# **Frida入门与基本使用**

## [优先级: 1] 会议背景**（What we do? ）**

**Frida作为我们逆向开发者的必备逆向调试工具之一，简单易用的的特性，让我们可以更好的了解到程序在运行时的行为，并对其进行修改,以及查询参数信息。帮助安全研究员、逆向工程师和开发人员分析和调试应用程序，以及发现和修复漏洞。但对于开发的同事来说，可能比较模糊。所以这篇文章让大家都能很好的上手frida，更好的帮助我们推进问题、修复问题。**

## [优先级: 2] **解决什么问题（Why we do it? or What’s the problem）**

**Frida解决了在应用程序运行时进行调试和修改的问题，克服了传统静态分析和调试工具的限制，大大提高了安全研究和开发的效率。**

## [优先级: 3] **我们怎样使用它（How we use it?）**

## **1、Frida原理：**

**官网：**[**https://frida.re/**](https://frida.re/)

**Frida的工作原理是通过在目标应用程序中注入自己的JavaScript运行时环境来实现。Frida运行时环境允许用户在目标应用程序的上下文中执行JavaScript代码，从而实现动态修改应用程序的行为。**

**具体地说，Frida在运行时将自己的JavaScript运行时环境注入到目标进程中，然后通过与其交互来执行JavaScript代码。在注入过程中，Frida会使用一些技术来绕过应用程序的安全保护机制，以确保注入的成功和稳定性。**

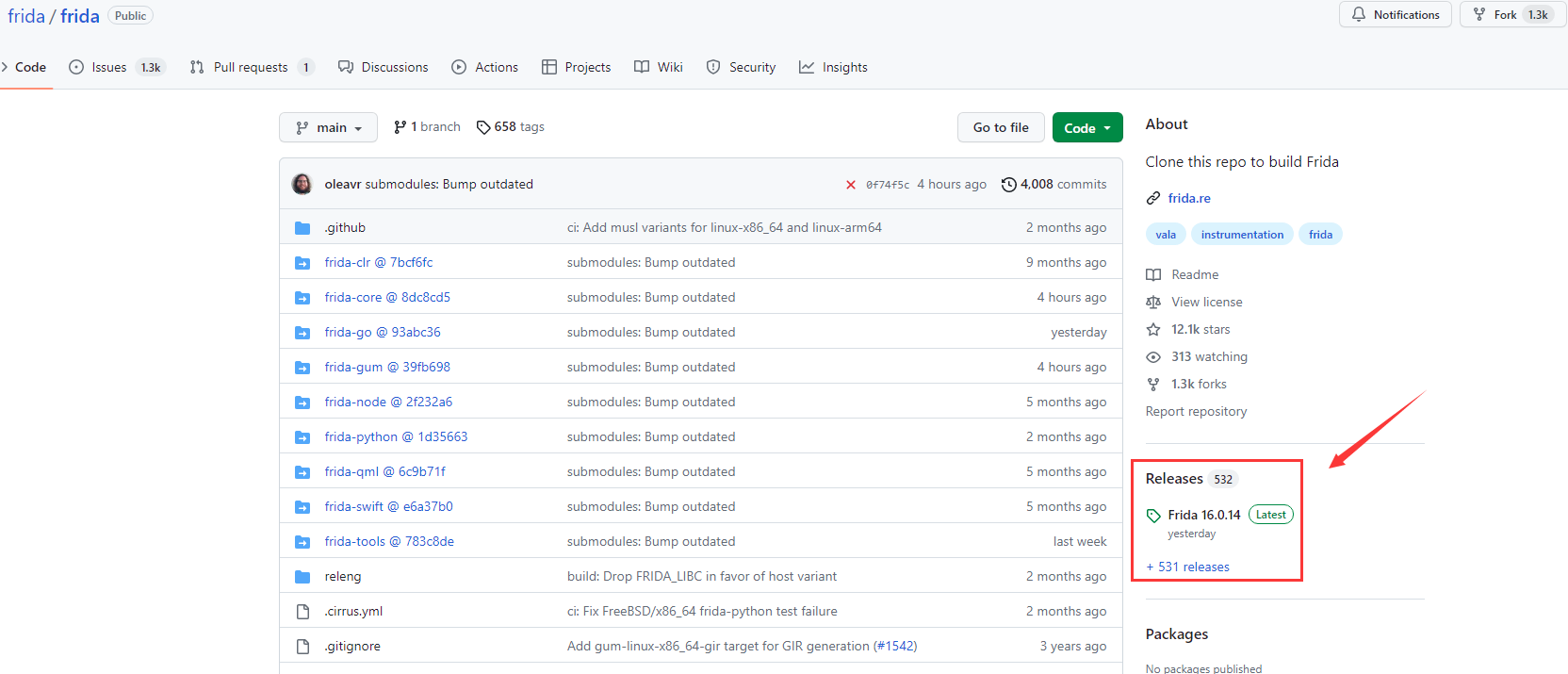
**一旦Frida注入成功，用户就可以使用Frida提供的API来执行一系列操作，比如：拦截函数调用、修改函数参数和返回值、修改内存中的数据等等。**

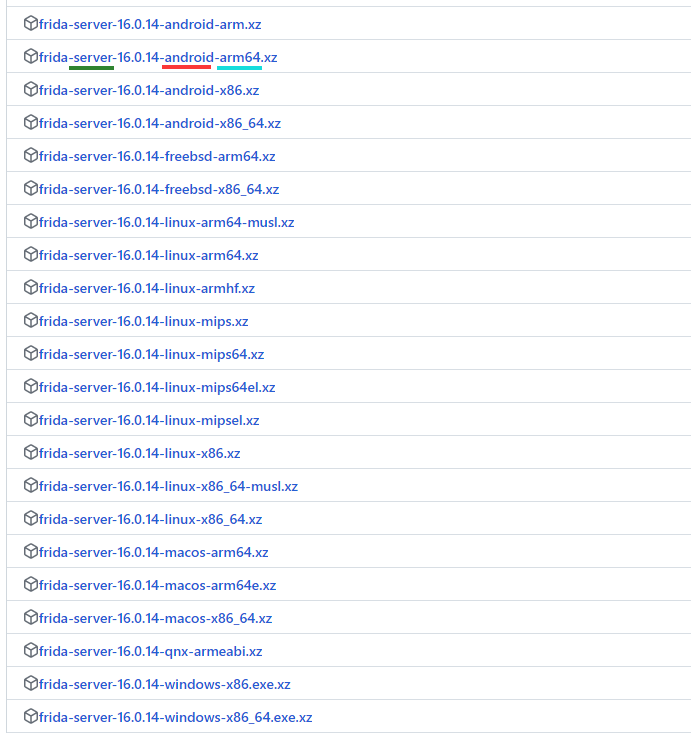
## **2、Window安装（环境搭建）**

### **一台Root的手机**

### **手机端安装frida-server:**

**Frida 提供了所有版本都可以在 github网站获取(持续更新中...)：**[**https://github.com/frida/frida**](https://github.com/frida/frida)



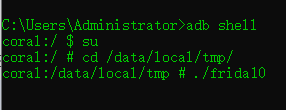
**我们手机的架构这里我们都是用的arm64平台的，所以选择的是android-arm64:**

**安装好之后，解压，将文件拷贝至手机中：**

**进入目录下给权限:**

**启动frida(服务端):**

**当然也可以加 & 使其后台运行:**



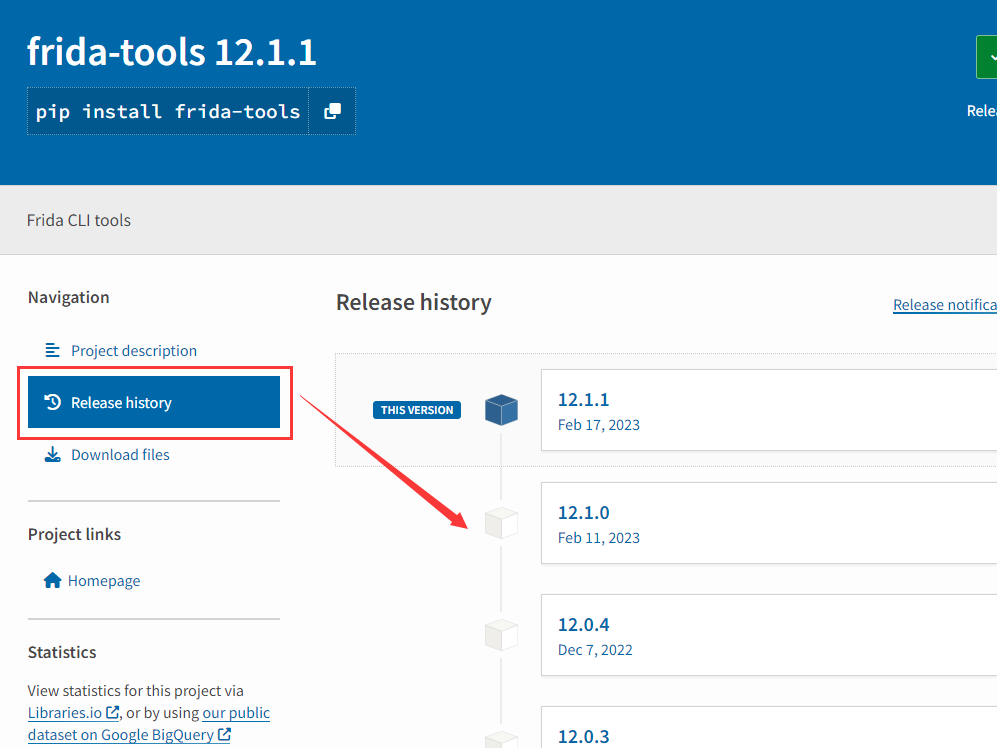
**这样手机端的frida-server就运行起来了。**

**电脑端需要安装frida 、frida-tools，2者的版本要相近即可**

**前置条件：需要安装 python3**

**注意事项:电脑端安装的frida 要和 服务端(手机上的frida)的版本一致,服务端是16.0.14**

**frida-tools的版本如何选择：**

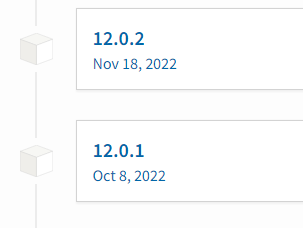
**Frida-tools 版本:**[**https://pypi.org/project/frida-tools/#history**](https://pypi.org/project/frida-tools/#history)

**我们的frida-server是最新的版本，那么我们这里的frida-tools也是选择最新的版本即可.**

**如果你需要安装的frida是旧版本的(比如16.0.2)，第一点手机上的frida-server和电脑的frida版本必须一致(都是16.0.2)，然后 你需要去frida-tools官网看看与16.0.2版本最近的是哪一个版本,这里比较的是时间:**



**可以知道16.0.2是2022.10.22发出的，我们找最近的frida-tools版本：**



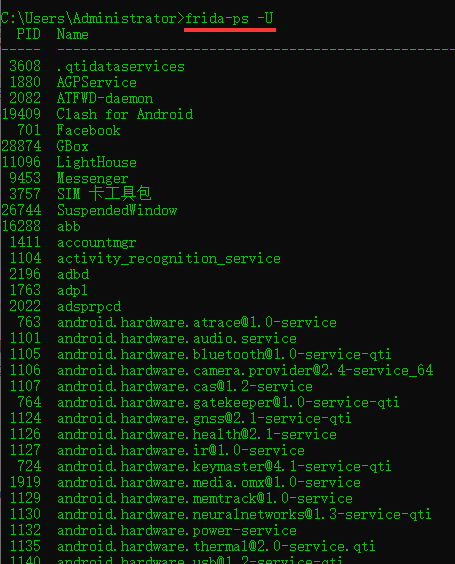
**可以看出他们的大版本(16、12)不一样，小版本还是几乎一样的(0.2)。安装11月18号的：12.0.2**

**查询电脑端frida版本：**



**手机连接到电脑上，然后开启手机上的frida服务端之后,检查手机上的frida是否运行成功:**

**成功则会打印手机中运行的进程：**



**现在，我们的手机上有了frida,电脑上也通过pip3安装了frida和frida-tools**

**手机上先启动frida,然后电脑端执行frida-ps -U 返回进程。至此基础环境搭建完成.**

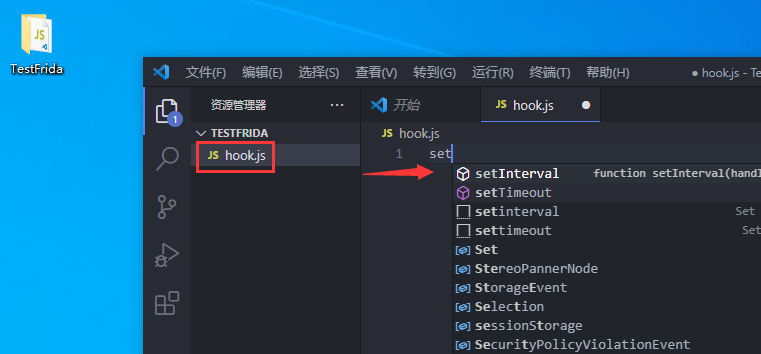
## **2-1实用环境搭建**

**1、安装VScode(默认已完成)**

**2、安装Nodejs(默认已完成)**

**3、执行命令下载frida自动补全功能**

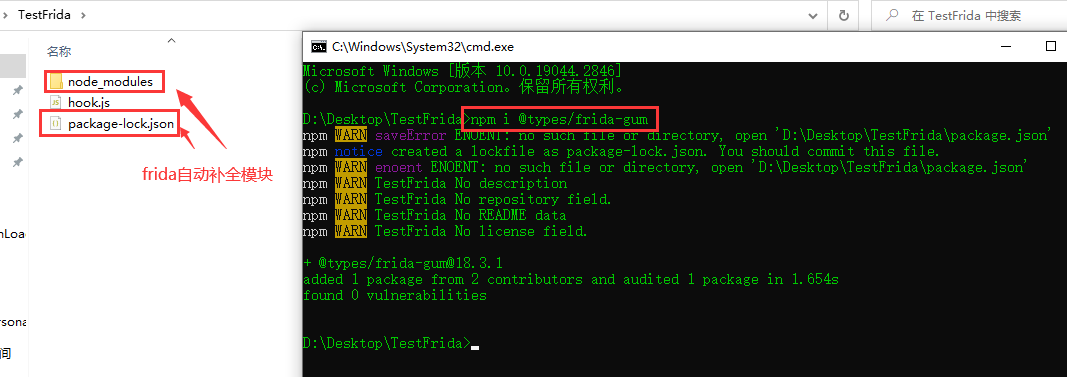
**默认使用vscode编辑js代码是没有fridaAPI补全的**



**fridaAPI补全方式:**

**在需要进行js脚本编写的目录下 执行一下命令:**

**执行命令后，该目录下就会多出1个文件目录，和一个json文件。**



**有了这2个文件后，写代码就可以会自动补全了：**

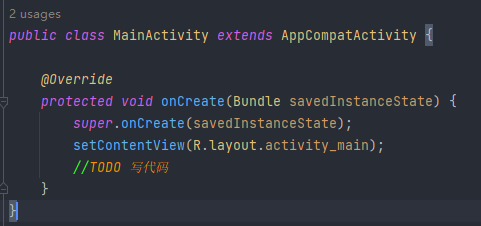


## **3、Frida基本操作**

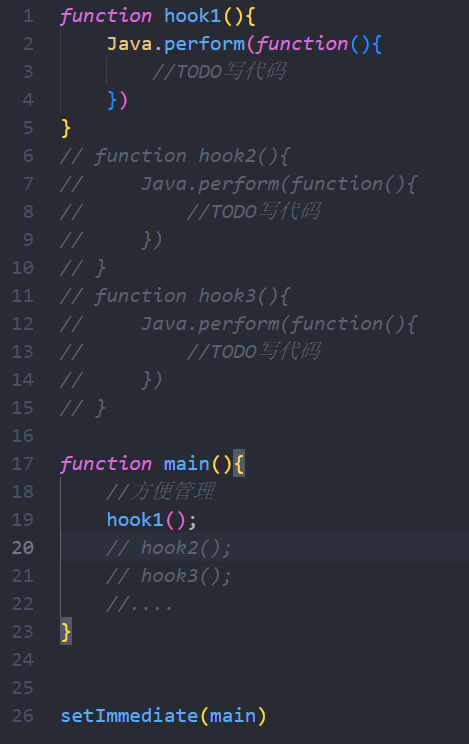
### **1、基础代码搭建：**

**frida的基础代码搭建，就跟我们AS创建一个activity的基础搭建类似：**

**Activity的代码基础搭建：**



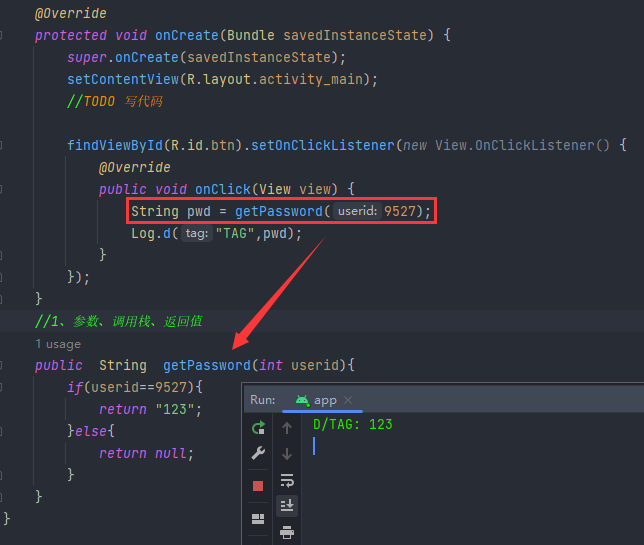
**Frida的代码基础搭建:**



**当然，随着你frida用得熟练，你可以写出很多自定义得代码框架**

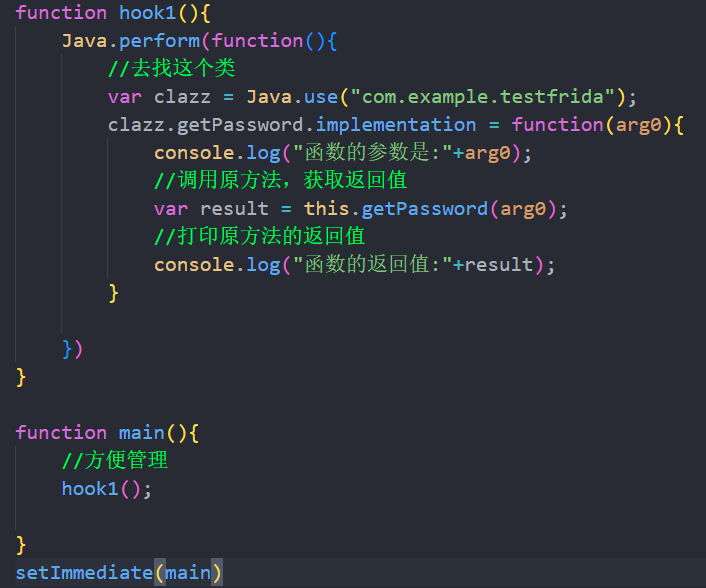
### **2、编写简单的hook代码**

**Demo：**



**点击按钮，调用getPassword函数，返回123**

**上hook代码：**

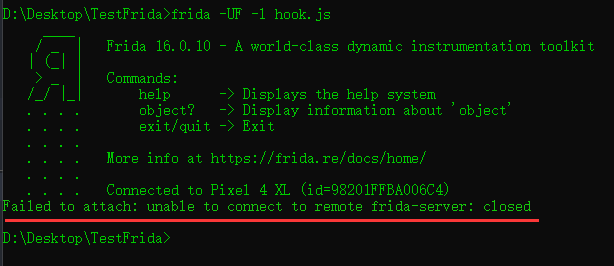


### **执行脚本：**

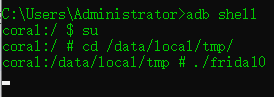
**在脚本目录下执行命令:**

**-U：通过USB启动 -F:挂载前面的应用 -l 挂载的脚本**

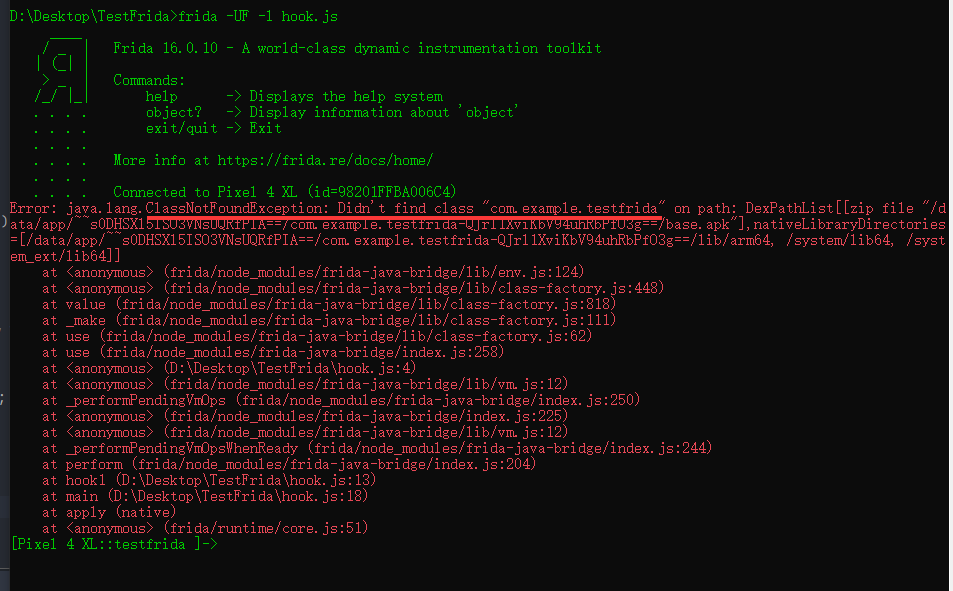
**解释：通过USB的方式 将脚本hook.js挂载到手机当前前台应用上。**



**报错了，attach是失败，不能连接到远端的frida-server：原因是远端服务是关闭的，这里也提醒一下，每次手机启动了frida服务，就不要拔掉USB，否者frida服务就会关闭，需要从新启动：**



**再次执行脚本： 好的，再次报错，说找不到这个类**



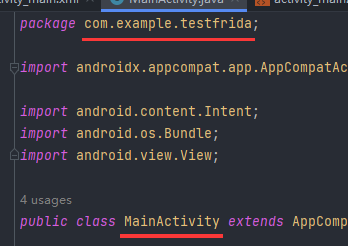
**出现这个问题的情况：**

**1、这是因为我们hook代码的包名不是完整**

**2、类名写错了**

**3、你当前hook的应用在联网wifi状态下自动更新了，导致你hook的类名(混淆过的)改变了。**

**4、类加载器不对。**



**这里修改类名:**

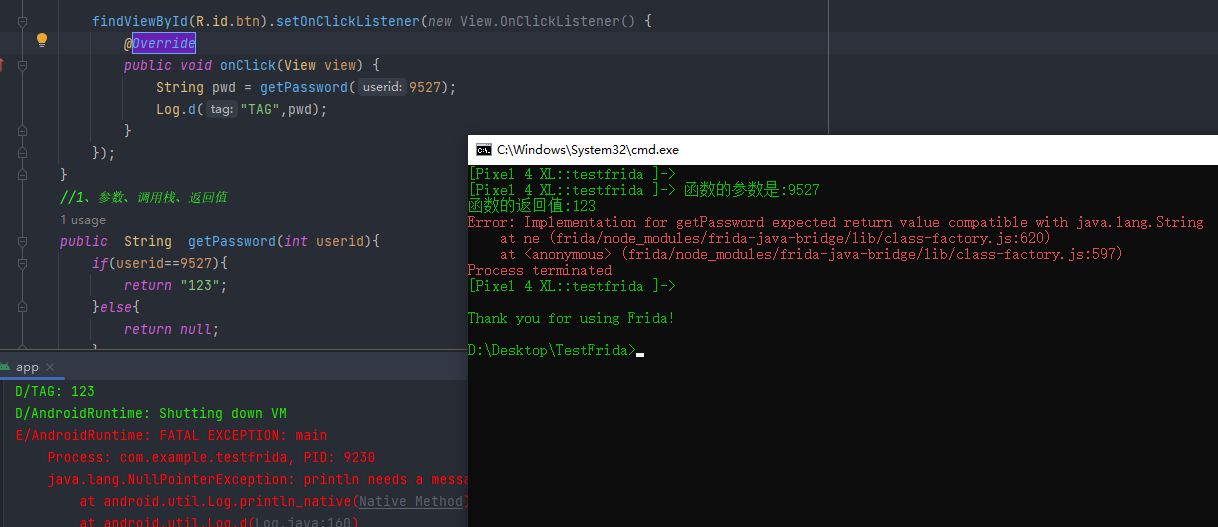


**所以我们要hook的是具体的类，要写全。**

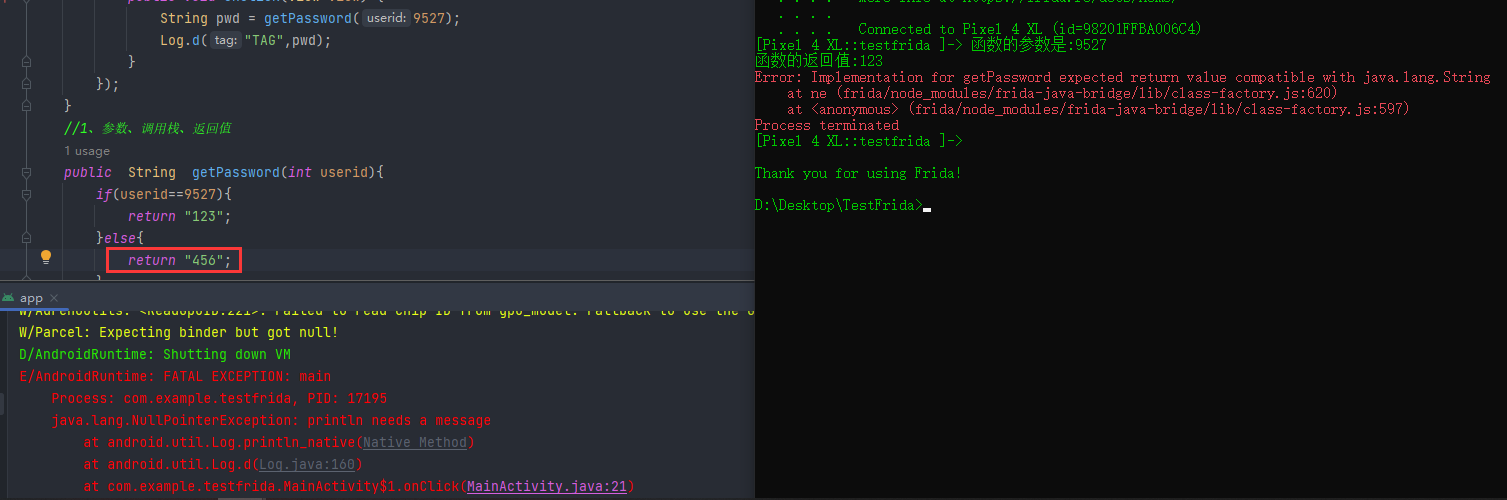
**再次执行脚本：**



**接下来我们在点击按钮：**



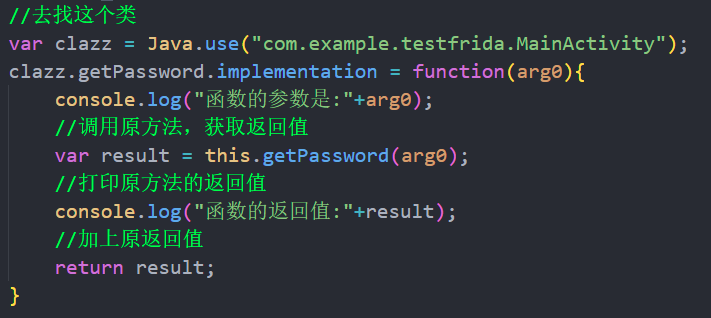
**并不是else返回的 null导致:**



**所以说是因为我们hook的代码出了问题.根据frida的报错，原因是说 实现的getPassword期望返回的值是一个兼容java.lang.String的，而我们的hook代码返回值是什么： 没有返回值**



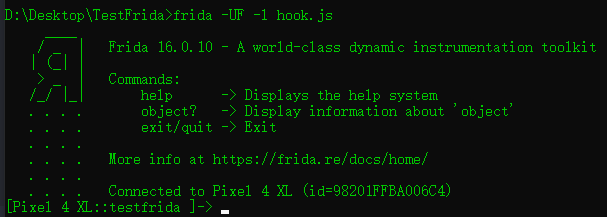
**好的。加上原返回值：**



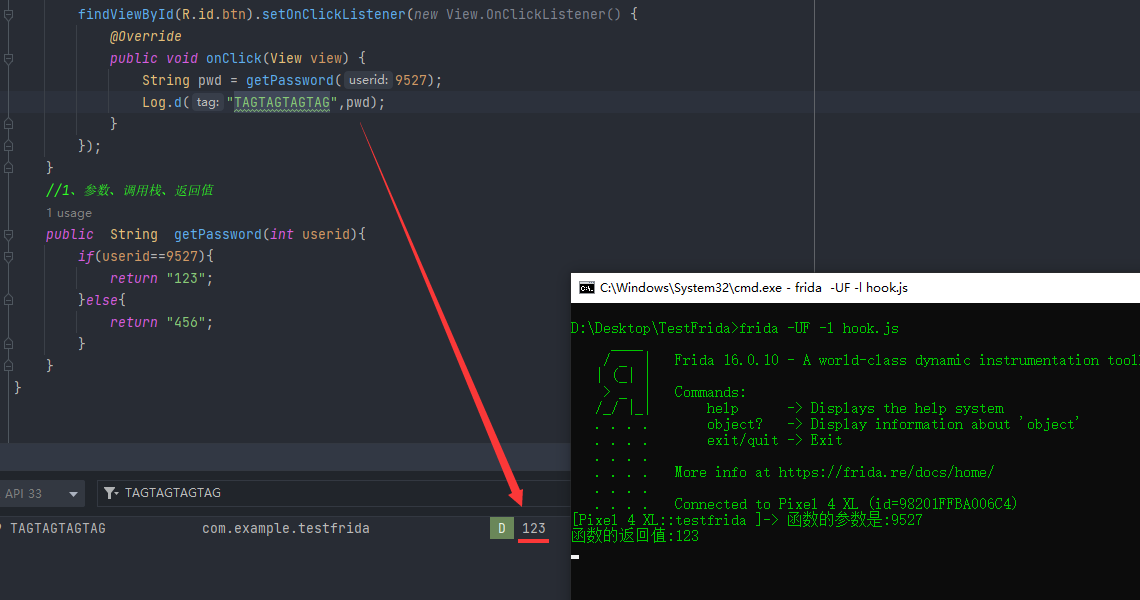
**再次启动脚本： 提示我们在手机上没有前台应用。**



**好的，所以告诉我们要先启动前台应用，在执行脚本：正常了**



**点击按钮：调用成功! hook成功 ! 返回值打印成功!**

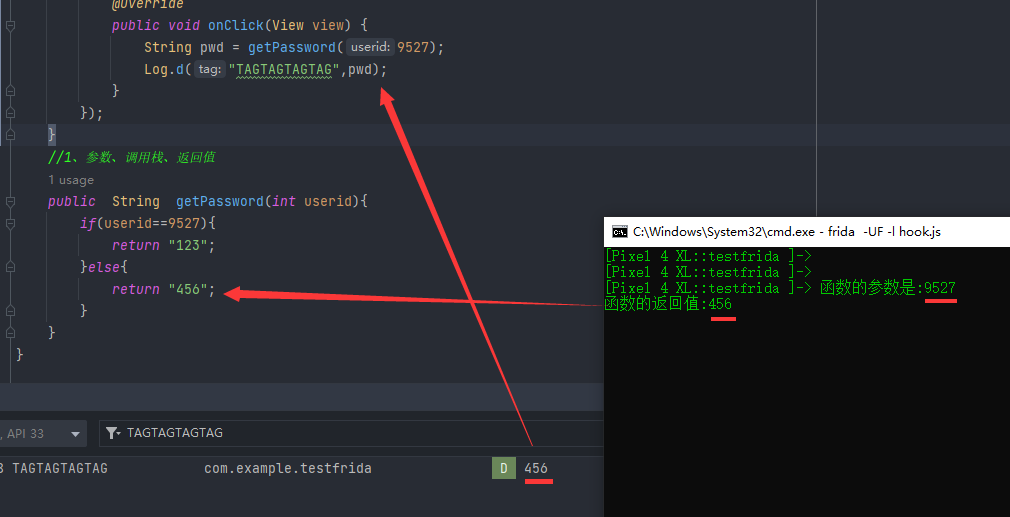


**接下来，我们进行才是真正的修改了:**

**修改参数**



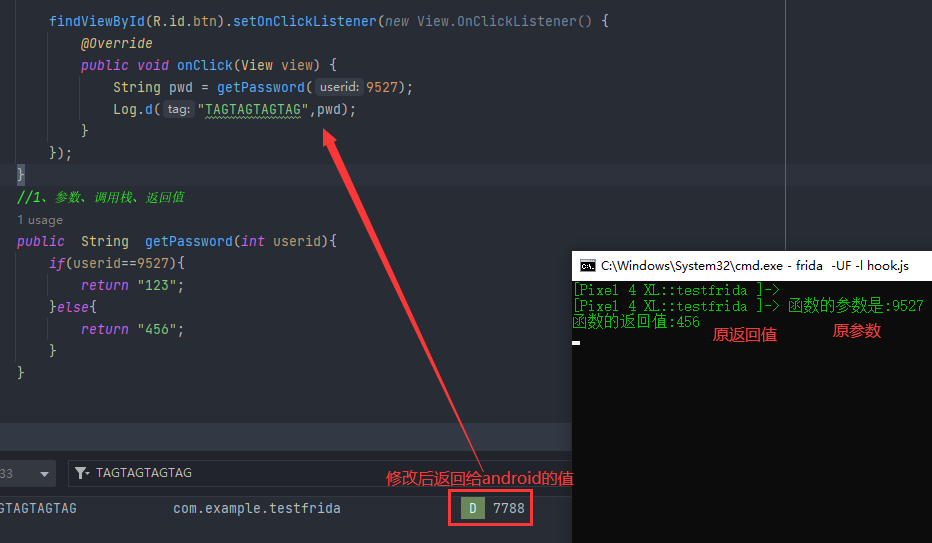
**点击按钮：可以看到，android代码走了的参数被改为10086后，走else语句返回456了，我们传入的还是9527,被我们拦截并修改了**



**修改返回值：**



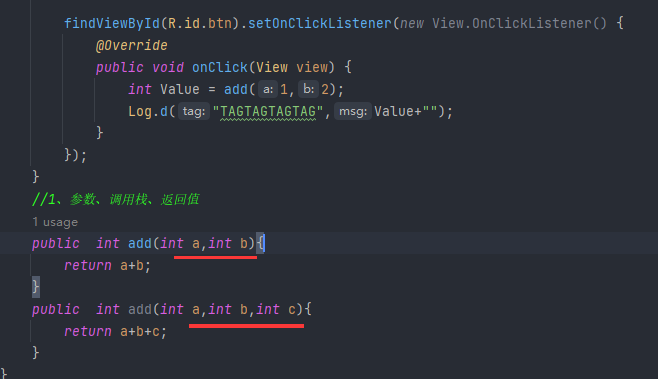
**点击按钮**



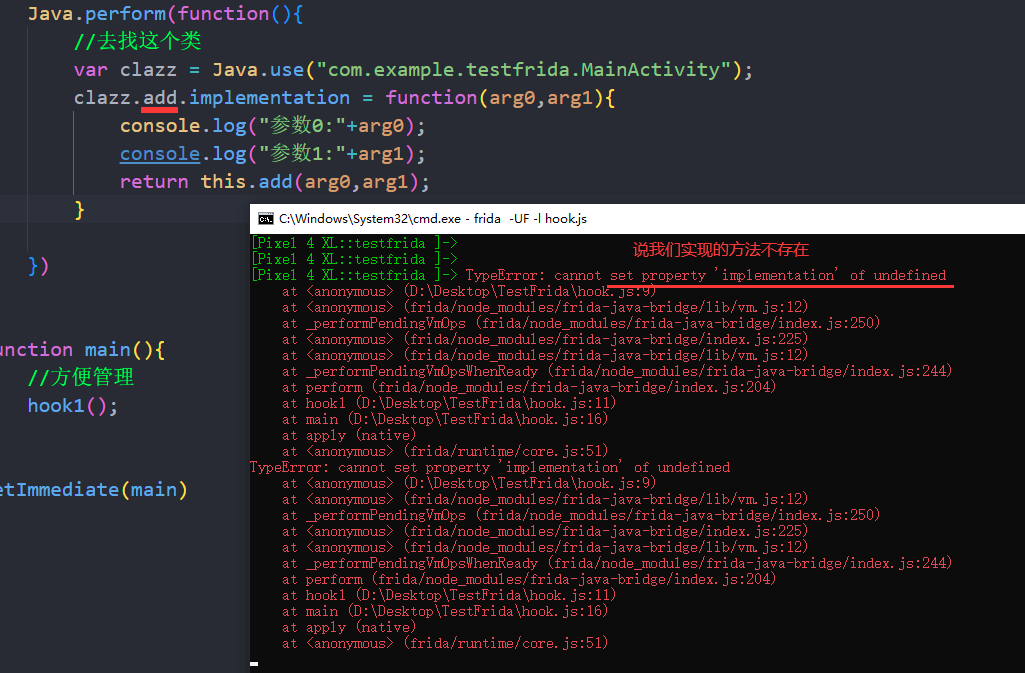
注意！修改了参数直接ctrl+S保存js脚本，我们执行成功的frida脚本就会自动重新执行一遍!!!，所以不需要在exit退出frida环境，在重新执行脚本。

**如果有重载，怎么hook？关键字: overload**

**Code：**



**Hook代码：我这里修改了代码，直接保存脚本，frida马上就报错了：说实现的方法不存在:**

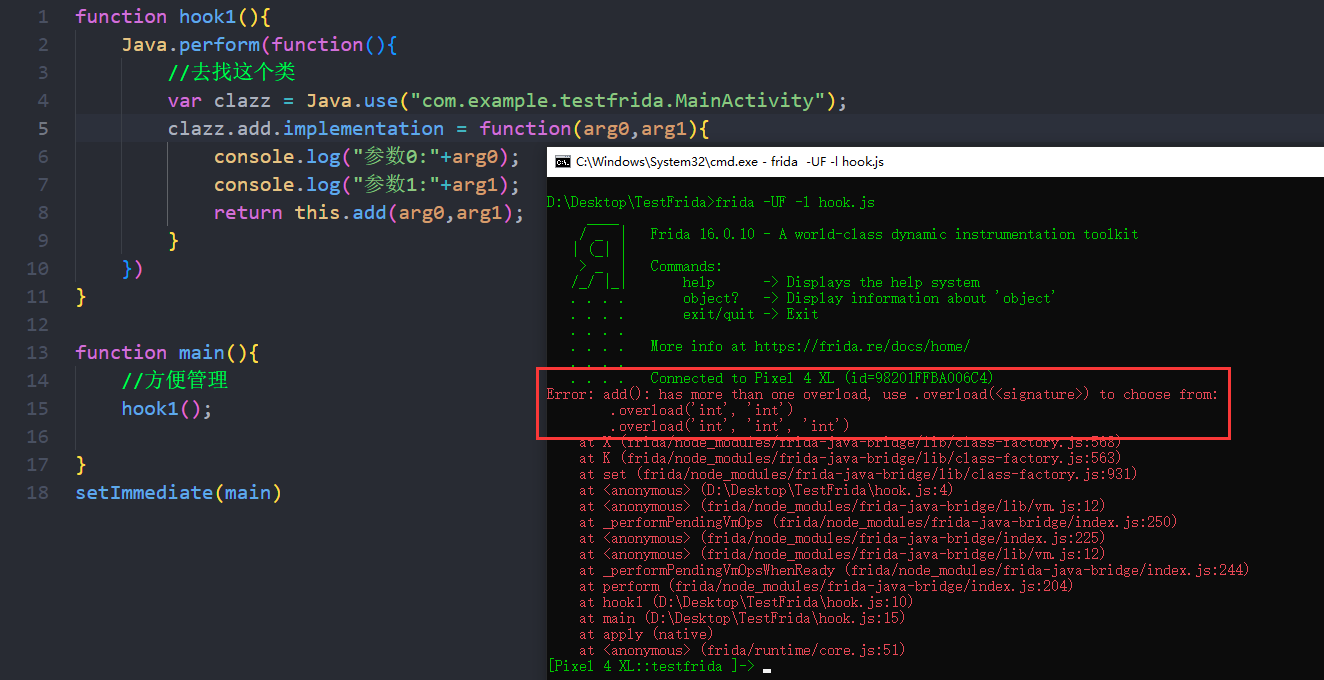


**这里的原因是因为我这里hook的代码改了，但是应用的代码，还没有改，还是之前那个getPassoword的那个代码，现在已经是add了。所以提示我们我们要实现的方法(add)，找不到。 这种问题还是比较常见出现的情况：**

**1、我们的hook代码的函数名字拼错了**

**2、海外app自己更新了，导致了代码的混淆随机更新了(举例:以前函数名是a0,更新后是X9),甚至如果app混淆加上了这个函数的类名，那么这个类名就会在更新中，被修改。就会报前面讲到的找不到类的错误了。**

**好，我们也更新下手机的应用包，再次启动脚本:**

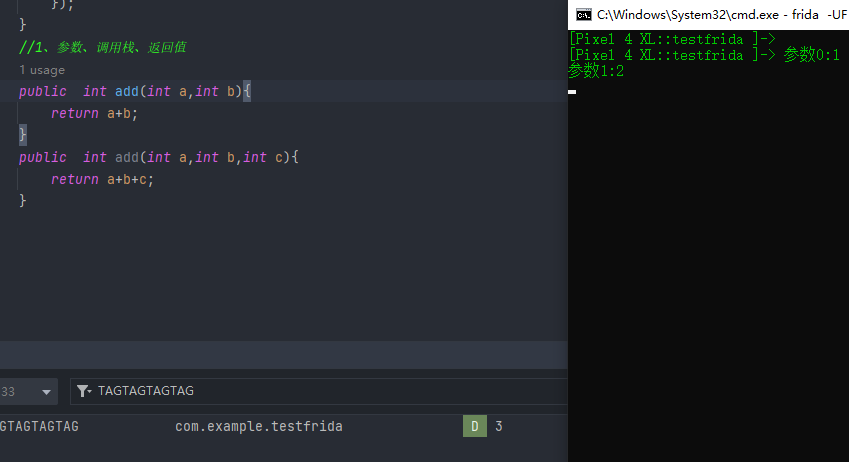


**再次报错了：说add()方法有跟多的重载，使用overload(方法签名)去选择:**

**那我们这里其实想要hook的是2个参数的，现在还不需要hook第三个参数，所以我们这里加上 :**



**点击按钮：正常hook打印**

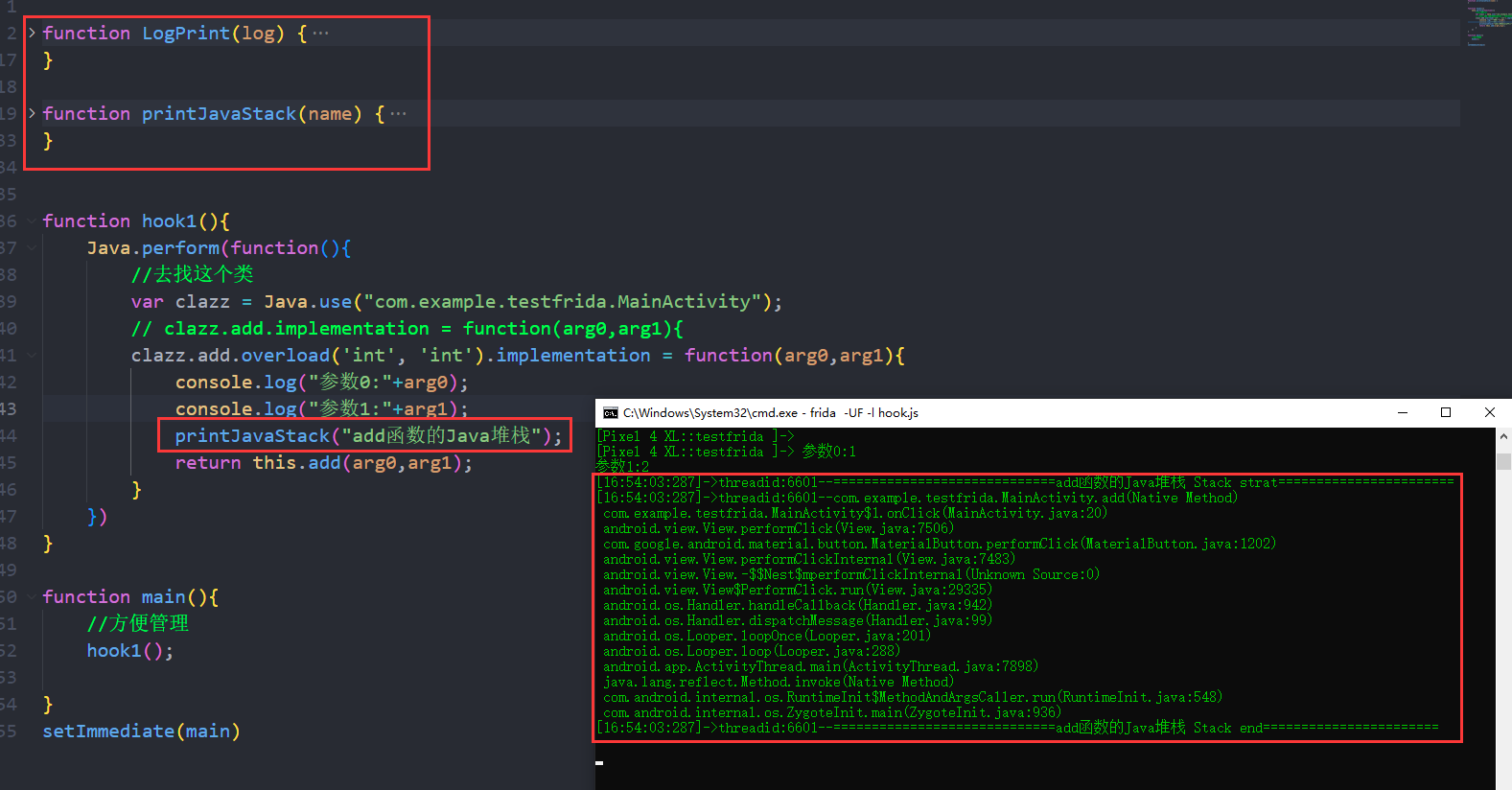


**最后讲一下打印堆栈：**

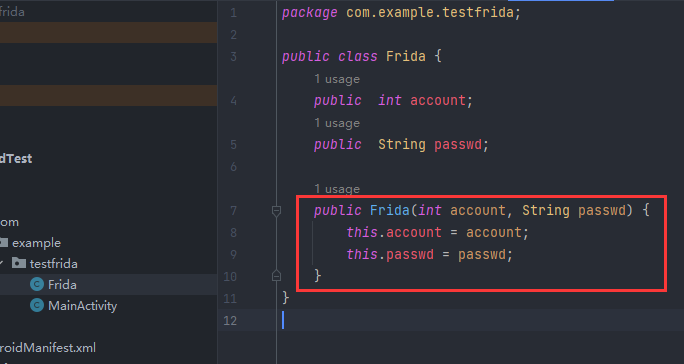
**直接上干货： 使用方式** printJavaStack("自定义字符串")

function LogPrint(log) {  
 var theDate = new Date();  
 var hour = theDate.getHours();  
 var minute = theDate.getMinutes();  
 var second = theDate.getSeconds();  
 var mSecond = theDate.getMilliseconds();  
   
 hour < 10 ? hour = "0" + hour : hour;  
 minute < 10 ? minute = "0" + minute : minute;  
 second < 10 ? second = "0" + second : second;  
 mSecond < 10 ? mSecond = "00" + mSecond : mSecond < 100 ? mSecond = "0" + mSecond : mSecond;  
 var time = hour + ":" + minute + ":" + second + ":" + mSecond;  
 var threadid = Process.getCurrentThreadId();  
 console.log("[" + time + "]" + "->threadid:" + threadid + "--" + log);  
   
}  
   
function printJavaStack(name) {  
 Java.perform(function () {  
 var Exception = Java.use("java.lang.Exception");  
 var ins = Exception.$new("Exception");//$new 创建实例 $init()调用构造函数  
 var straces = ins.getStackTrace();  
 if (straces != undefined && straces != null) {  
 var strace = straces.toString();  
 var replaceStr = strace.replace(/,/g, " \n ");  
 LogPrint("=============================" + name + " Stack strat=======================");  
 LogPrint(replaceStr);  
 LogPrint("=============================" + name + " Stack end======================= \n ");  
 Exception.$dispose();  
 }  
 });  
}

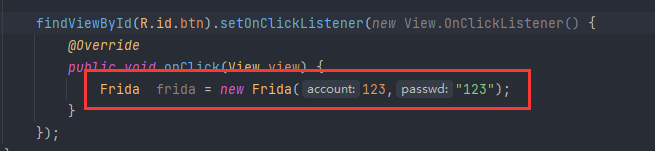
**演示:首先 将代码加入到我们的脚本最上面,然后点击按钮:**



**如何hook构造函数？关键字： $init**



**调用:**



**Hook代码:点击按钮**

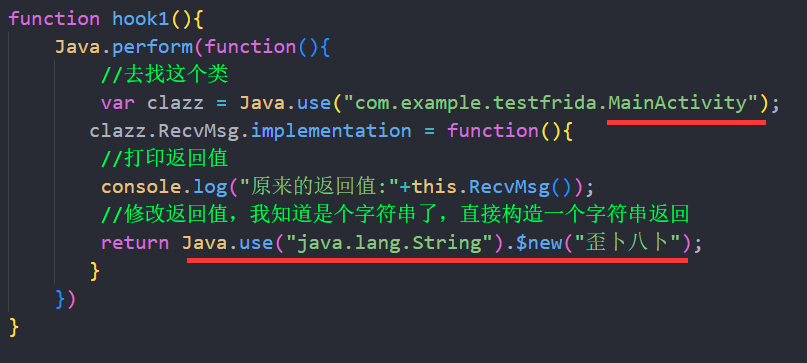


**可以看到，构造函数时没有返回值的，所以得到的result是未定义的，所以在hook构造函数的时候，就直接return this.$init(xx)就可以了**

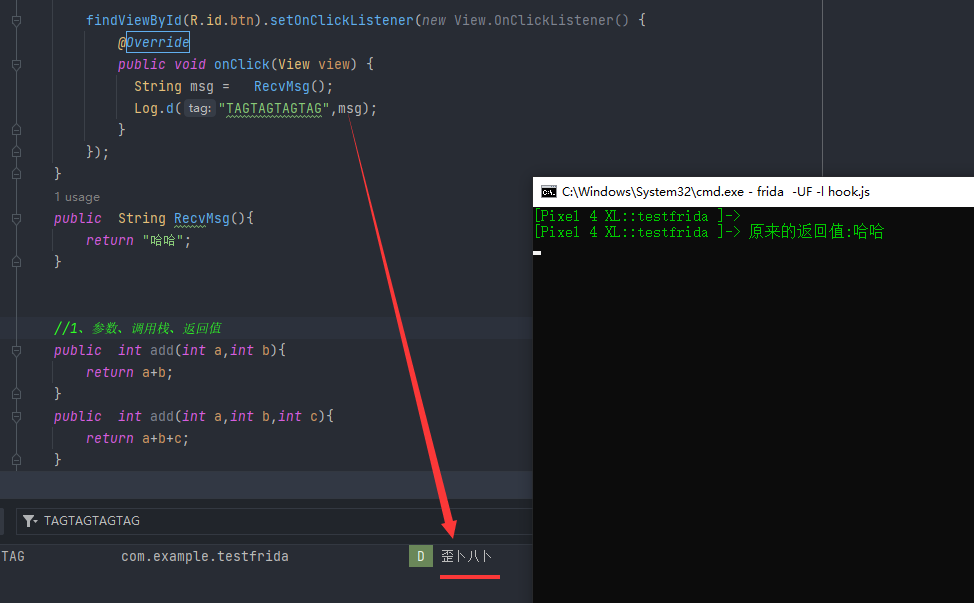
**如何在hook代码中创建java的实例？$new**



**Hook代码：**



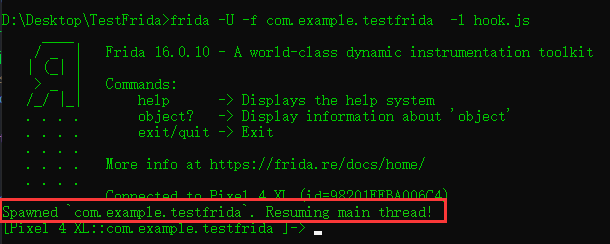
**点击按钮：已经修改为我们的值了**



**如何在程序启动时就hook上?**

**执行命令:**

**将-F 替换成 -f 包名 即可： 执行命令后，应用会重新启动，并在第一时间挂载脚本**



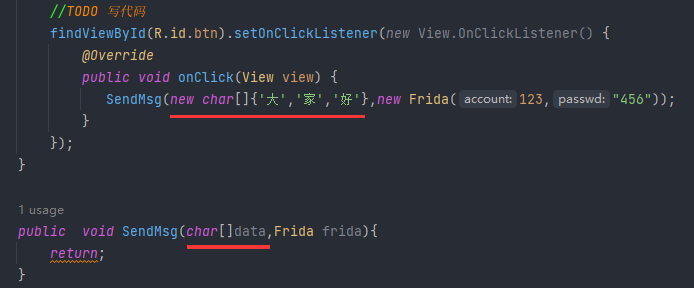
**小结：**

**现在已经学会了如何修改参数、返回值、打印堆栈、hook构造函数、实例化java层的对象。**

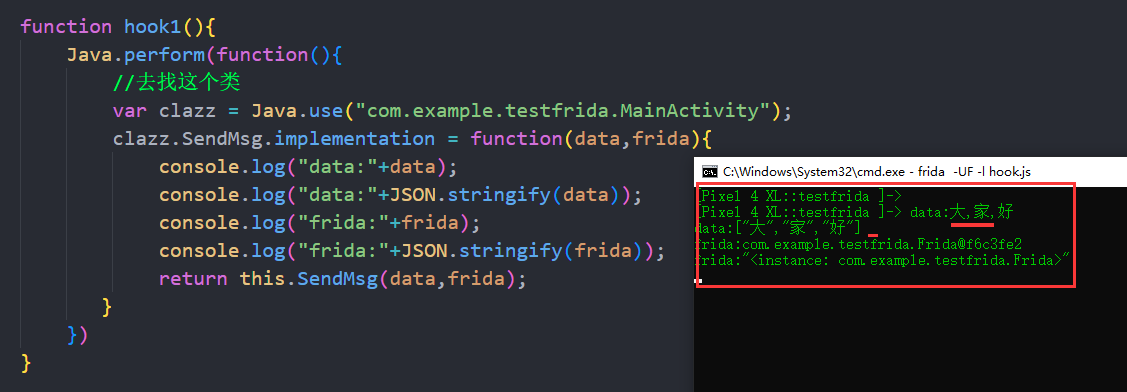
## **4、**进阶操作

### **1、json.stringify()将对象、数组转换成字符串:**

**如果参数是数组或者对象:**



**Hook代码：点击按钮：**

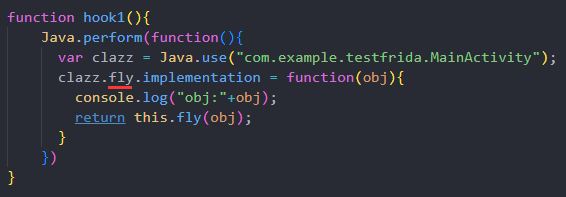


**实际运用：**

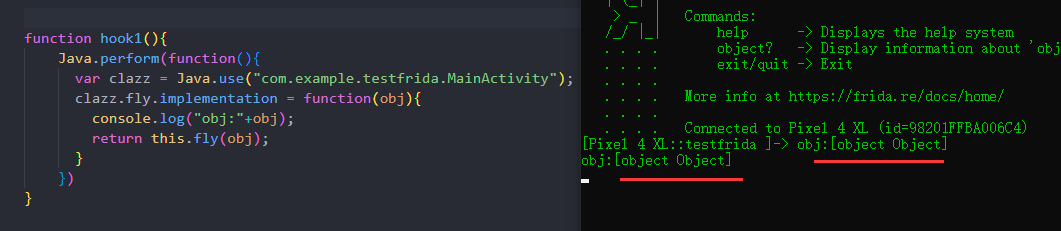
**我们在分析代码的时候，经常会看到函数的参数是个接口，又不知道是谁调用的这个时候打印数据出来就是一个object:**



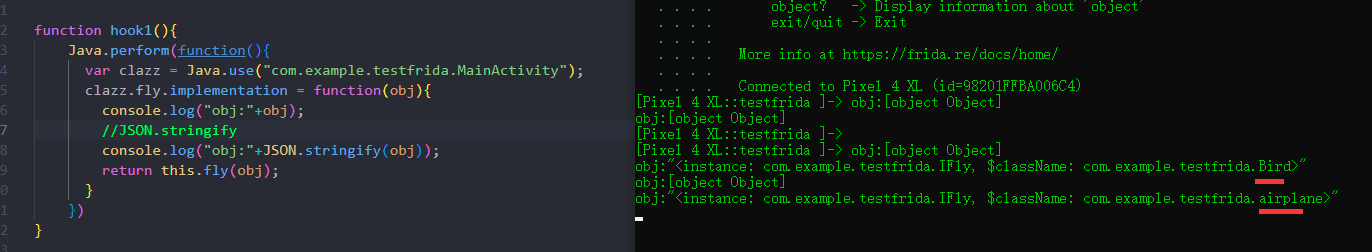
**Hook代码：**



**点击按钮：可以看到，当多态使用的时候，我们直接打印一个参数是接口的对象，打印的并不是对象本身，而是object**



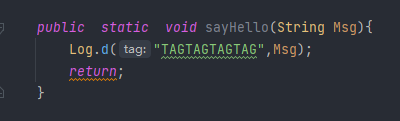
**这个时候我们就使用JSON.stringify:**



**可以打印出真实的对象.**

### **2、主动调用函数(静态、动态)**

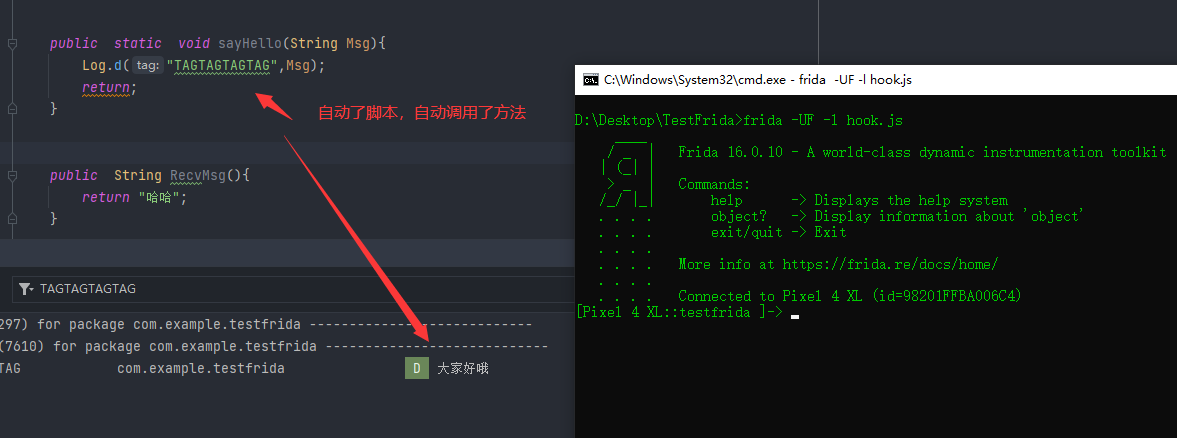
#### **静态方法：**



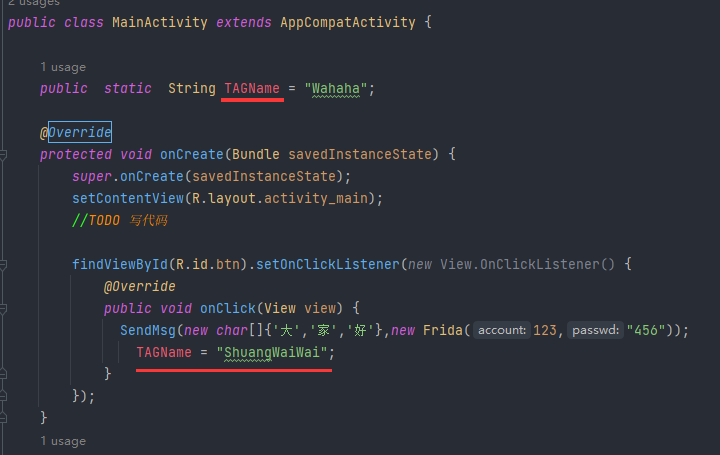
**Hook代码：**



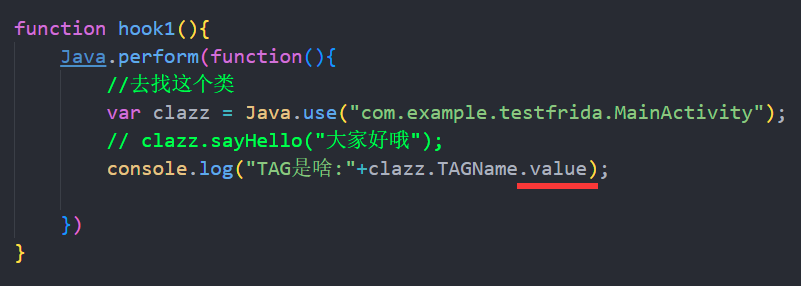
**启动脚本：**



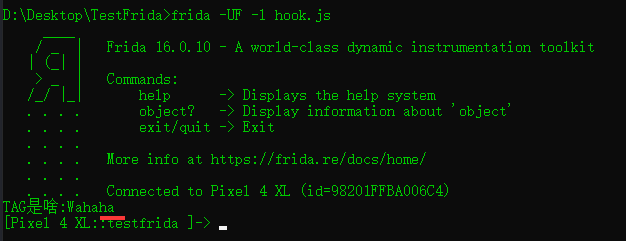
#### **静态属性的修改和获取：**



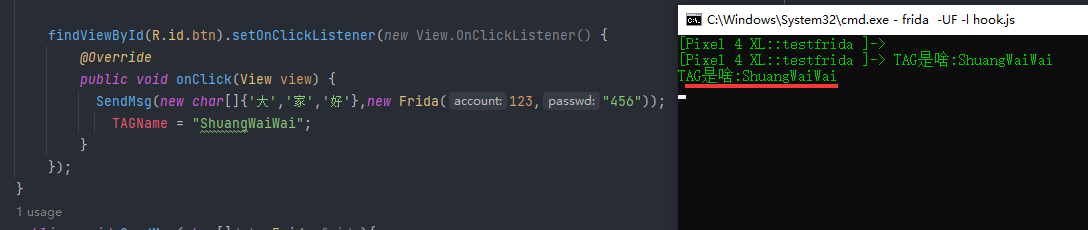
**Hook代码：静态成员变量是值，这里获取值的时候，记得加.value:否者打印出来的就是object（细节拉满）:**



**启动脚本：**



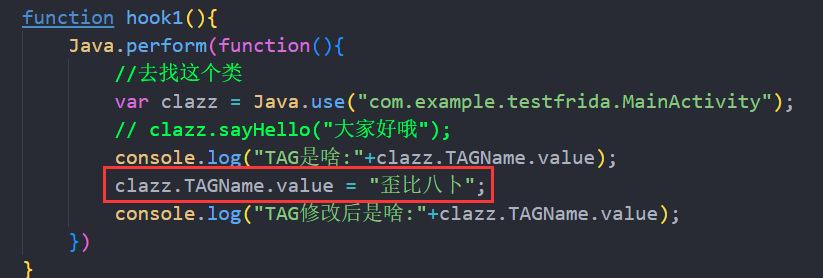
**点击按钮：在保存脚本一次，脚本会重新加载一次：**



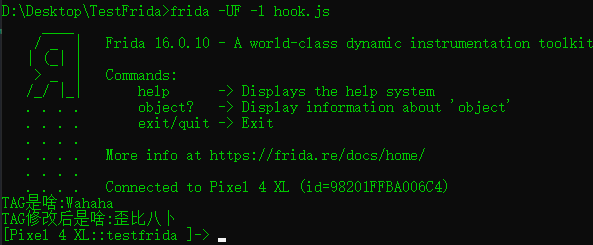
**修改属性：**



**hook代码：**



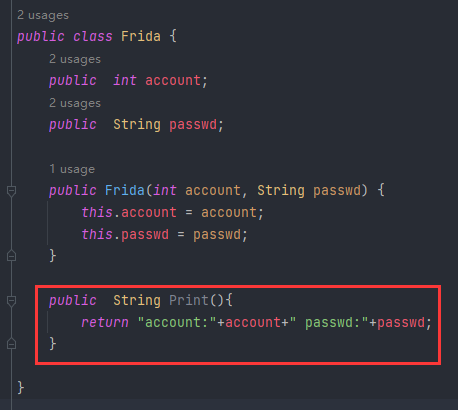
**启动脚本：**

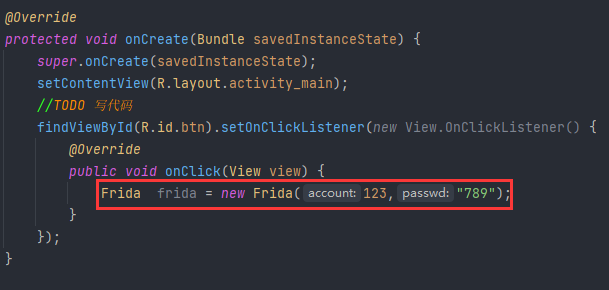


**点击按钮：修改成功**



#### **动态方法：**

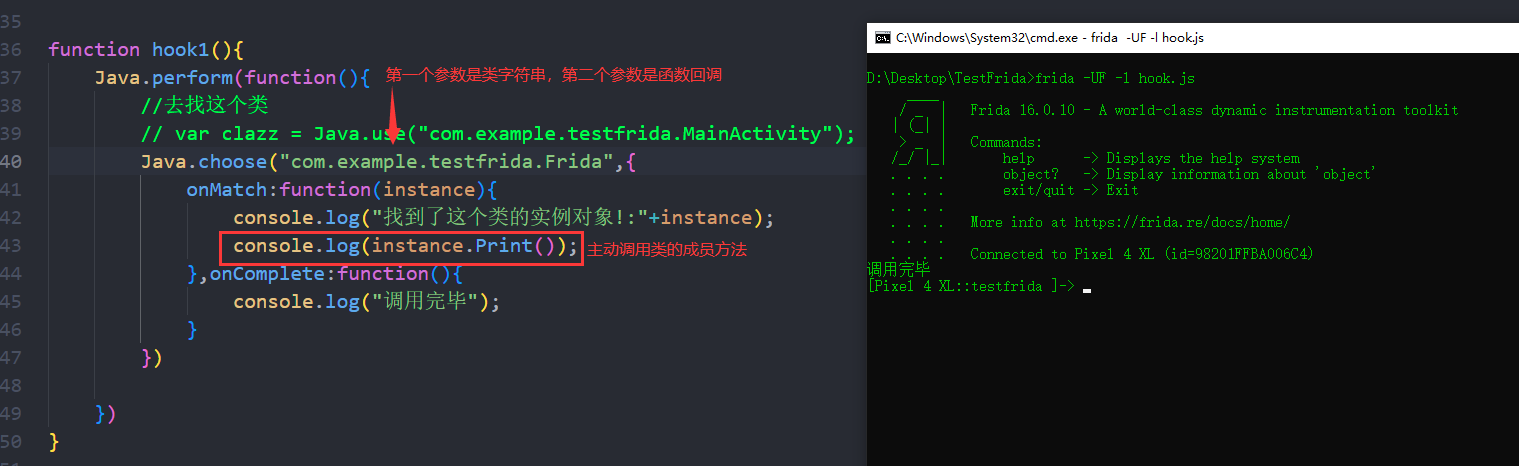




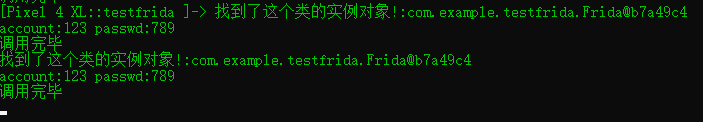
**给Frida类重写了Print方法，在点击按钮后，创建了Frida对象。**

**hook代码： Java.choose()**

**启动脚本，后显示调用完毕，并没有找到实例对象，说明该类并没有被实例化**



**点击按钮： 点击按钮后，会创建Frida对象，我们这里就找到了实例对象，并打印出信息**



**choose常见使用场景: 当应用中一个 对象的生命周期很短时，我们想要获取到该对象存在所存储的信息时，可以通过该方法来找到实例对象**

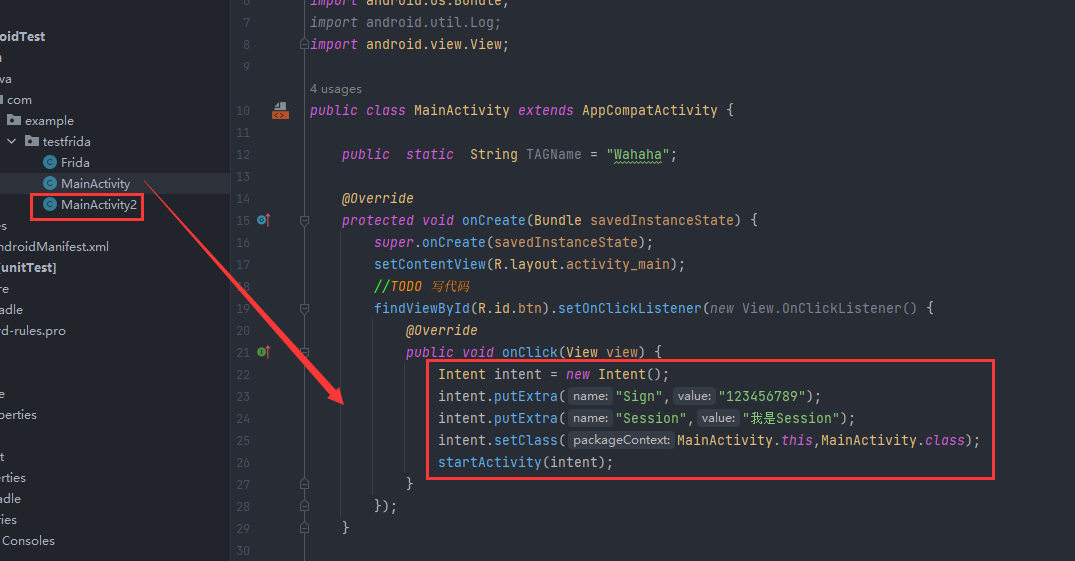
**小结：静态函数没有实例对象，直接通过来访问，动态函数调用需要实例对象。对静态和非静态成员了解的应该知道什么意思.同样的现象有：**

**java反射/C#反射/Android插件化**

## **5、**高级操作/实用操作

### **1、反射机制 Java.case**

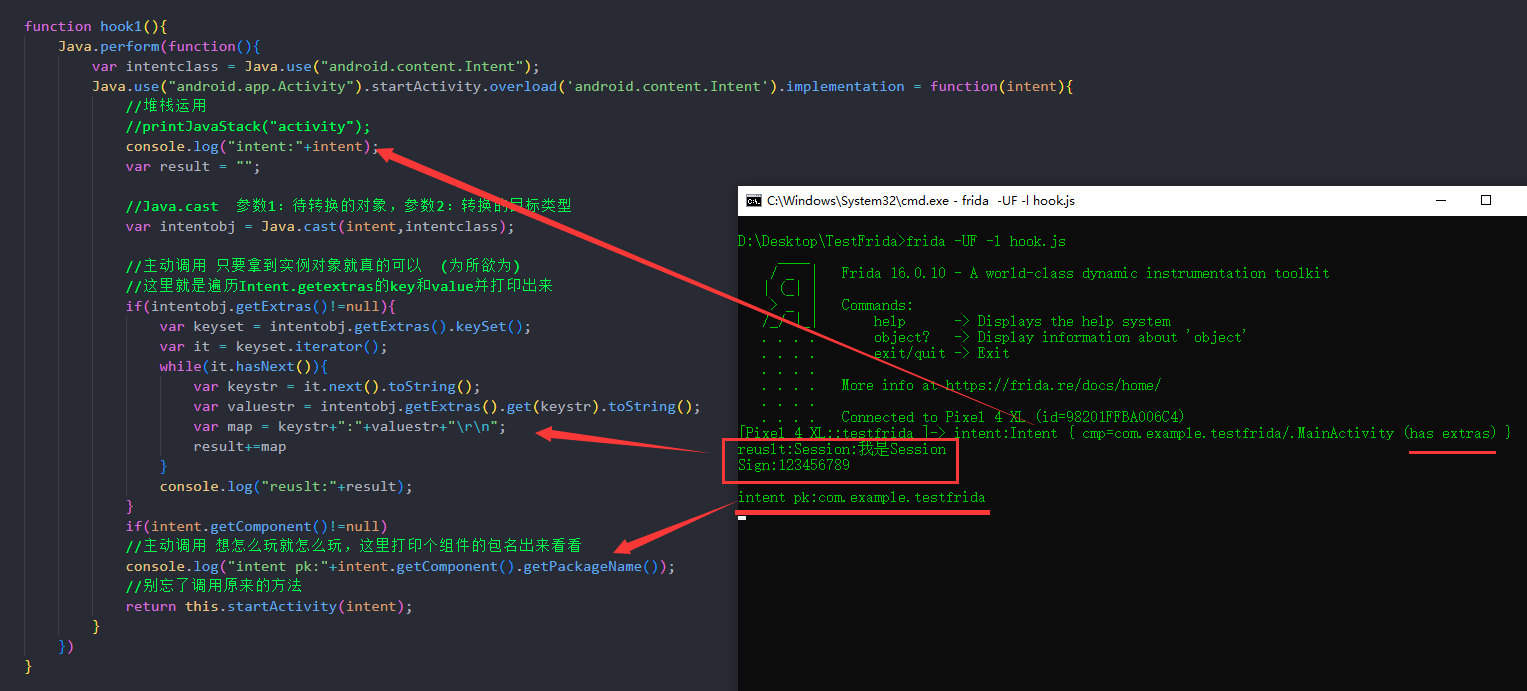
**环境：Android新建一个空的Activity:MainActivity2，点击按钮跳转到Activity2中:**



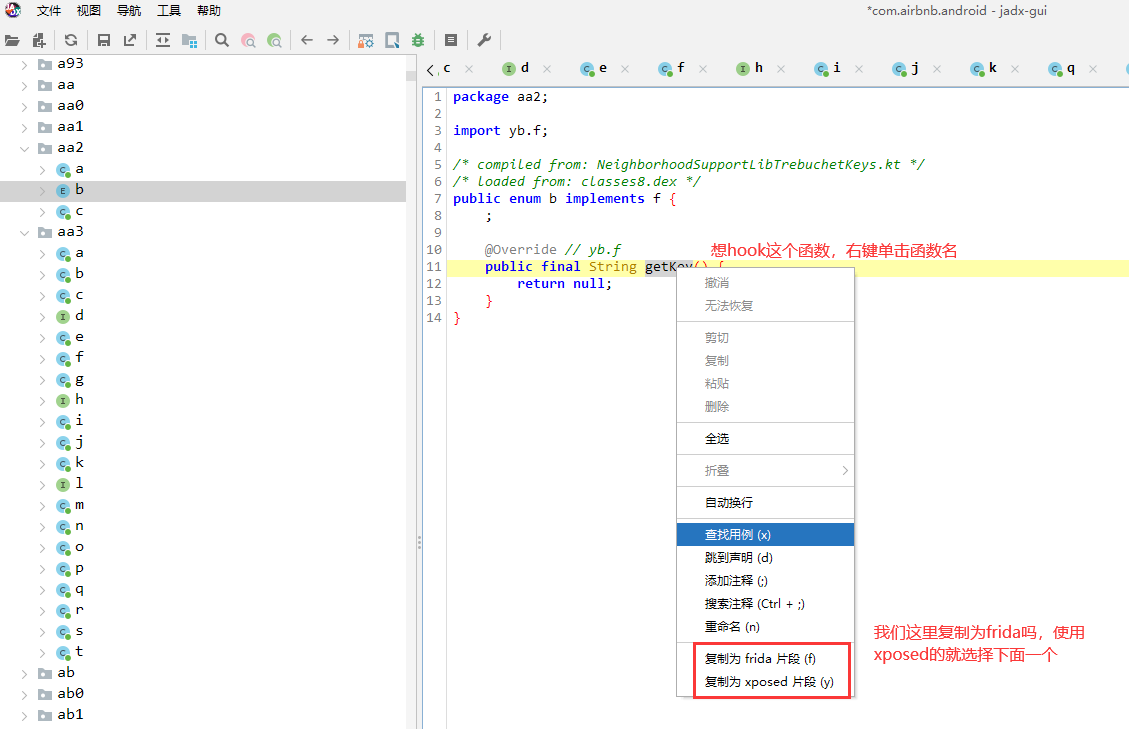
**Hook代码：**



**将参数intent，转为java的intent，就可以随机调用intent可以调用的方法了，这里我们就打印出intent的Extras内容,点击按钮：**



### **2、jadx，自动补全Hook代码：**



**点击 "复制为frida 片段(f)"后，在js 脚本中粘贴即可:Typescript语法**



**他也给我们的格式写好了，所以一般我们用这个的时间更多。**

**特点：当有一些嵌套的方法，不知道怎么hook的时候，可以直接右键点击hook：比如 内部类 的成员方法**

**但是，如果我们想要hook的类、函数，是android系统的，那么就需要 我们自己编写代码了去实现了，就不能复制了.**

### **3、枚举类加载器 Java.eumeratClassLoaders**

**何时使用类加载器:**

**1、动态加载：当app的dex是动态加载的时候，可能会在点击或者达到条件的情况下会动态加载一些类，所以我们一开始就去hook这些类，那么就 hook不上，会报错：找不到类**

**2、自定义类加载器加载**

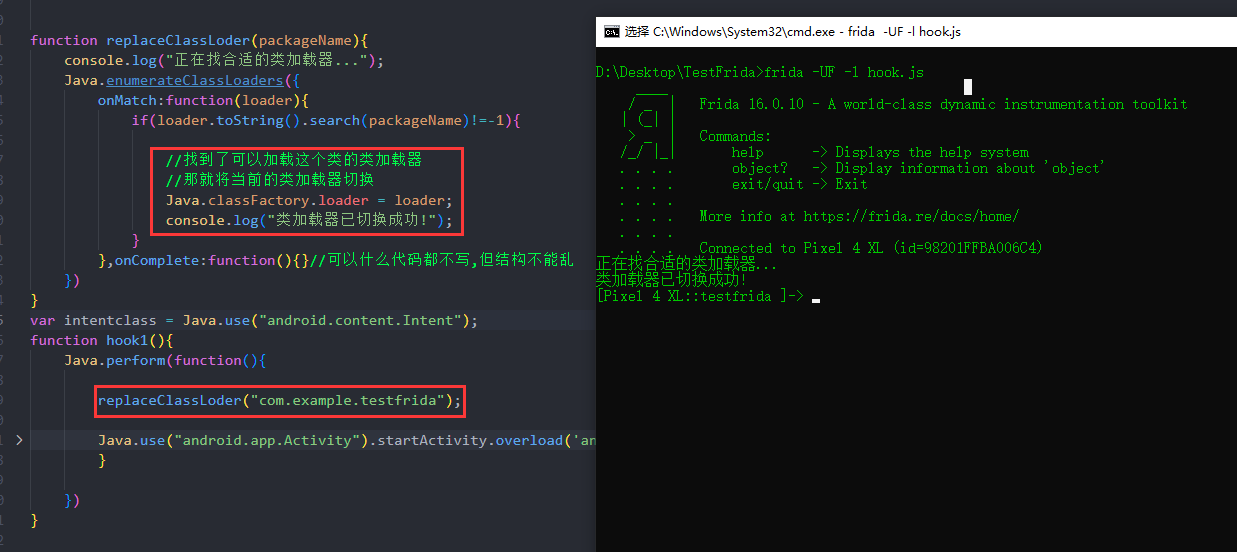
**3、插件化加载dex**

**这个时候，我们先排除一下情况：**

**1、类名确实没写错且是完整的**

**2、手机上的应用和jadx里面分析的包是对应的（检查是否应用自己更新了）。**

**那么这个时候就要考虑使用我们的类加载器了：**



**Code:**

function replaceClassLoder(packageName){  
 console.log("正在找合适的类加载器...");  
 Java.enumerateClassLoaders({  
 onMatch:function(loader){  
 if(loader.toString().search(packageName)!=-1){  
   
 //找到了可以加载这个类的类加载器  
 //那就将当前的类加载器切换  
 Java.classFactory.loader = loader;  
 console.log("类加载器已切换成功!");  
 }  
 },onComplete:function(){}//可以什么代码都不写,但结构不能乱  
 })  
}

**原理：就是枚举应用用到的类加载器，用类加载器去加载我们的类，如果能加载，我们就切换我们的类加载器，切换类加载器之后，我们现在的类加载器就可以加载这个类了，就找到这个类了，就解决了hook 类报找不到类的报错了，就能hook我们想要的类了。**

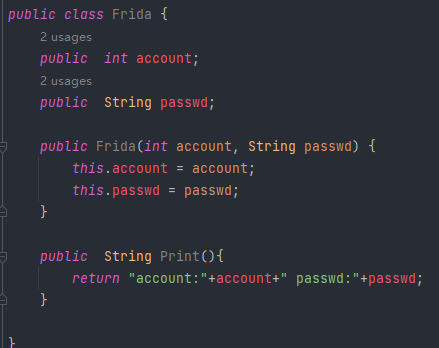
**当然，你要不嫌麻烦，可以在每一次hook类时，都去检查类加载器。**

### **4、动态加载Dex文件 Java.openClassFiel("dex文件路径").load();**

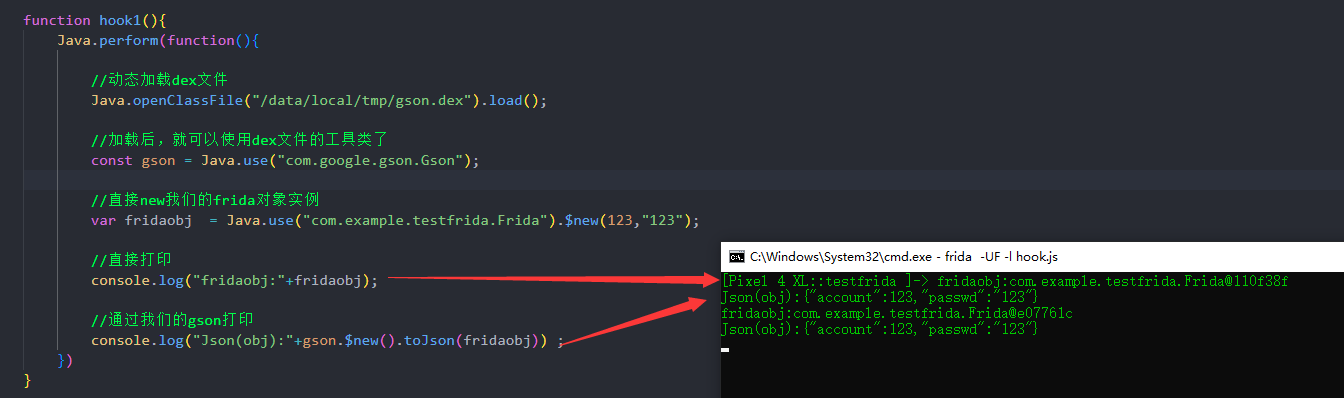
**使用第三方dex来将类打印成json格式数据:**

**第一步：将gson.dex 文件 push 到手机上**

**第二步：hook代码动态加载gson.dex文件**

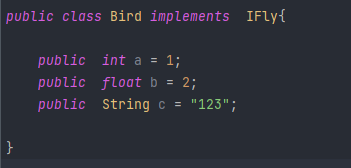


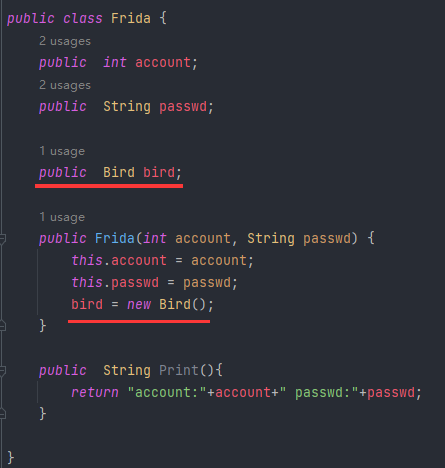
**hook代码，我们这里延时打印这个frida的对象，将其转为Json格式输出**



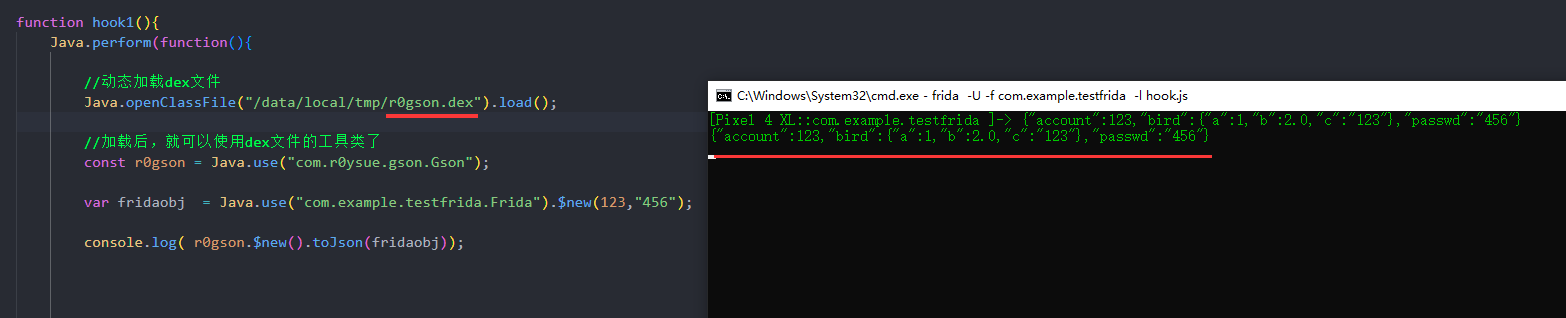
**注意事项： 不是什么类都可以这样去打印，结合实际需求 。太过复杂的类也是打印不出来的**

**这里提供一个网上提供的三方的 对原Gson封装后的 Gson工具:r0Gson.使用方法一样的:**





**Hook代码： 同样可以打印：**



**上干货,直接用（推荐使用r0gson）**

[请至钉钉文档查看附件《gson.dex》](https://alidocs.dingtalk.com/i/nodes/oP0MALyR8kR5B0D0UmrEyRDnJ3bzYmDO?doc_type=wiki_doc&iframeQuery=anchorId%253DX02lgopsp88943c49sku29)

[请至钉钉文档查看附件《r0gson.dex》](https://alidocs.dingtalk.com/i/nodes/oP0MALyR8kR5B0D0UmrEyRDnJ3bzYmDO?doc_type=wiki_doc&iframeQuery=anchorId%253DX02lgou3dae61d5s1m8ll5)

**小结：实用操作里面我们讲解了：**

**1、反射机制(Java.cast)**

**解决问题：对hook的参数进行转换为真实类型的对象**

**2、jadx补全**

**解决问题：重复的劳动**

**3、枚举类加载器**

**解决问题：hook的类不存在、找不到，**

**无法解决问题：**

**1、app有这个类，但是没有调用,**

**2、jadx混淆重命名了**

**4、动态加载dex文件**

**解决问题：对象数据的清晰打印**

**这些都是工作中常用的知识点，掌握这些知识点，解决问题可以事半功倍。**

### **5、**整理的**Frida库文件: base.js**

[请至钉钉文档查看附件《base.js》](https://alidocs.dingtalk.com/i/nodes/oP0MALyR8kR5B0D0UmrEyRDnJ3bzYmDO?doc_type=wiki_doc&iframeQuery=anchorId%253DX02lgpzv7xmqwu0c28or1b)

**只需要在你的脚本目录下有这个文件就可以调用了，不需要在该脚本里面写代码。当然也可以写.我们更希望他是作为一个库来使用**

## **6、Frida其他领域的探讨**

### **-1、Linux配置frida**

### **-2、Linux配置多版本frida**

### **-3、配套使用工具 objection、wallbreaker**

### **-4、由frida延申出的各种可视化工具以及功能库**

### **-5、frida 脱壳**

### **-6、frida 反调试、反反调试、重编译frida、对抗**

### **1、frida-RPC调用**

### **2、frida Native Hook**

### **3、frida 密码学/魔改 破解（摘要算法、数字签名、对称、非对称加密算法）**

### **4、frida 自定义算法、 网络协议破解**

### **5、frida 特殊场景操作**

### **6、frida 动态加载dex破解算法**

## [优先级: 4] **总结（Summary）**

Frida是一个功Frida是一个功能强大的动态分析和调试工具，可以帮助安全研究人员、逆向工程师和开发人员分析和修改应用程序的行为，从而发现和修复漏洞。以下是Frida的一些主要特点和优势：

* 动态分析：Frida可以在应用程序运行时分析和修改其行为，克服了传统静态分析和调试工具的限制。
* 跨平台：Frida可以在多个平台上运行，包括Windows、macOS、Linux、Android、iOS等。
* 易于使用：Frida提供了简单易用的API和命令行工具，使得使用者可以快速上手并进行动态分析和调试。
* 灵活性：Frida支持多种编程语言和脚本语言，包括JavaScript、Python、C#等，并提供了许多插件和扩展，使得用户可以根据自己的需要进行自定义和扩展。
* 安全性：Frida具有高度的安全性和稳定性，可以通过多种技术绕过应用程序的安全保护机制，并且不会对目标应用程序造成任何损害。

**文章可能存在误解,大家可以一起交流!**

# **持续更新：**

### **update Date:2023.4.27**

**1、如果hook的类中属性和方法重名了，想打印属性的值， 需要在属性前面加上 下划线 ：**

**比如person类中有属性money和方法：money,想要获取属性的值:**