1. **强制挂接**

   ps | grep “xxx”　　查找进程 id 值

   kill -19 pid　　暂停进程

   ida 挂接，dump 出需要的数据

   ida 脱离挂接

   kill -18 pid　　继续进程

2. **内存 dex 定位**

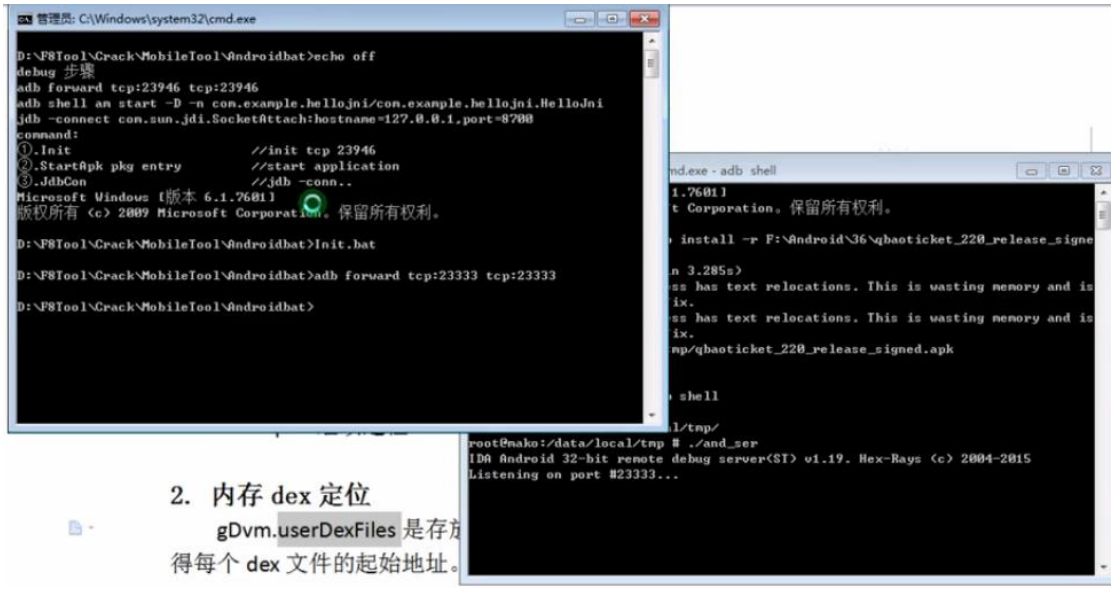
   gDvm.userDexFiles 是存放 dex cookie(DexOrJar 结构)的地方，因此可以通过遍历该结构获取每个 dex 文件的起始地址。

3. **Dex 重构**

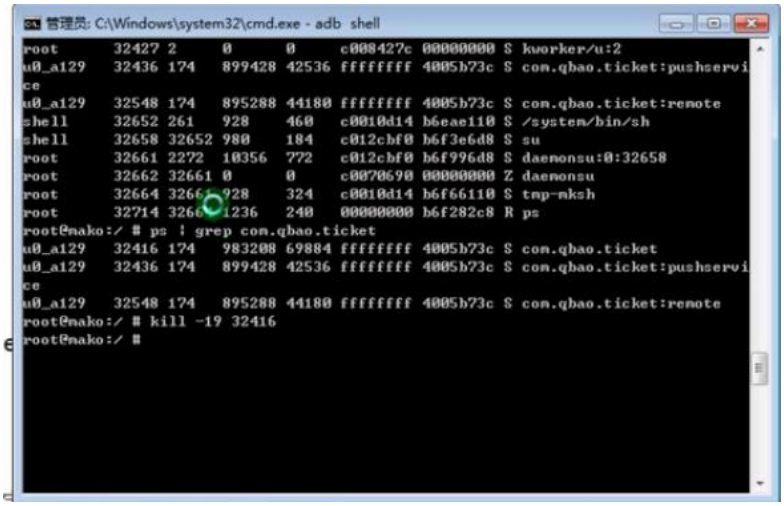
   通过分析内存中的 dex 存储结构，完成对整个 dex 文件的 dump。

4. **Demo 演示（爱加密壳）**

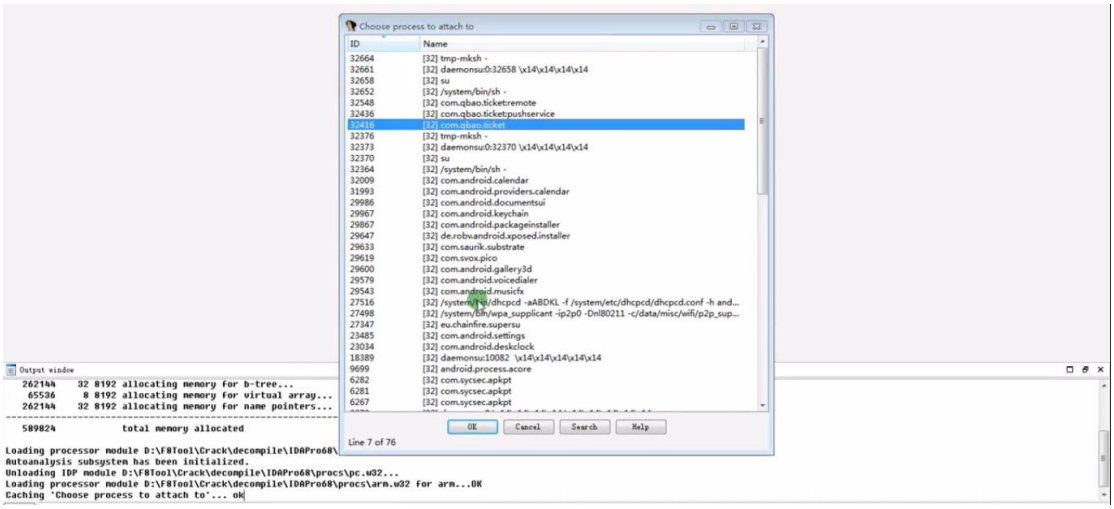
   安装目标 app，然后开启调试端口并进行端口转发



在 ida 挂接之前需要暂停进程，为了不让 ida 被壳发现



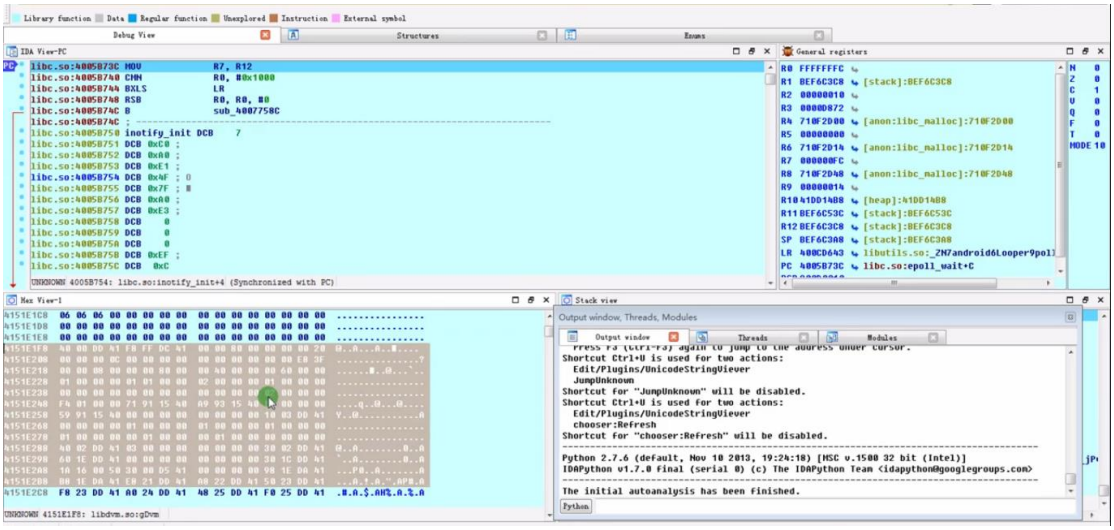
然后使用 ida 完成挂接



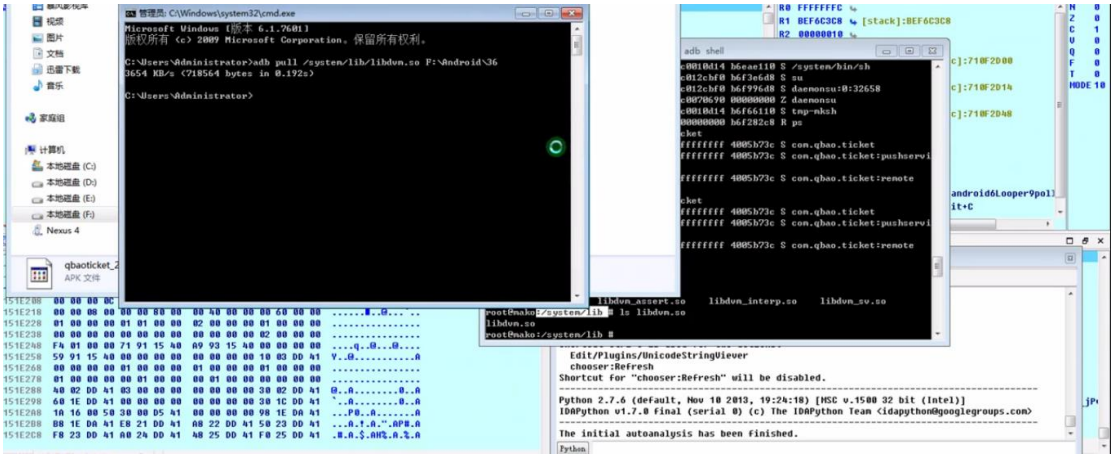
1) 找到 gDvm.userDexFiles 的位置
2) 遍历该结构来找到加载过的 dex 文件数据

关键结构：struct DvmGlobals gDvm; HashTable* userDexFiles;

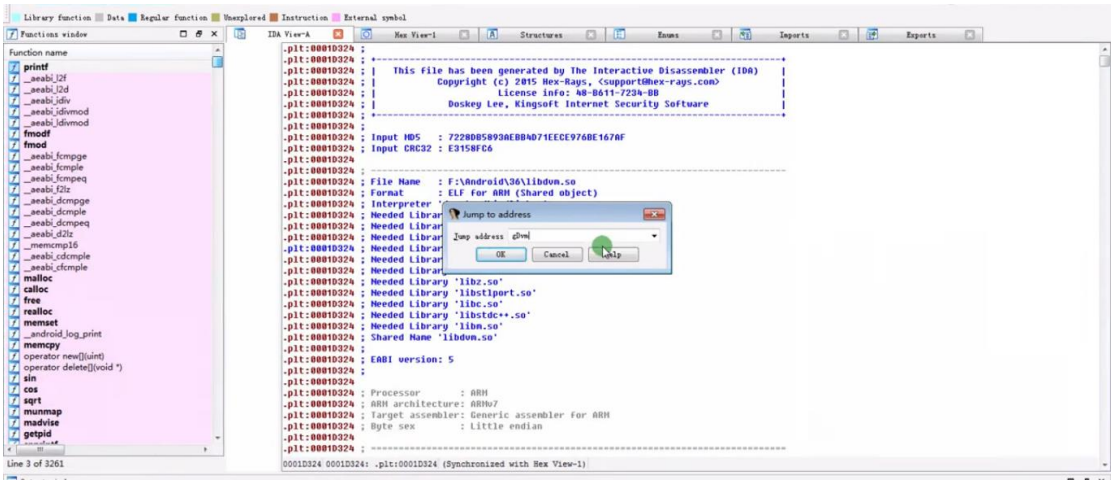
搜索到 gvm 相关的结构体

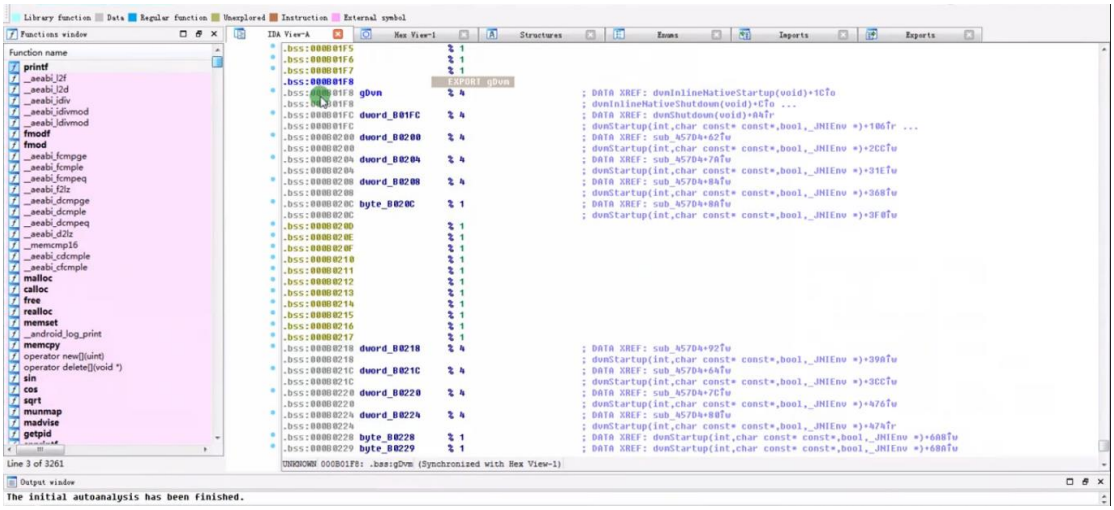


将相关的 lib 文件 down 下来

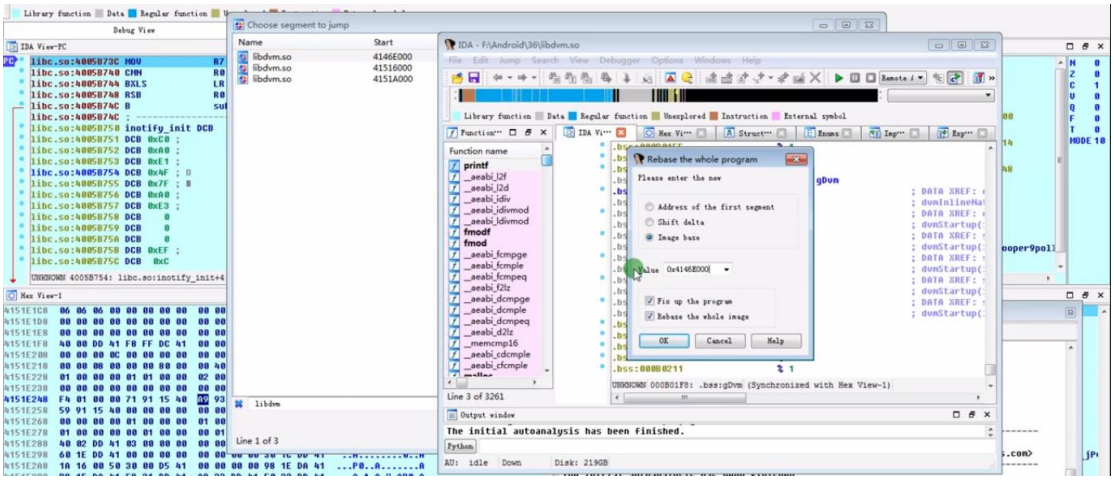


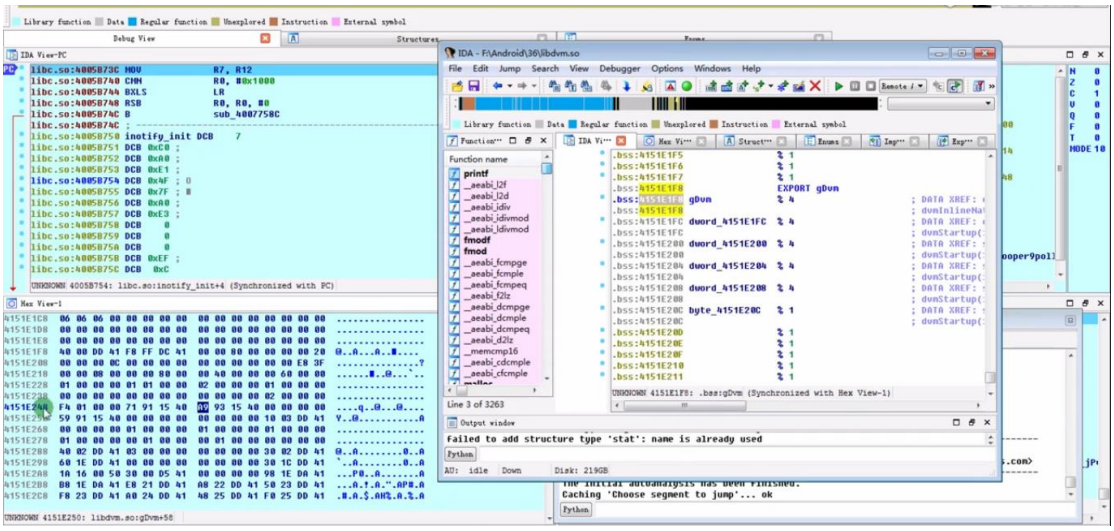
down 下来的 lib 文件用 ida 打开并定位到 gDvm



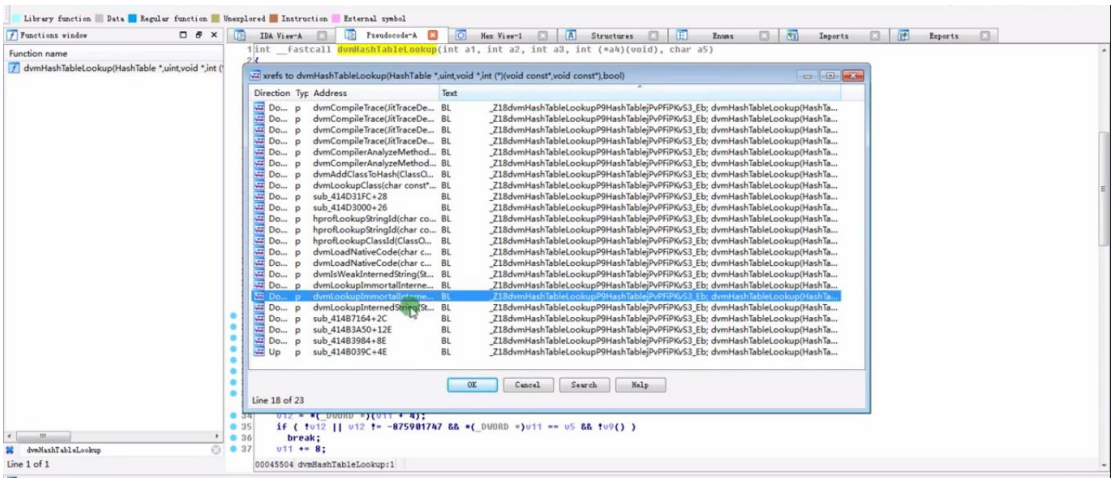


完成与内存位置的对齐，方便寻址和遍历

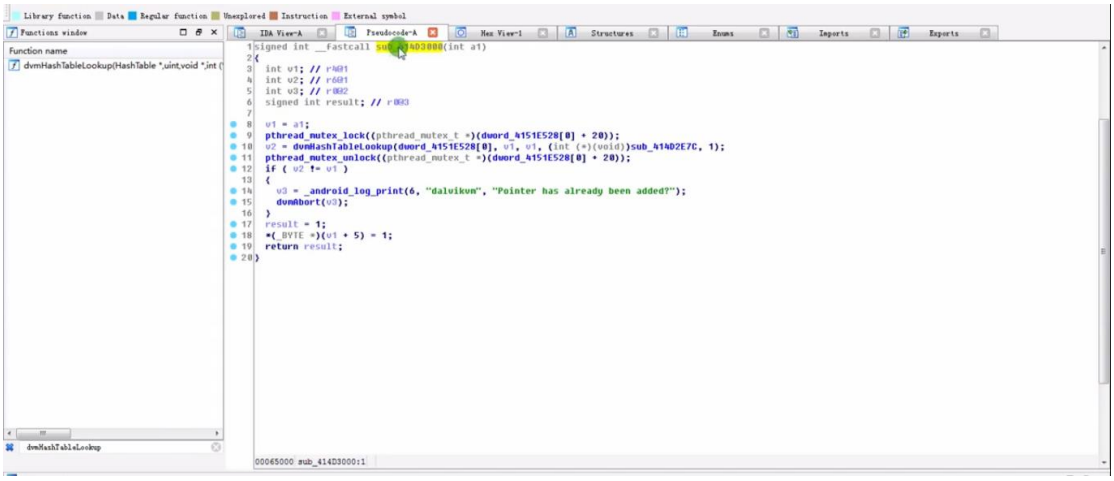




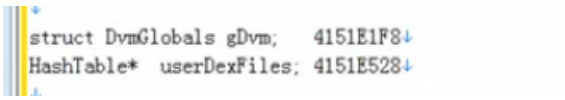
查找相关函数确定 gDvm 的调用位置



根据源代码中的相关信息，找到对应的函数位置

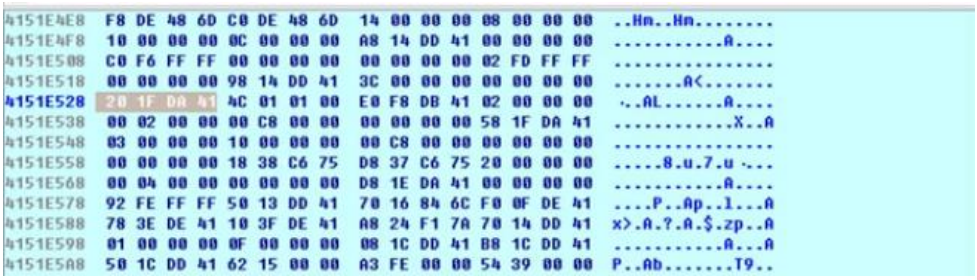


确定出 struct DvmGlobals gDvm; HashTable* userDexFiles;的地址

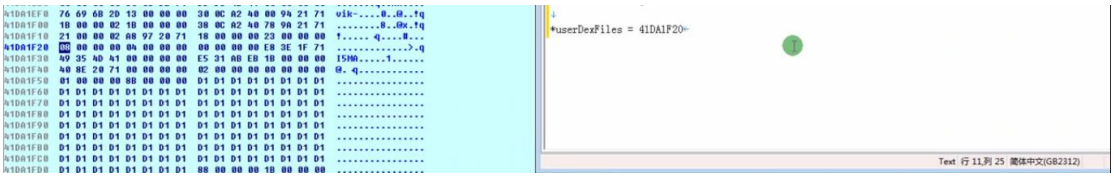


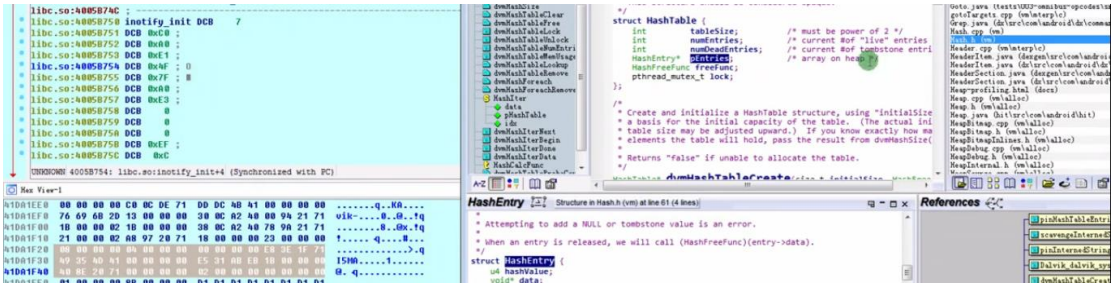
得到：userDexFiles = gDvm + 0x330

以此也确定了 userDexFiles 的位置

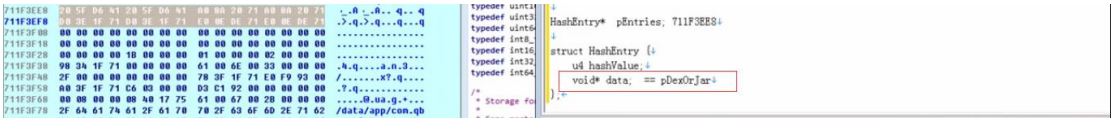


根据指针地址跳转过去，得到 userDexFiles 的 HashTable 结构体

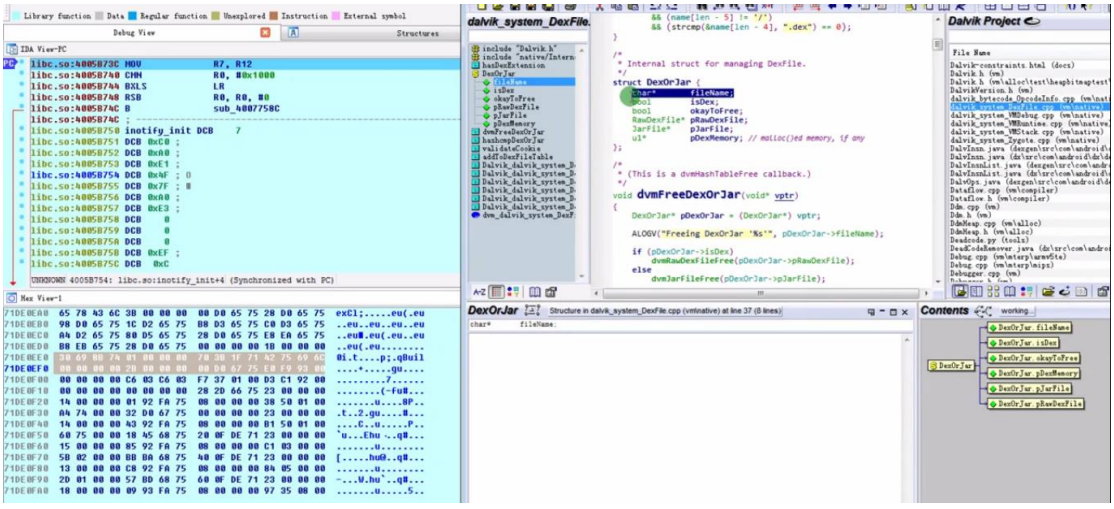




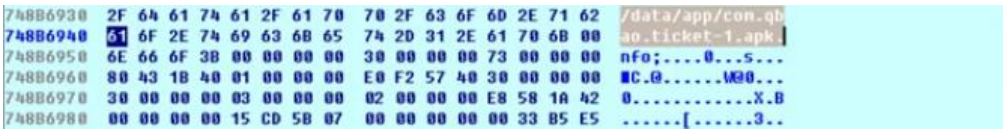
得到 HashEntry* pEntries 地址: 711F3EE8 并跳转



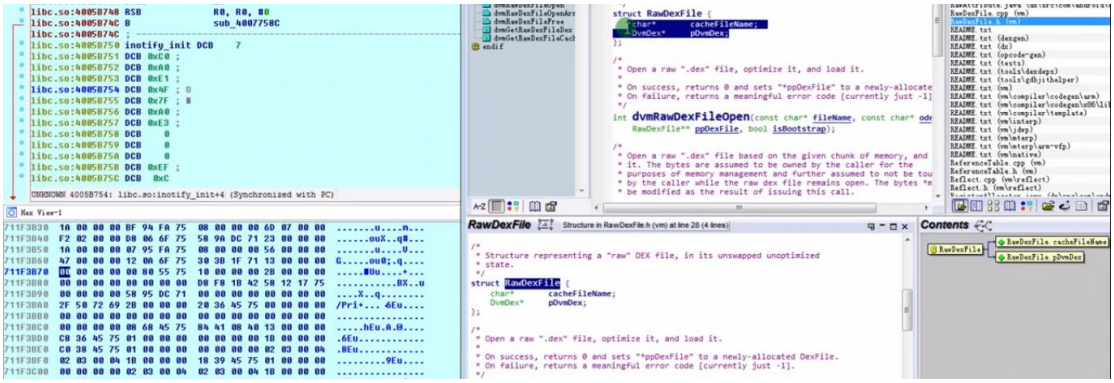
分析得到其中一个 DexOrJar 的结构体地址为：71DE0EE0；跳转过去得到结构体的内容



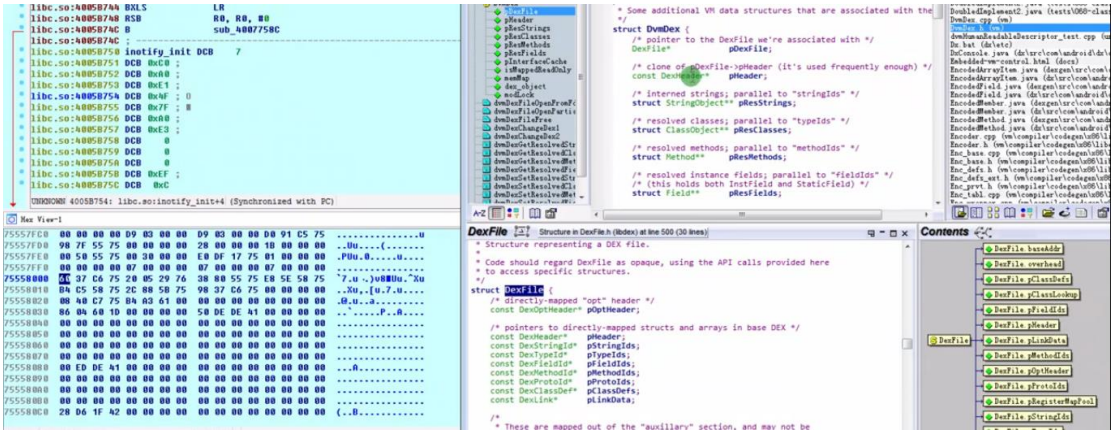
跳转到 fileName 位置，发现存放的就是壳的 dex 文件数据



回到刚才结构体的位置，确定 pRawDexFile: = 711f3b70，跳转过去

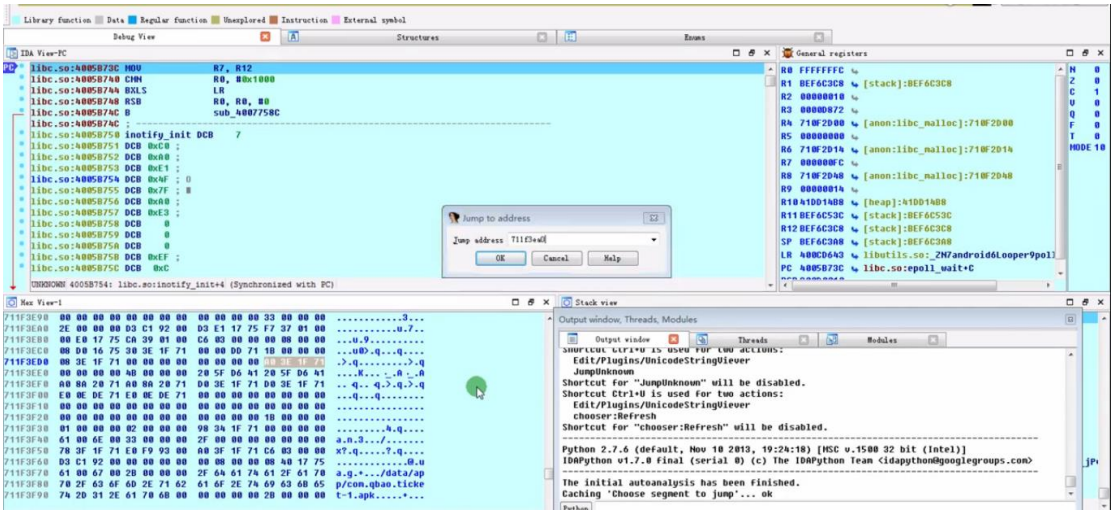


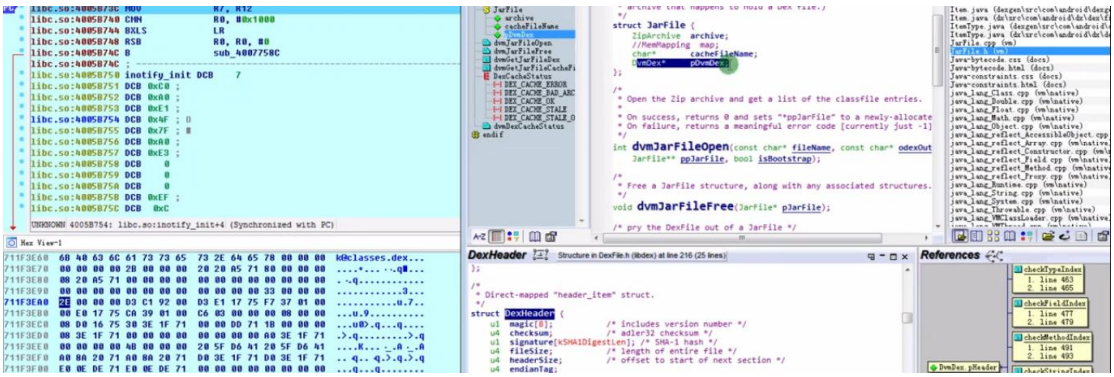
得到 pDvmDex 的指针，跳转过去



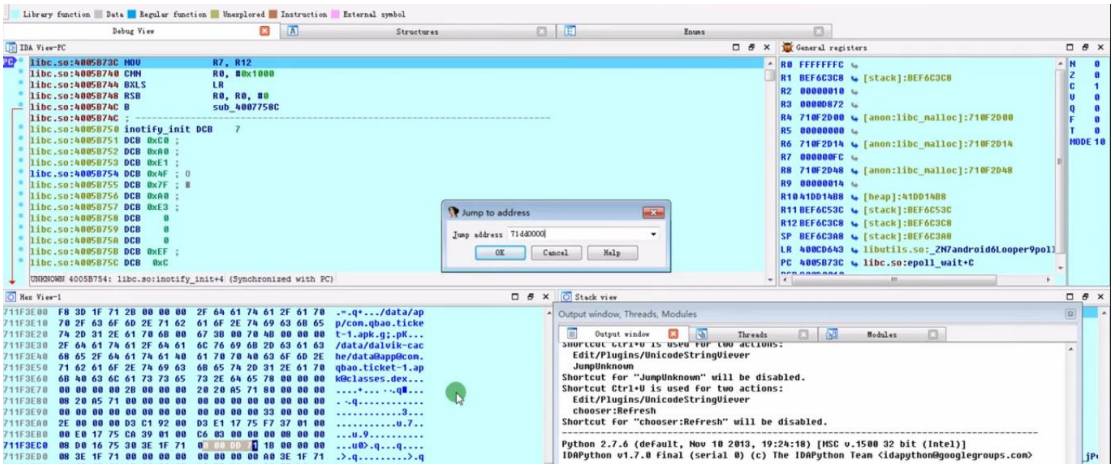
取 pHeader 数据地址，跳转发现 dex 头被抹去成为无效数据了，换到其他 DexEntries 按照相同的步骤进行

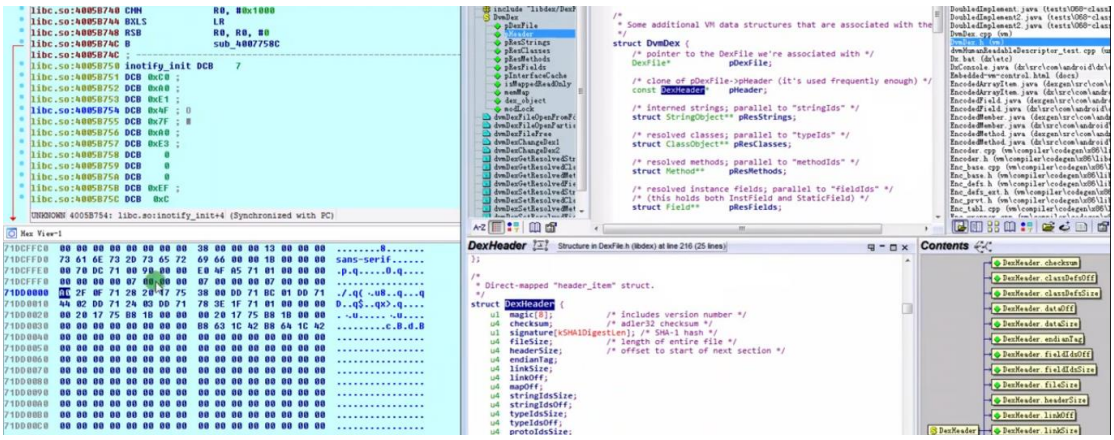
取倒数第二个，这次是 jar 结构，跳转过去



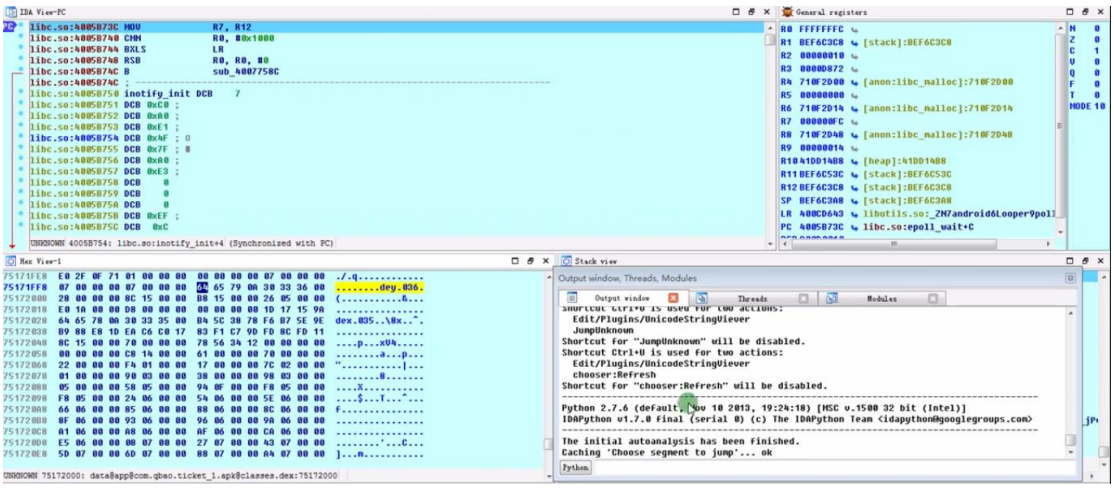


跳转到 pDexDvm 的结构



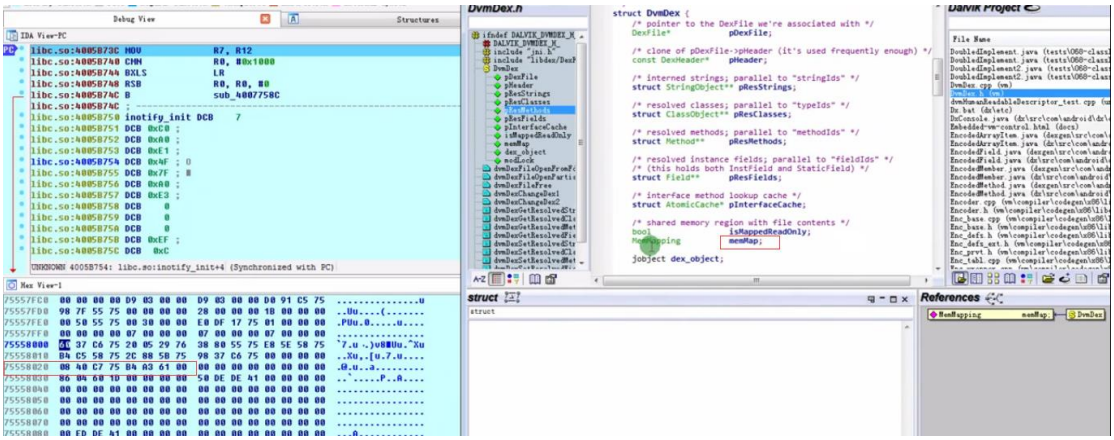


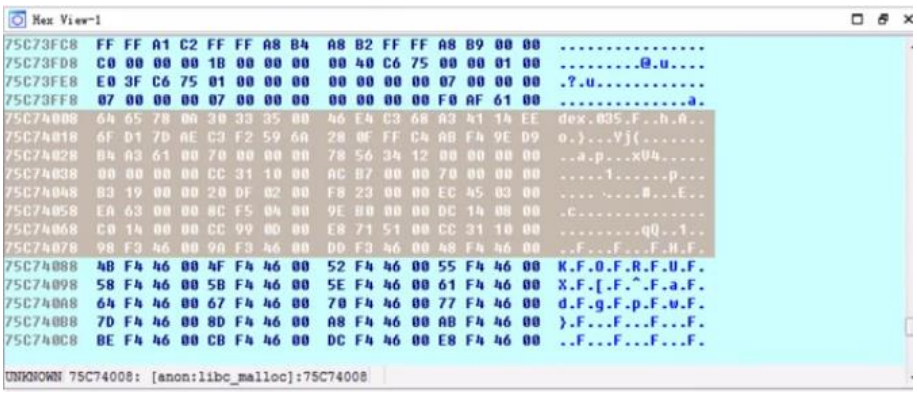
跳转之后找到 pHeader 位置并跳转，得到 dex 文件位置



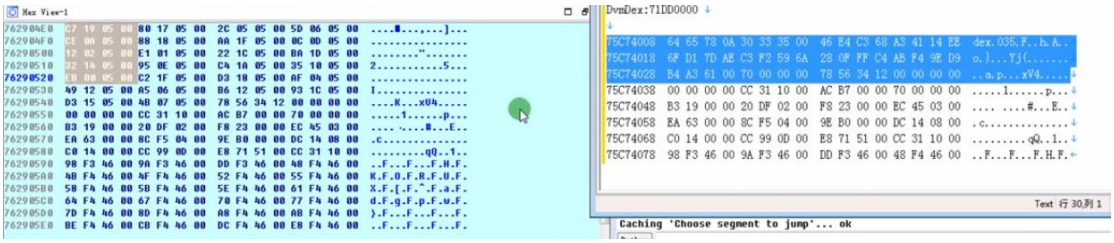
不过由于该 dex 数据不完整，所以要进行修复

根据 memMap 找到原始没有被抹去的 dex 文件数据位置

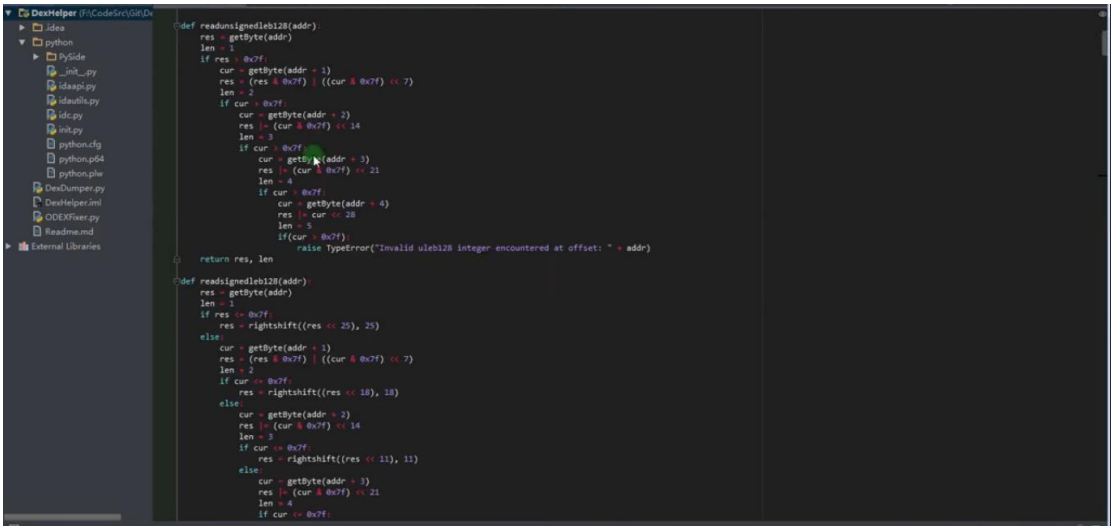


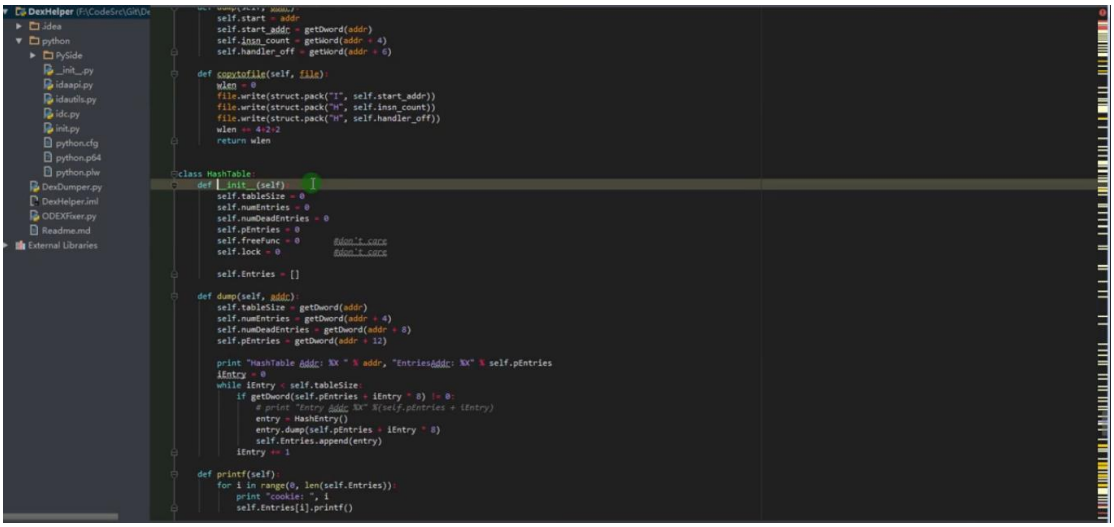


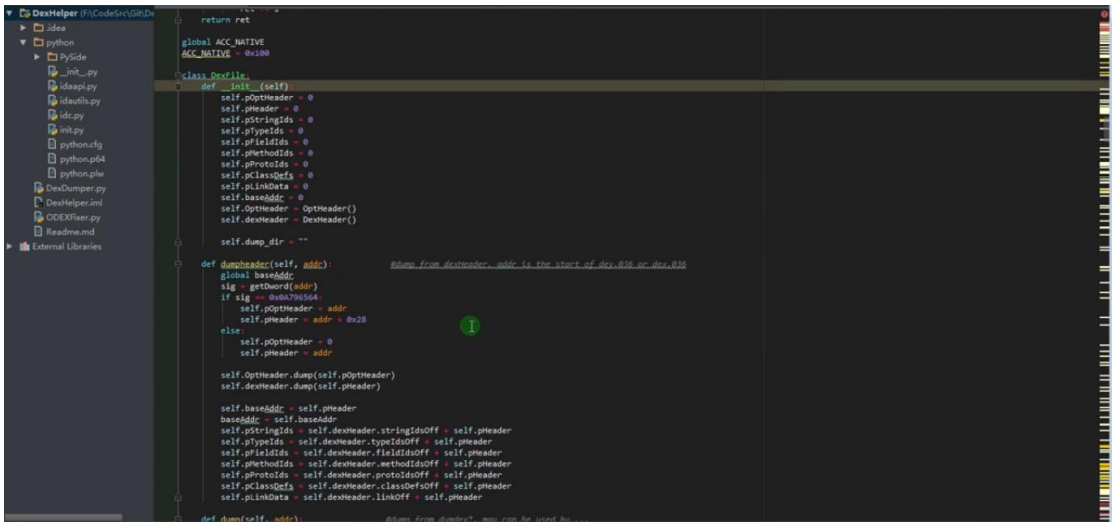
没有被抹去的 dex 与被抹去 dex 的对比


Hex View-1
DvmDex:71DD0000
Caching 'Choose segment to jump'... ok

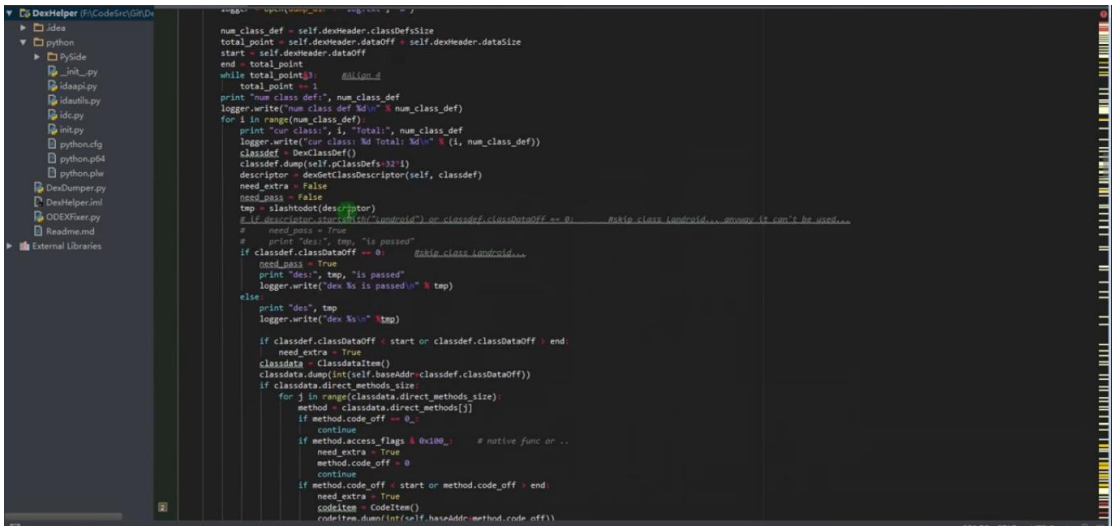
dump 脚本：



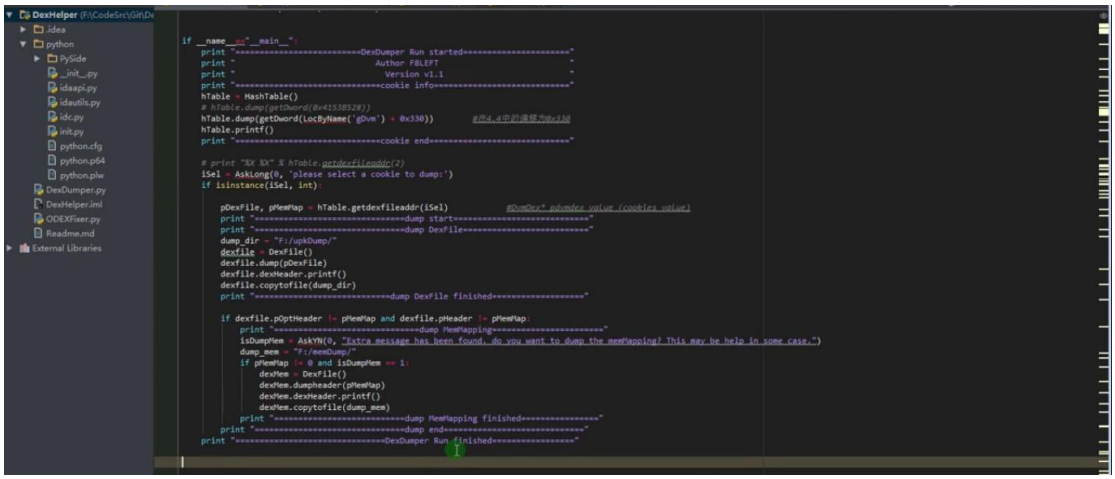




```
    return ret

global ACC_NATIVE
ACC_NATIVE = 0x100

class DexFile:
    def __init__(self):
        self.pOptHeader = 0
        self.pHeader = 0
        self.pStringIds = 0
        self.pTypeIds = 0
        self.pFieldIds = 0
        self.pMethodIds = 0
        self.pProtoIds = 0
        self.pClassDefs = 0
        self.pLinkData = 0
        self.baseAddr = 0
        self.OptHeader = OptHeader()
        self.dexHeader = DexHeader()

        self.dump_dir = ""

    def dumpheader(self, addr):            #dump from dexHeader, addr is the start of dex,035 or dex,038
        global baseAddr
        sig = getDword(addr)
        if sig == 0x0A796564:
            self.pOptHeader = addr
            self.pHeader = addr + 0x28
        else:
            self.pOptHeader = 0
            self.pHeader = addr

        self.OptHeader.dump(self.pOptHeader)
        self.dexHeader.dump(self.pHeader)

        self.baseAddr = self.pHeader
        baseAddr = self.baseAddr
        self.pStringIds = self.dexHeader.stringIdsOff + self.pHeader
        self.pTypeIds = self.dexHeader.typeIdsOff + self.pHeader
        self.pFieldIds = self.dexHeader.fieldIdsOff + self.pHeader
        self.pMethodIds = self.dexHeader.methodIdsOff + self.pHeader
        self.pProtoIds = self.dexHeader.protoIdsOff + self.pHeader
        self.pClassDefs = self.dexHeader.classDefsOff + self.pHeader
        self.pLinkData = self.dexHeader.linkOff + self.pHeader

    def dump(self, addr):                  #dump from dumpler, may can be used by ...
```

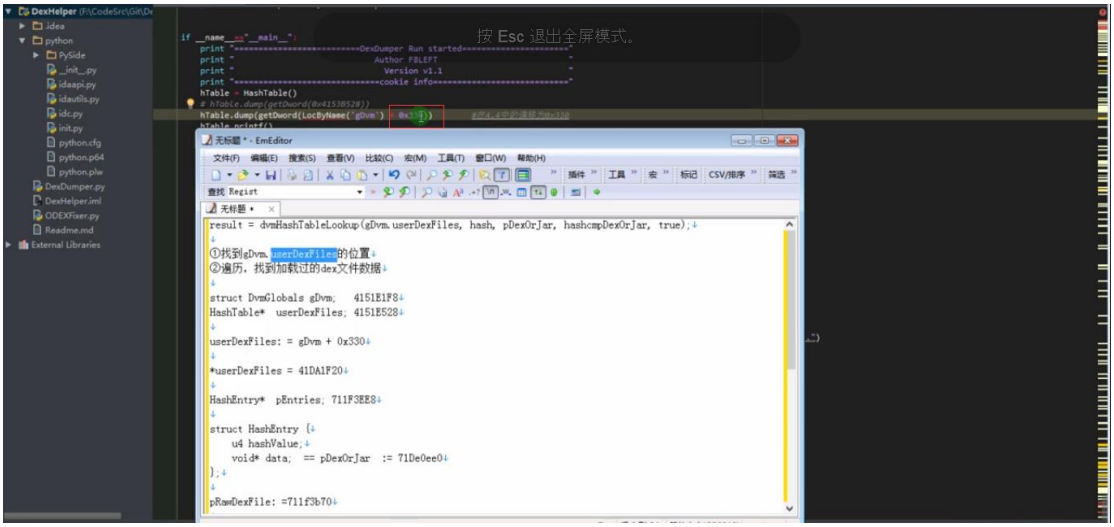


```
        num_class_def = self.dexHeader.classDefsSize
        total_point = self.dexHeader.dataOff + self.dexHeader.dataSize
        start = self.dexHeader.dataOff
        end = total_point
        while total_point%3:       #align 4
            total_point += 1
        print "num class def:", num_class_def
        logger.write("num class def %d\n" % num_class_def)
        for i in range(num_class_def):
            print "cur class:", i, "Total:", num_class_def
            logger.write("cur class: %d Total: %d\n" % (i, num_class_def))
            classdef = DexClassDef()
            classdef.dump(self.pClassDefs+32*i)
            descriptor = dexGetClassDescriptor(self, classdef)
            need_extra = False
            need_pass = False
            tmp = slashtodot(descriptor)
            # if descriptor.startswith("Landroid") or classdef.classDataOff == 0:       #skip class Landroid... anyway it can't be used...
            #     need_pass = True
            #     print "des:", tmp, "is passed"
            if classdef.classDataOff == 0:          #skip class Landroid...
                need_pass = True
                print "des:", tmp, "is passed"
                logger.write("dex %s is passed\n" % tmp)
            else:
                print "des", tmp
                logger.write("dex %s\n" %tmp)

                if classdef.classDataOff < start or classdef.classDataOff > end:
                    need_extra = True
                classdata = ClassdataItem()
                classdata.dump(int(self.baseAddr+classdef.classDataOff))
                if classdata.direct_methods_size:
                    for j in range(classdata.direct_methods_size):
                        method = classdata.direct_methods[j]
                        if method.code_off == 0:
                            continue
                        if method.access_flags & 0x100:     # native func or ..
                            need_extra = True
                            method.code_off = 0
                            continue
                        if method.code_off < start or method.code_off > end:
                            need_extra = True
                            codeItem = CodeItem()
                            codeItem.dump(int(self.baseAddr+method.code_off))
```

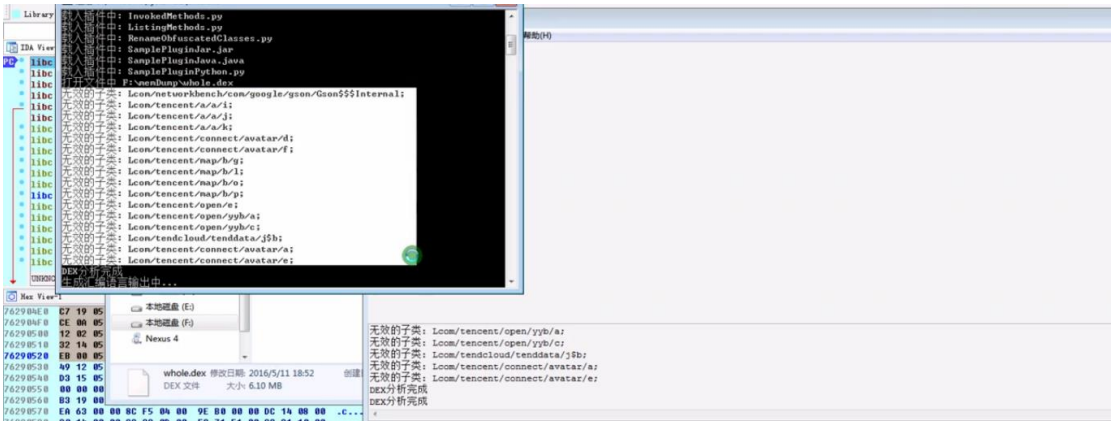


```
if __name__ == "__main__":
    print "=========================DexDumper Run started======================="
    print "                              Author F8LEFT                          "
    print "                                Version v1.1                         "
    print "================================cookie info=============================="
    hTable = HashTable()
    # hTable.dump(getDword(0x41538528))
    hTable.dump(getDword(LocByName('gDvm') + 0x330))        #在4.4里的偏移为0x330
    hTable.printf()
    print "================================cookie end==============================="

    # print "%X %X" % hTable.getdexfileaddr(2)
    iSel = AskLong(0, 'please select a cookie to dump:')
    if isinstance(iSel, int):

        pDexFile, pMemMap = hTable.getdexfileaddr(iSel)          #DvmDex* pDvmDex value (cookies value)
        print "================================dump start==============================="
        print "================================dump DexFile============================="
        dump_dir = "F:/apkDump/"
        dexfile = DexFile()
        dexfile.dump(pDexFile)
        dexfile.dexHeader.printf()
        dexfile.copytofile(dump_dir)
        print "================================dump DexFile finished====================="

        if dexfile.pOptHeader != pMemMap and dexfile.pHeader != pMemMap:
            print "================================dump MemMapping==========================="
            isDumpMem = AskYN(0, "Extra message has been found, do you want to dump the memMapping? This may be help in some case.")
            dump_mem = "F:/memDump/"
            if pMemMap != 0 and isDumpMem == 1:
                dexMem = DexFile()
                dexMem.dumpheader(pMemMap)
                dexMem.dexHeader.printf()
                dexMem.copytofile(dump_mem)
            print "================================dump MemMapping finished=================="
        print "================================dump end=================================="
    print "=========================DexDumper Run finished======================"
```

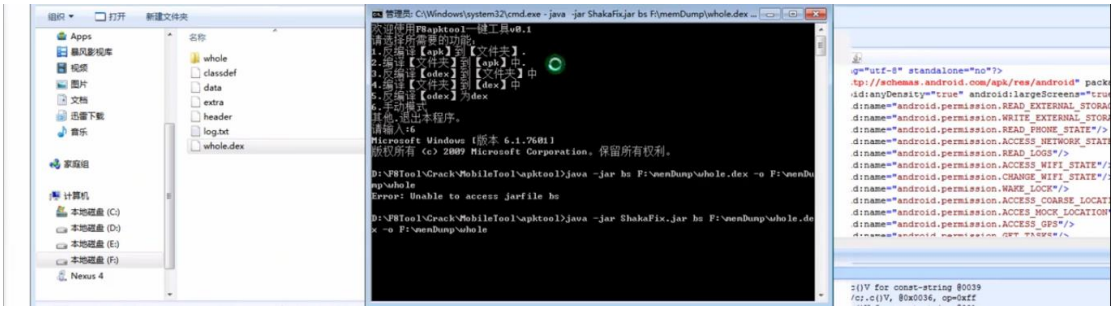
注意要修改 gDvm 地址数据



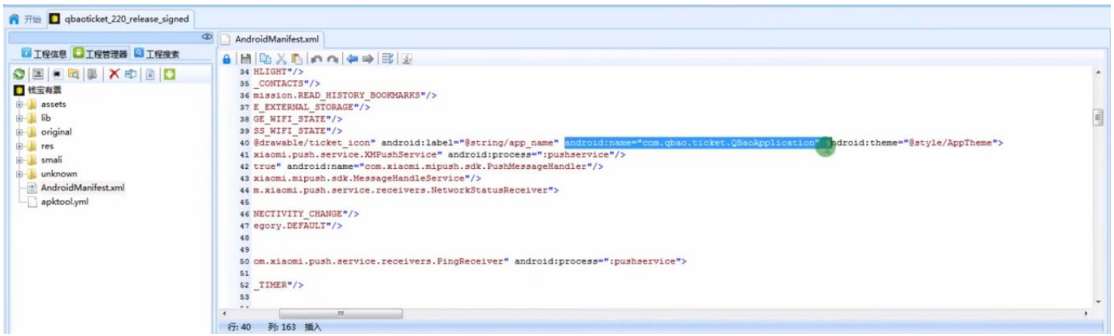
将原始 dex 文件 dump 下来后使用 jeb 打开



先还原为 smali 代码然后替换原项目中的 smali 文件



然后删除 AndroidManifest.xml 中的 android:name 字段



重新编译安装，运行后异常退出，原因是没有找到原始的 Application，解决方法：将

android:name 字段保留