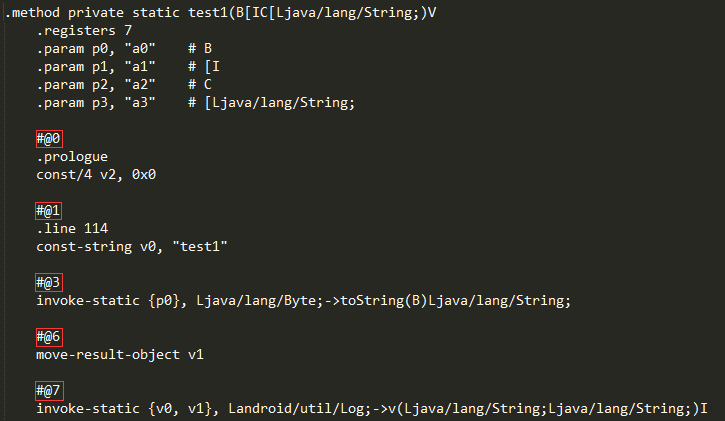
上面的断点事件我们只设置了一个过滤器，就是断点位置Location，这个结构体用于指明触发事件的代码位置，包含类型标记，ClassID，MethodID以及代码偏移DexPc。

1. 其中的ClassID和MethodID两个字段的值可以通过“VirtualMachine： ClassesBySignature”命令与“ReferenceType： MethodsWithGeneric”命令获取。



[9] 获取ClassID与MethodId的演示代码

1. 代码偏移位置DexPc值可以使用反汇编工具baksmali的“-l”参数获取。下图红圈中的数字就是代码偏移位置。



[10] backsmali反汇编结果中的代码偏移标记

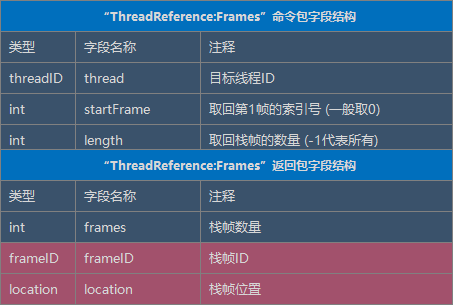
当虚拟机运行到我们设置的断点位置以后，会发送“Event： Composite”命令给调试器，目标虚拟机发送的命令中会包含线事件类型、请求序号、线程序号以及代码位置等信息，其中请求序号和断点命令返回的请求序号是对应的，调试器可以根据请求序号进行相应后续处理，例如获取线程堆栈、变量值等，最后需要恢复虚拟机的运行状态。断点事件报告的字段结构如下图：



[11] 断点事件报告的数据包字段结构

1. **如果获取当前函数调用栈信息?**

通过获取当前线程的调用栈信息可以定位目标函数的调用路径和源头，对于逻辑层次复杂的逆向分析非常有用。对应的JDWP命令为“[ThreadReference：Frames](http://docs.oracle.com/javase/8/docs/platform/jpda/jdwp/jdwp-protocol.html#JDWP_ThreadReference_Frames)”，该命令的请求包与返回包字段比较简单，结构如下：



[12] “ThreadReference：Frames”命令相关包的字段结构

返回数据为栈帧数组，每一层栈帧信息包含栈帧ID和栈帧位置2个字段，第一层栈帧一般为当前函数，其栈帧ID在后面堆栈获取变量值命令中需要使用。栈帧信息解析后输出的例子如下：

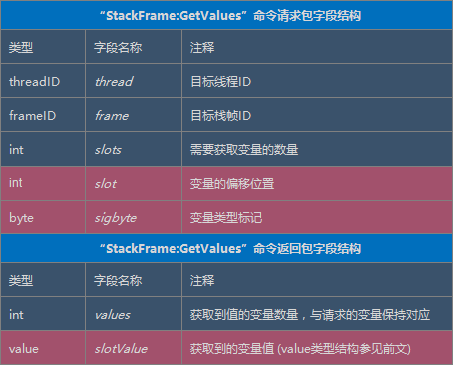


[13] “ThreadReference：Frames”返回数据的解析结果

1. **如何获取函数参数值与变量值?**

dalvik虚拟机和普通java虚拟机最大的区别之一应该是dalvik是基于寄存器架构的，可以观察dex反汇编后的smali代码内容，参数传递、变量赋值全部都是对寄存器的操作。关于smali语法与寄存器变量等方面的基础知识，推荐几篇文章作为前置阅读：[《smali-Registers》](https://github.com/JesusFreke/smali/wiki/Registers)、[《smali-TypesMethodsAndFields》](https://github.com/JesusFreke/smali/wiki/TypesMethodsAndFields)、[《Smali学习笔记》](http://wenku.baidu.com/link?url=sHJqwulm3Uc6kIO6e0KIpAX6p_NyedmIe4khwGgAWXsBOFPpOH64th0vpYZvyLjQ4fgEhzLDY7lB3Q1rDxNQr4TzfY21600v7Mg3uIxHiOy)，本文就不再赘述。

从当前堆栈中获取寄存器值需要使用“[StackFrame：GetValues](http://docs.oracle.com/javase/8/docs/platform/jpda/jdwp/jdwp-protocol.html#JDWP_StackFrame_GetValues)”命令，下面我们解析下该命令的请求包、返回包的字段结构：



[14] “StackFrame： GetValues”命令的注解

该命令请求中的有几个关键字段需要详细解释：

1. ThreadId，返回包中触发断点事件被挂起的线程ID。
2. FrameId，可以通过“[ThreadReference： Frames](http://docs.oracle.com/javase/8/docs/platform/jpda/jdwp/jdwp-protocol.html#JDWP_ThreadReference_Frames)”命令获取，从返回的栈列表中取第一层栈帧，即可获得的我们需要的当前栈帧ID。
3. Slot，变量偏移位置，这个字段是“[StackFrame： GetValues](http://docs.oracle.com/javase/8/docs/platform/jpda/jdwp/jdwp-protocol.html#JDWP_StackFrame_GetValues)”命中最为关键的字段，对于Debug版的apk可以通过“[Method： VariableTable](http://docs.oracle.com/javase/8/docs/platform/jpda/jdwp/jdwp-protocol.html#JDWP_Method_VariableTable)”命令获取，但是逆向分析遇到的几乎都是Release版本，是不包含这些调试辅助信息的，关于参数和变量对应偏移位置slot的计算方法，我们可以尝试从dalvik虚拟机源码中寻找答案。

dalvik虚拟机处理“[StackFrame： GetValues](http://docs.oracle.com/javase/8/docs/platform/jpda/jdwp/jdwp-protocol.html#JDWP_StackFrame_GetValues)”命令的函数为[handleSF\_GetValues](https://android.googlesource.com/platform/dalvik/+/eclair-release/vm/jdwp/JdwpHandler.c#1714)， handleSF\_GetValues函数主要负责解析请求包的字段信息，扩展返回数据的内存空间，最后调用[dvmDbgGetLocalValue](https://android.googlesource.com/platform/dalvik/+/eclair-release/vm/Debugger.c#2300)获取变量值。下面我们跟踪slot参数的传递使用过程来分析它的含义。



[15] [dvmDbgGetLocalValue](https://android.googlesource.com/platform/dalvik/+/eclair-release/vm/Debugger.c#2300)函数的代码解析

从上图的[dvmDbgGetLocalValue](https://android.googlesource.com/platform/dalvik/+/eclair-release/vm/Debugger.c#2300)函数代码解析过程中，我们有以下几点发现：

1. frameId值就是当前栈指针的内存地址，函数调用过程中使用的寄存器(参数与局部变量)相当于此处的内存映射，而slot值就相当于映射偏移索引号。参数使用最后的N个寄存器(内存段高地址)，局部变量使用从v0开始的前(M-N)个寄存器(注意 参数占用2个寄存器的情况)。
2. Slot偏移值会经过untweakSlot函数处理，这可能是针对Eclipse的变通方案。1000偏移被重定向到0，而0偏移被重定向到参数偏移起始处，在计算slot索引值时需要注意到这一点。
3. 获取变量值支持的几种类型细节如下：

boolean、byte、short、char、int、float类型的变量直接根据slot偏移取值，大小为4字节,占用1个寄存器；

array、object类型根据slot偏移取值为Object指针,查表获得ObjectId返回，大小为4字节，占用1个寄存器；

double、long类型根据slot偏移取值，大小为8字节，占用2个寄存器。

1. IDA的watch窗口指定变量类型取值后可能导致崩溃的原因在这里也可以找到，在进入其他函数调用过程时，没有及时手动修正watch窗口指定的寄存器类型，取值过程由于读取异常抛出导致虚拟机退出。

以下为2个函数的参数与局部变量slot偏移结果的对比，其中一个是普通成员函数，另外一个为静态成员函数，可以通过这两个例子加深对slot计算的理解：



[16] 普通成员函数参数与变量slot解析



[17] 静态成员函数参数与变量slot解析

1. **全新的android动态调试工具——AVMDBG**

AVMDBG是android虚拟机调试技术研究过程中一个实验项目，目前只支持dalvik虚拟机，已实现代码断点、堆栈输出、参数变量值获取等基础功能。

项目地址：<https://github.com/cheetahsec/avmdbg>

项目说明：AVMDBG的目标是打造一款轻量级的的android虚拟机调试器，底层使用C++编写，通过Python扩展的方式提供调试接口，可以通过编写python脚本实现对安卓app的快速动态调试，当前版本主要提供以下API：

1. bool attach(*string*& processName);

功能: 附加到目标进程

1. void waitLoop();

功能: 循环等待调试事件

1. bool setBreakPoint(py::dict& breakPoint);

功能: 设置断点事件

1. py::list getStackFrames(ObjectId threadId);

功能: 获取当前线程堆栈，可在断点回调函数中使用

1. py::dict getRegisterValue(py::dict& Context, *string*& regName, u1 varType);

功能: 获取寄存器变量的值，需要指定寄存器名称(V命名或P命名法)以及变量类型

1. py::list getObjectFieldValues(ObjectId objectId);

功能: 获取对象的成员变量信息，参数为对象ID

1. py::dict getArrayObjectValue(ObjectId objectId);

功能: 获取数组类型变量的信息

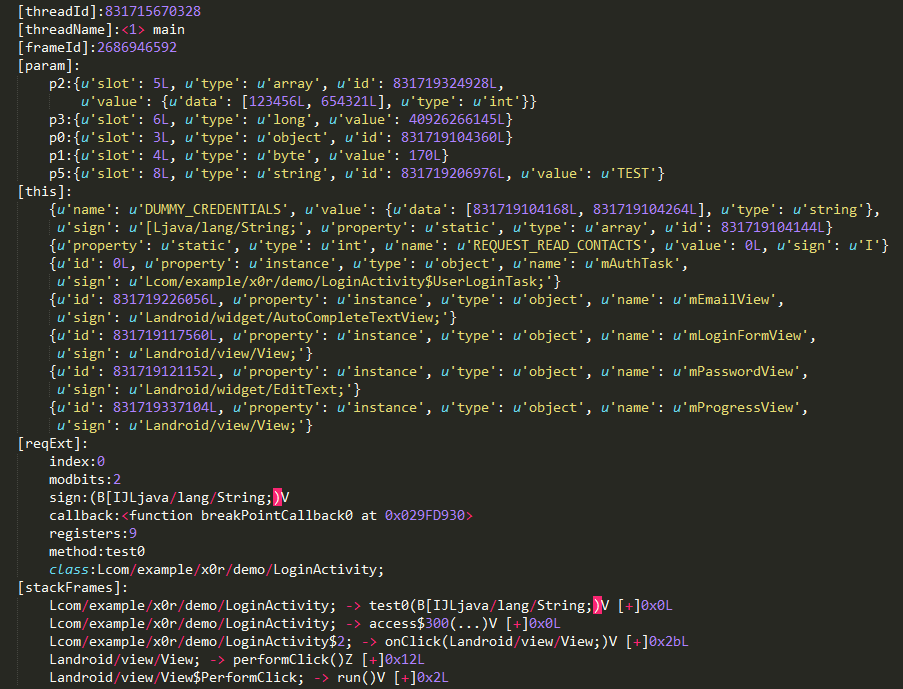
1. py::str getStringValue(ObjectId objectId);

功能:获取字符串String类型变量的值

详细的使用说明可以参考项目文档和测试demo，以下为部分测试代码展示：



[18] AVMDBG功能测试代码

[19] AVMDBG测试输出结果

1. **参考文档**
2. 《[深入Java调试体系](http://www.ibm.com/developerworks/cn/views/java/libraryview.jsp?search_by=%E6%B7%B1%E5%85%A5+Java+%E8%B0%83%E8%AF%95%E4%BD%93%E7%B3%BB)》
3. 《[Java(tm) Debug Wire Protocol](http://docs.oracle.com/javase/6/docs/platform/jpda/jdwp/jdwp-protocol.html)》
4. 《[深入理解Android之Java虚拟机Dalvik](http://blog.csdn.net/innost/article/details/50377905)》
5. 《[Dalvik虚拟机进程和线程的创建过程分析](http://blog.csdn.net/luoshengyang/article/details/8923484)》
6. 《[smali-Registers](https://github.com/JesusFreke/smali/wiki/Registers)》、《[smali-TypesMethodsAndFields](https://github.com/JesusFreke/smali/wiki/TypesMethodsAndFields)》
7. 《[Android dalvik虚拟机源代码](https://android.googlesource.com/platform/dalvik/+/eclair-release/vm/jdwp/)》