## 安装Frida

pip install frida # 安装Frida

Pip install frida-tools # 安装frida管理工具

在线安装比较慢

可以去<https://pypi.org/project/frida-tools/#files> 下载在安装

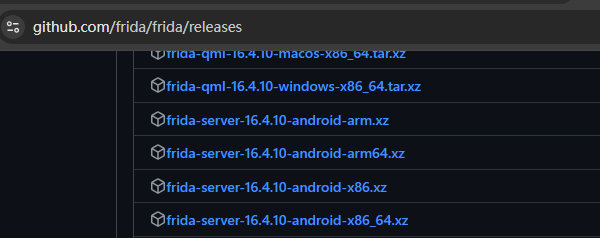
Pip install -U ojection 安装基于frida的objection工具

## 安装Frida-Server

确定Android cpu版本:



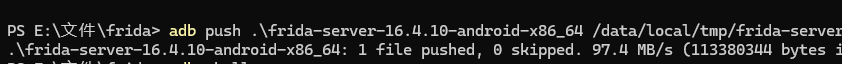
下载对应的Frida-Server



下载的:frida-server-16.4.10-android-x86\_64.xz

解压后推动到虚拟机

adb push .\frida-server-16.4.10-android-x86\_64 /data/local/tmp/frida-server



然后进入虚拟机运行Frida-server:

adb shell

Su # 获取超级权限

cd /data/local/tmp

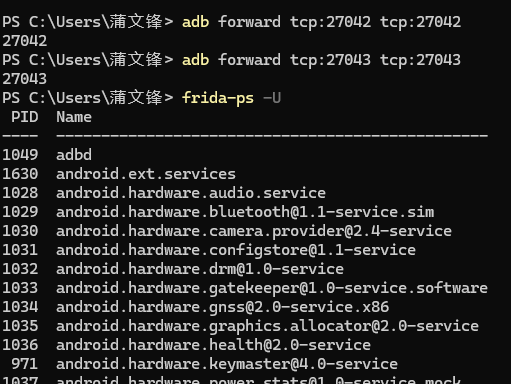
Chmod 777 frida-server

./frida-server # 运行



不要关闭此shell,另开shell

## 运行端口转发监听



adb forward tcp:27042 tcp:27042

adb forward tcp:27043 tcp:27043

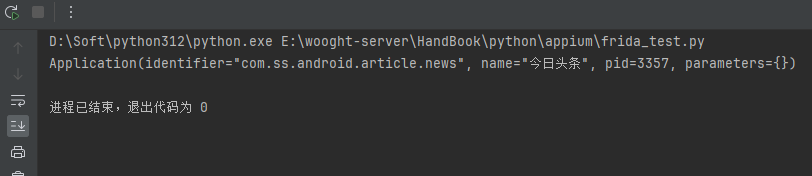
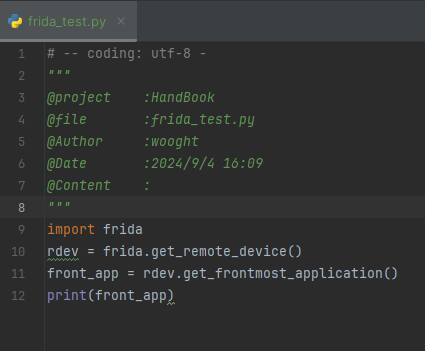
Frida-ps -U # 运行是否输出进程列表

27042端口用于frida-server通信的默认端口号，不转发的话客户端是连接不上服务端

使用默认的端口很容易被检查到,启动Frida-server的时候设置IP和端口

./frida-server -l 127.0.0.1:9999

## Python 运行测试



运行过程中,如果报:frida.TransportError: the connection is closed

则需要kill frida-serve 再重新运行Frida-server

## 淘宝Spdy协议修改

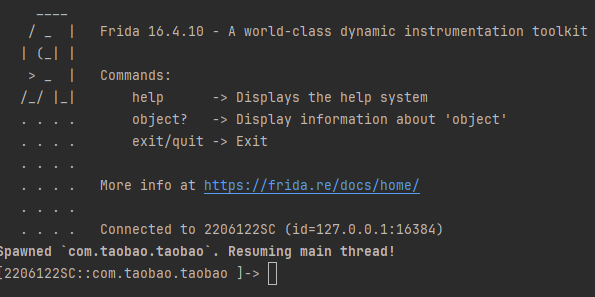
test.js:

Java.perform(function () {  
 var SwitchConfig = Java.use('mtopsdk.mtop.global.SwitchConfig');  
 SwitchConfig.isGlobalSpdySwitchOpen.overload().implementation = function () {  
 return false;  
 }  
});

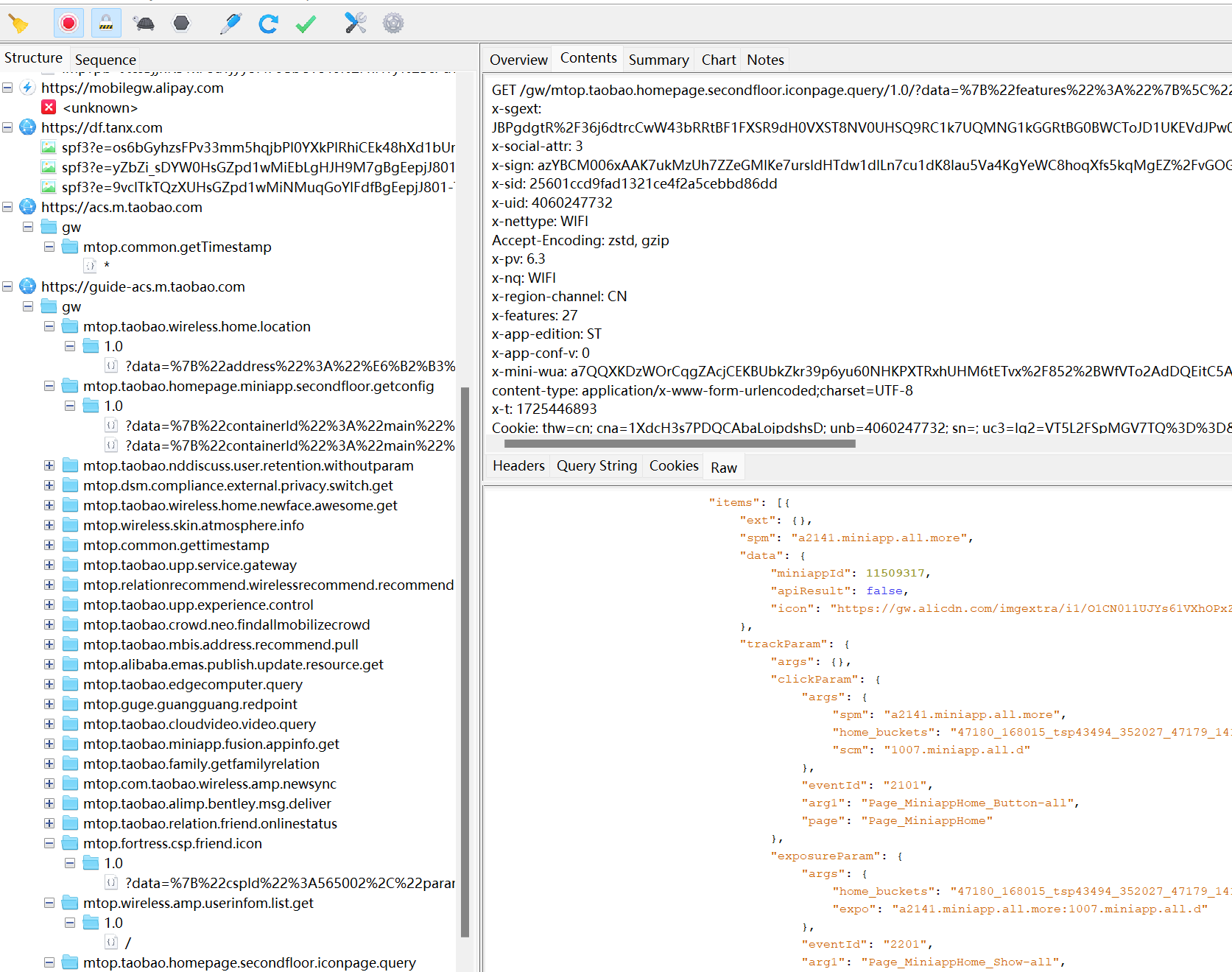
Frida 执行hook(钩子)操作



运行如下则钩子植入成功:



Charles 抓包成功:



## 安装objction

Pip install -U objection

安装完成后,查看基本信息 objection



由图而知:

Objection默认通过USB连接设备，这里不必和Frida的命令行一样通过-U参数指定USB模式连接

通过-g参数指定注入的进程，

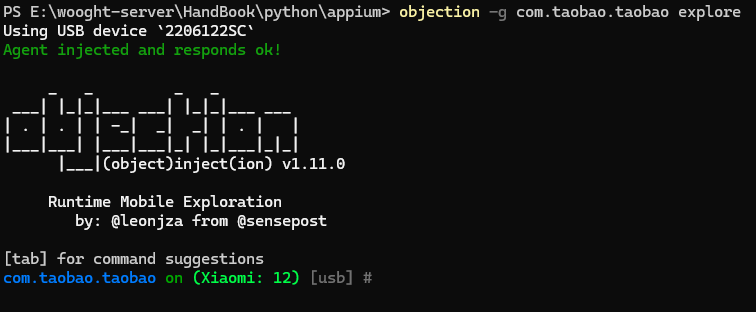
通过explore命令进入REPL模式（交互模式）。在进入REPL模式后便可以使用Objection进行Hook的常用命令。

通过-N参数来指定网络中的设备并通过-h参数和-p参数来指定对应设备的IP和端口以进行连接，从而完成对网络设备的注入与Hook。

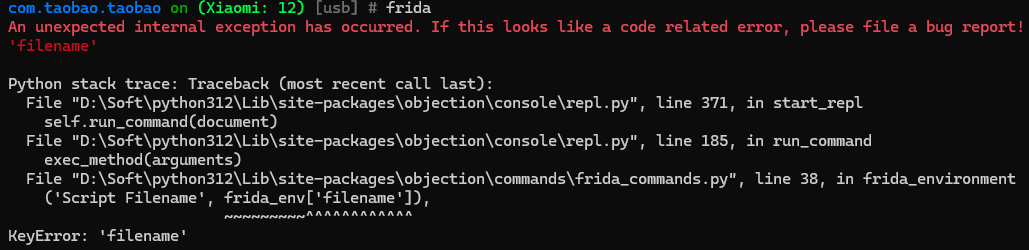
通过patchapk命令将frida-gadget.so打包进App。

通过objection进入淘宝app

Objection -g com.taobao.taobao explore



在objection中查看frida的信息



注释掉报错的行.

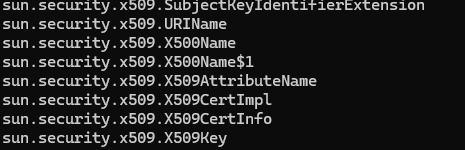
基本命令:

Android ui screenshot [image.png] #截屏,并保存到当前文件

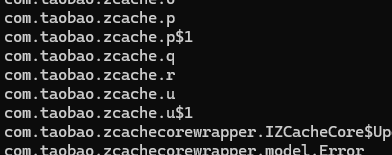
android intent launch\_service # 强制启动service

内存相关命令:

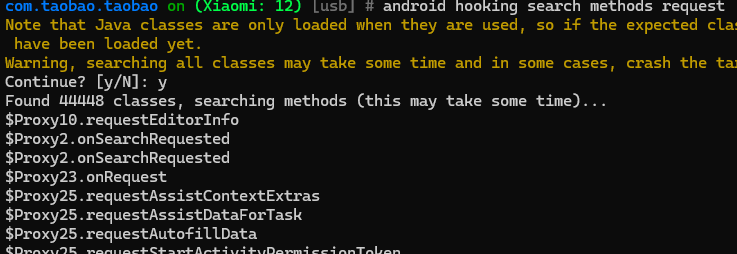
Android hooking list classes # 列出内存中所有实例化/运行的类



Android hooking search classes taobao # 在所有类中查询 taobao相关的类

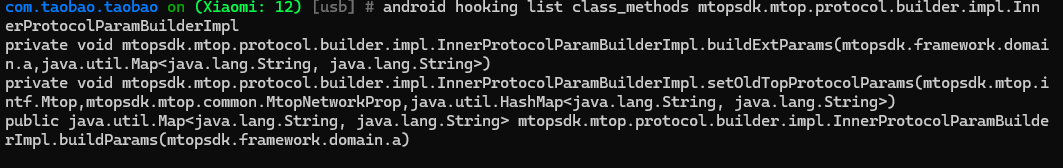


Android hooking search methods request # 查找由request关键词的方法



Android hooking list class\_methods mtopsdk.mtop.protocol.builder.impl.InnerProtocolParamBuilderImpl

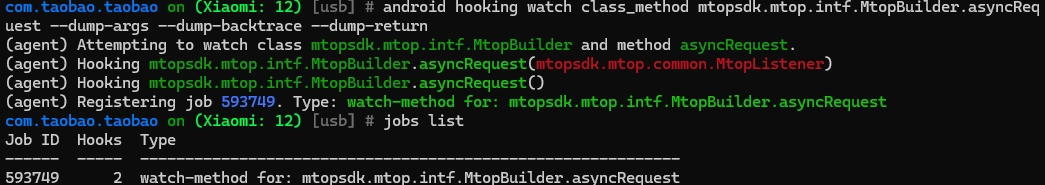
搜索类方法



android hooking watch class\_method mtopsdk.mtop.intf.MtopBuilder.asyncReq

uest --dump-args --dump-backtrace --dump-return # 对类/方法进行hooking 追踪

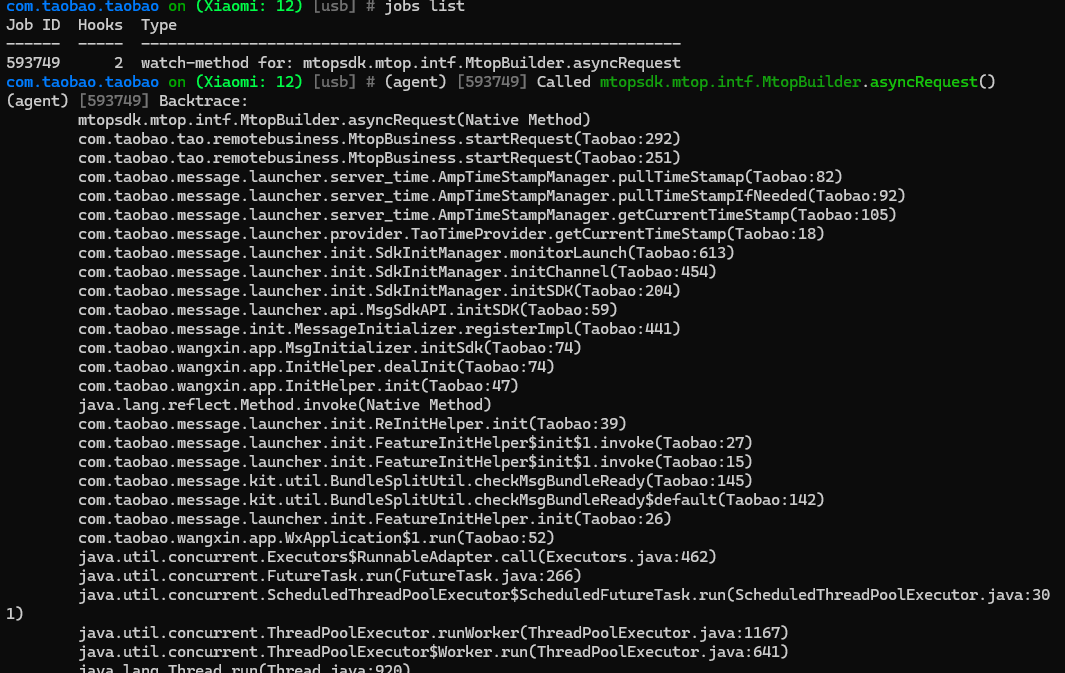
–dump-args、–dump-backtrace、–dump-return 分别用于打印函数的参数、调用栈以及返回值。



此时虽然只确定了Hook构造函数，但是默认会Hook对应方法的所有重载。

在输出的最后一行显示Registering job 593749，这表示这个Hook被作为一个“作业”添加到Objection的作业系统中了，此时运行job list命令可以查看到这个“作业”的相关信息，如下图所示。可以发现这里的JobID对应的是593749，同时Hooks对应的2正是Hook的函数的数量。

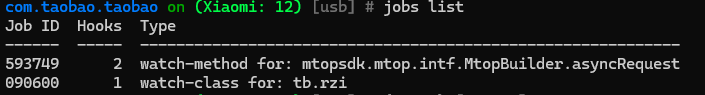
在操作app时,会触发frida进行hook追踪



Android hooking watch class tb.rzi # 查看类



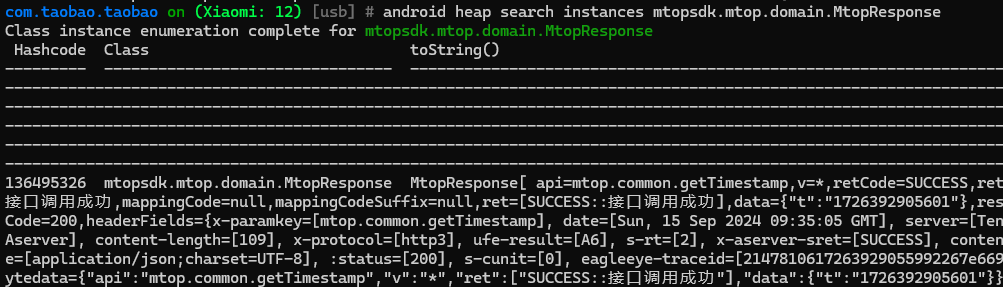
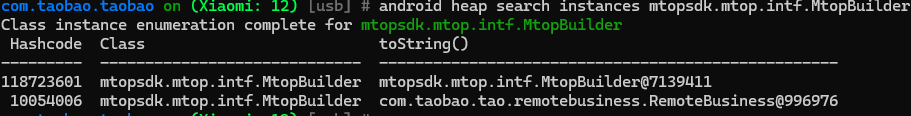
执行android hooking watch class/class\_method 命令,会默认加入job中,jobs list 查看job



Jobs kill ID # 杀死jobs



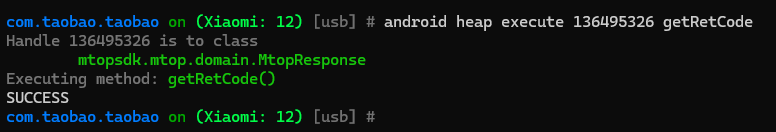
Heap 命令(主动调用)

Android heap search instances mtopsdk.mtop.intf.MtopBuilder # 搜索实例

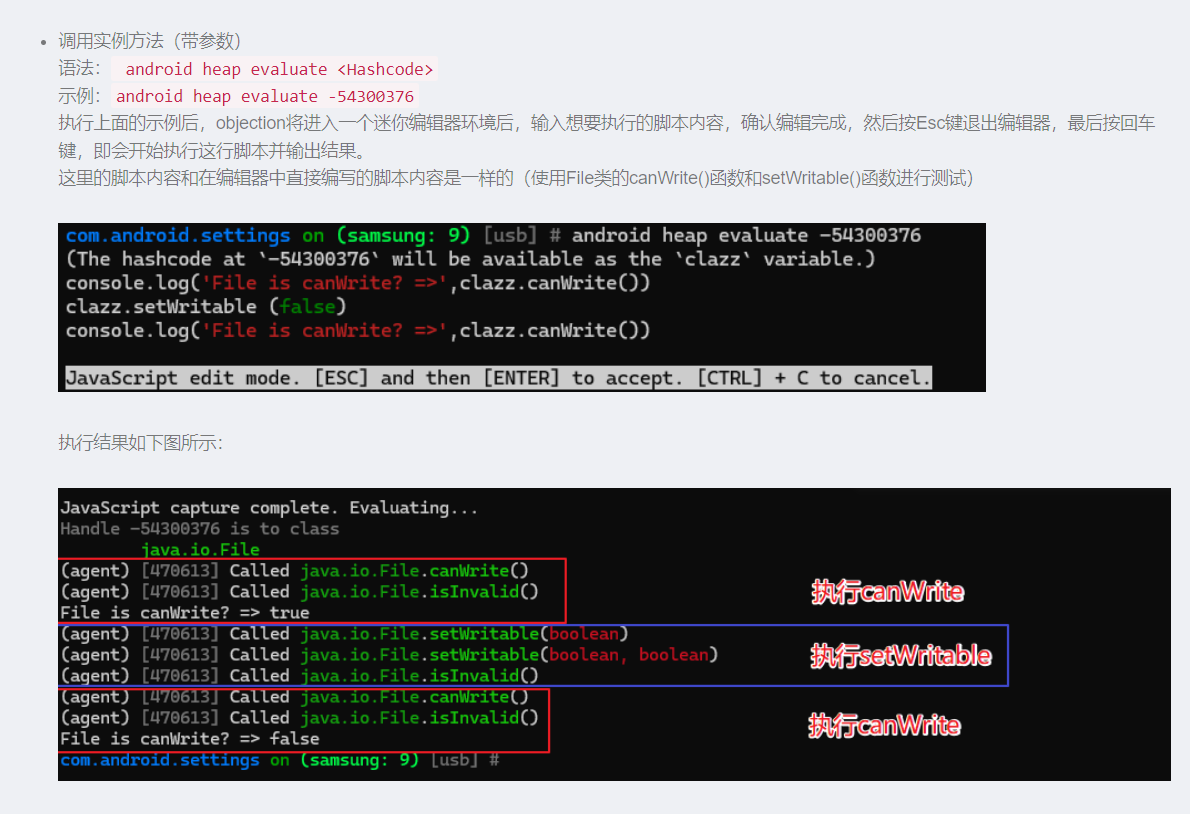
调用实例方法

Android heap execute 136495326 getRetCode

其中:136495326 指的是上一步获取到的Hashcode, 数字后面的是调用的方法



调用带参数方法:



Memory 命令:

Memory list modules # 内存中加载的库

