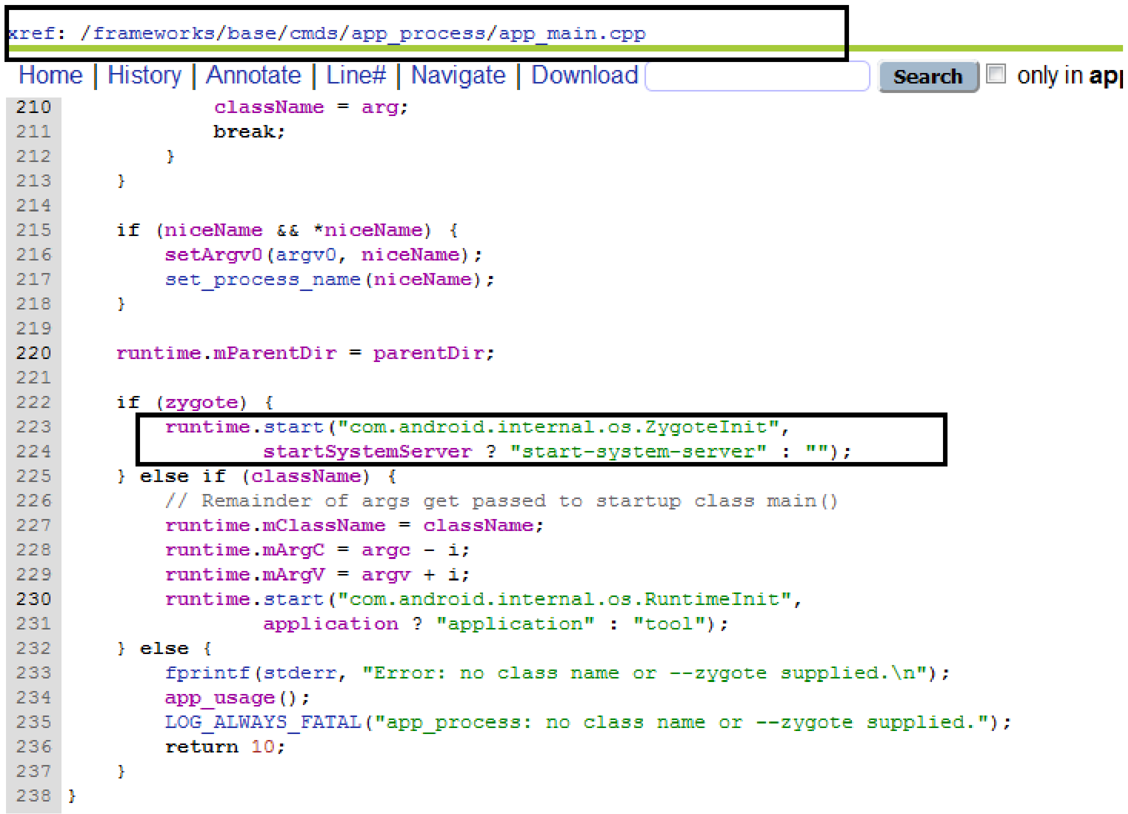
从Dalvik初始化到优化dex函数（源代码查找）

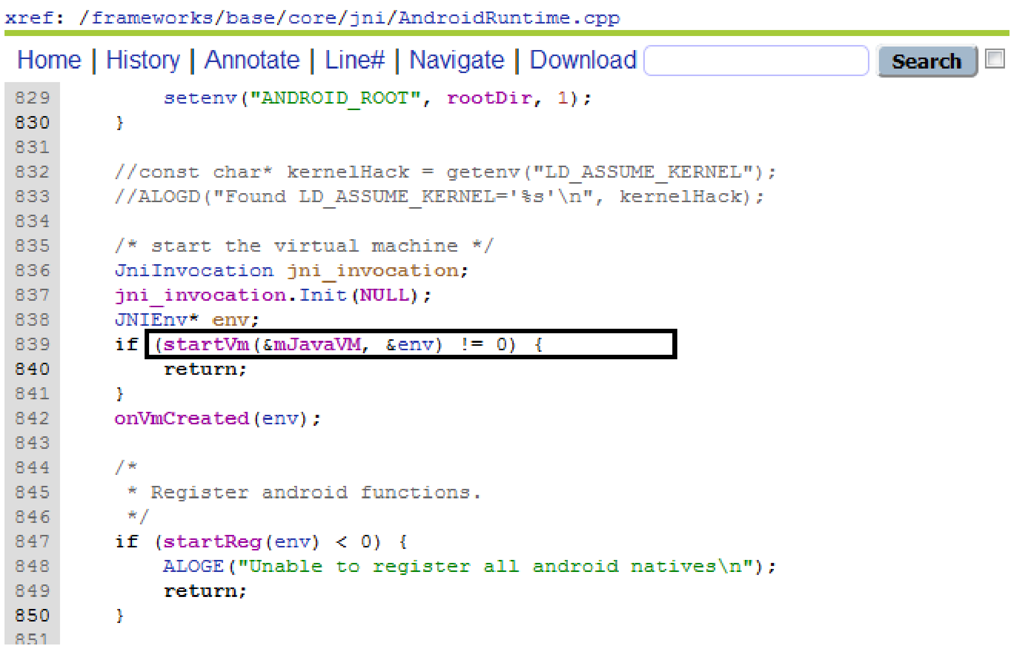
//关于dex加壳，他们说可以在优化dex函数前下断，可我不是很了解为什么，那么就开始接触源代码吧
//经过第一次糊里糊涂的摸索，这一次应该会顺利很多
//

//首先是入口点



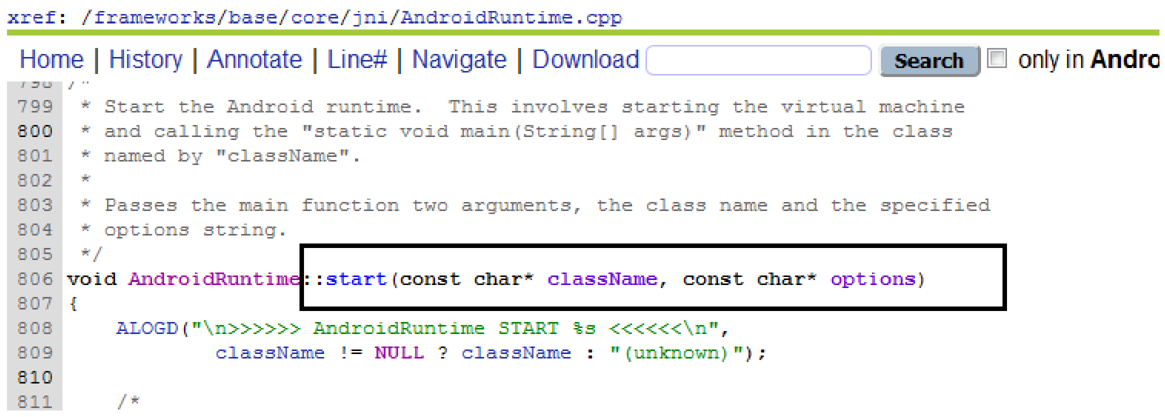
xref: /frameworks/base/cmds/app_process/app_main.cpp

```
            className = arg;
            break;
        }
    }

    if (niceName && *niceName) {
        setArgv0(argv0, niceName);
        set_process_name(niceName);
    }

    runtime.mParentDir = parentDir;

    if (zygote) {
        runtime.start("com.android.internal.os.ZygoteInit",
                startSystemServer ? "start-system-server" : "");
    } else if (className) {
        // Remainder of args get passed to startup class main()
        runtime.mClassName = className;
        runtime.mArgC = argc - i;
        runtime.mArgV = argv + i;
        runtime.start("com.android.internal.os.RuntimeInit",
                application ? "application" : "tool");
    } else {
        fprintf(stderr, "Error: no class name or --zygote supplied.\n");
        app_usage();
        LOG_ALWAYS_FATAL("app_process: no class name or --zygote supplied.");
        return 10;
    }
}
```



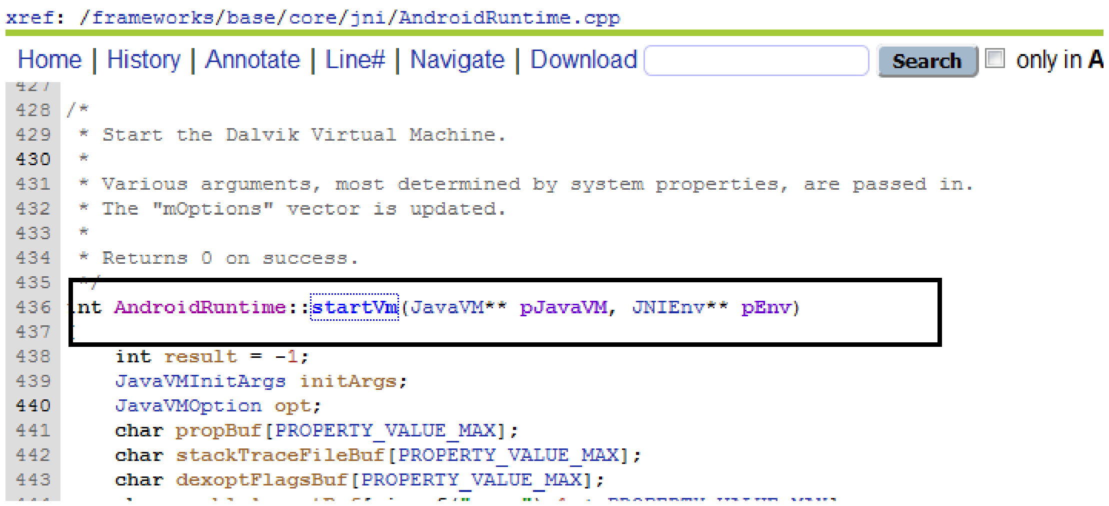
xref: /frameworks/base/core/jni/AndroidRuntime.cpp

```
        setenv("ANDROID_ROOT", rootDir, 1);
    }

    //const char* kernelHack = getenv("LD_ASSUME_KERNEL");
    //ALOGD("Found LD_ASSUME_KERNEL='%s'\n", kernelHack);

    /* start the virtual machine */
    JniInvocation jni_invocation;
    jni_invocation.Init(NULL);
    JNIEnv* env;
    if (startVm(&mJavaVM, &env) != 0) {
        return;
    }
    onVmCreated(env);

    /*
     * Register android functions.
     */
    if (startReg(env) < 0) {
        ALOGE("Unable to register all android natives\n");
        return;
    }
```



xref: /frameworks/base/core/jni/AndroidRuntime.cpp

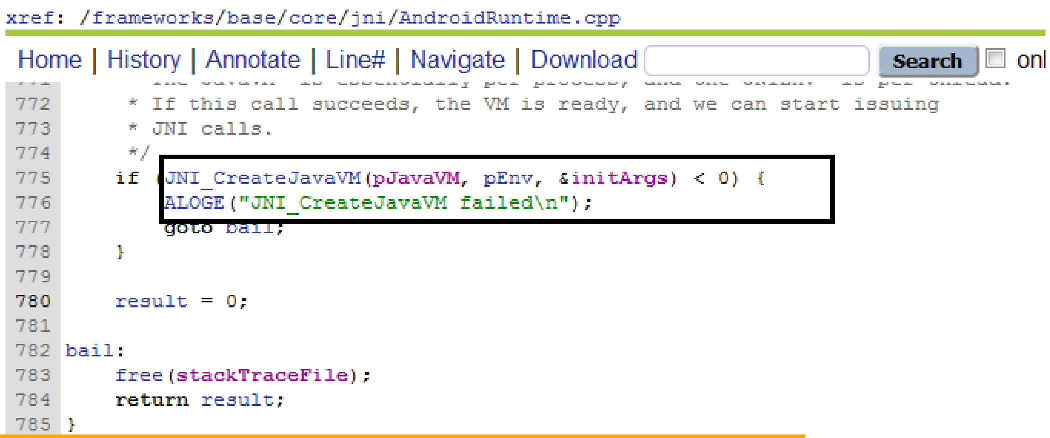
```
/*
 * Start the Android runtime.  This involves starting the virtual machine
 * and calling the "static void main(String[] args)" method in the class
 * named by "className".
 *
 * Passes the main function two arguments, the class name and the specified
 * options string.
 */
void AndroidRuntime::start(const char* className, const char* options)
{
    ALOGD("\n>>>>>> AndroidRuntime START %s <<<<<<\n",
            className != NULL ? className : "(unknown)");

    /*
```



xref: /frameworks/base/core/jni/AndroidRuntime.cpp

```
/*
 * Start the Dalvik Virtual Machine.
 *
 * Various arguments, most determined by system properties, are passed in.
 * The "mOptions" vector is updated.
 *
 * Returns 0 on success.
 */
int AndroidRuntime::startVm(JavaVM** pJavaVM, JNIEnv** pEnv)
{
    int result = -1;
    JavaVMInitArgs initArgs;
    JavaVMOption opt;
    char propBuf[PROPERTY_VALUE_MAX];
    char stackTraceFileBuf[PROPERTY_VALUE_MAX];
    char dexoptFlagsBuf[PROPERTY_VALUE_MAX];
```

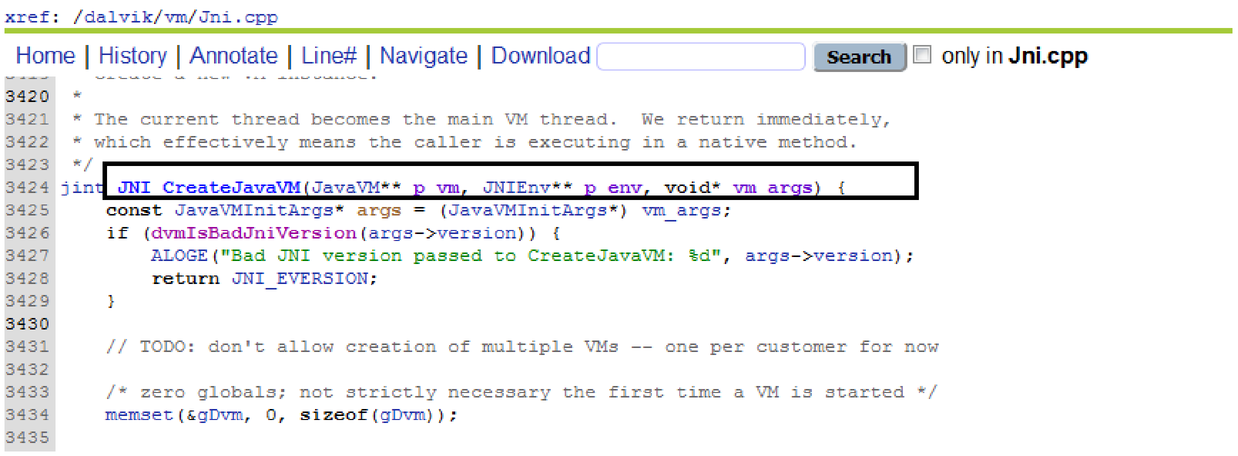
//创建虚拟机



xref: /frameworks/base/core/jni/AndroidRuntime.cpp

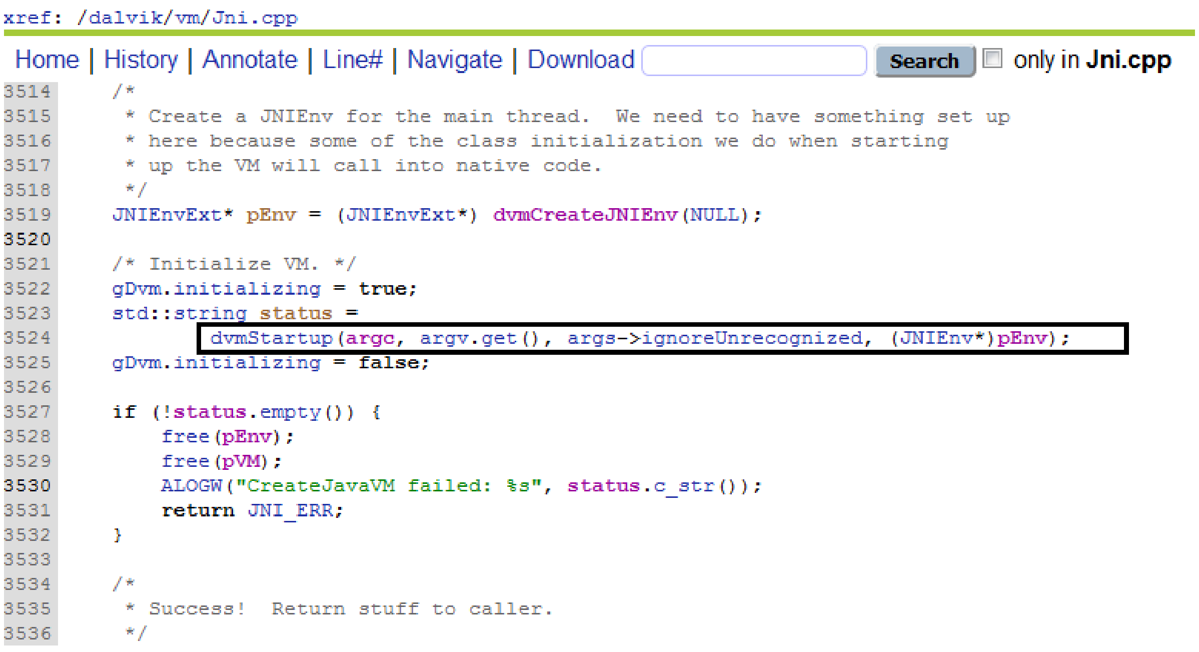
```
     * If this call succeeds, the VM is ready, and we can start issuing
     * JNI calls.
     */
    if (JNI_CreateJavaVM(pJavaVM, pEnv, &initArgs) < 0) {
        ALOGE("JNI_CreateJavaVM failed\n");
        goto bail;
    }

    result = 0;

bail:
    free(stackTraceFile);
    return result;
}
```

//我的理解是JavaVM虚拟机实例指针，整个机子只有一个。而JNIEnv则对应每个进程凑合着理解吧！以后再纠正。0.0



xref: /dalvik/vm/Jni.cpp

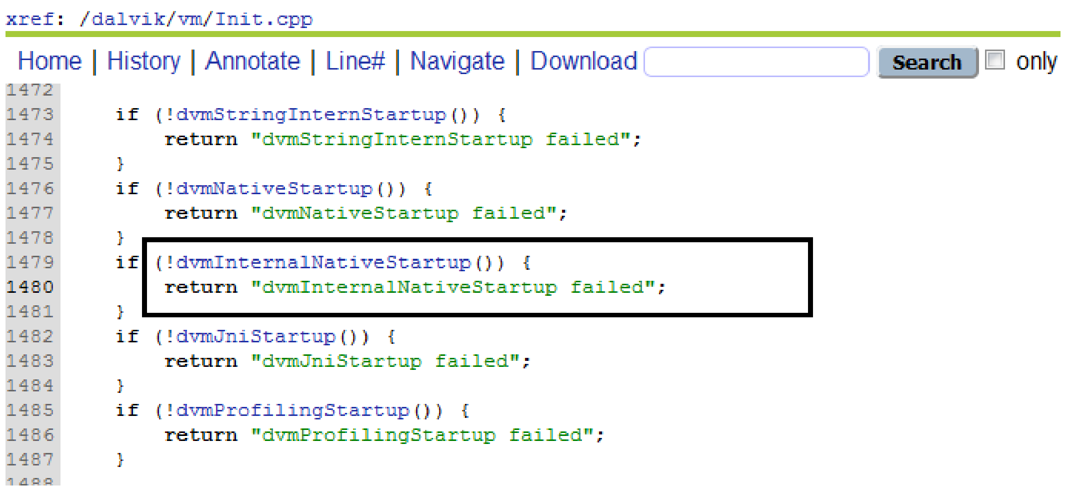
```
 *
 * The current thread becomes the main VM thread.  We return immediately,
 * which effectively means the caller is executing in a native method.
 */
jint JNI_CreateJavaVM(JavaVM** p_vm, JNIEnv** p_env, void* vm_args) {
    const JavaVMInitArgs* args = (JavaVMInitArgs*) vm_args;
    if (dvmIsBadJniVersion(args->version)) {
        ALOGE("Bad JNI version passed to CreateJavaVM: %d", args->version);
        return JNI_EVERSION;
    }

    // TODO: don't allow creation of multiple VMs -- one per customer for now

    /* zero globals; not strictly necessary the first time a VM is started */
    memset(&gDvm, 0, sizeof(gDvm));
```



xref: /dalvik/vm/Jni.cpp

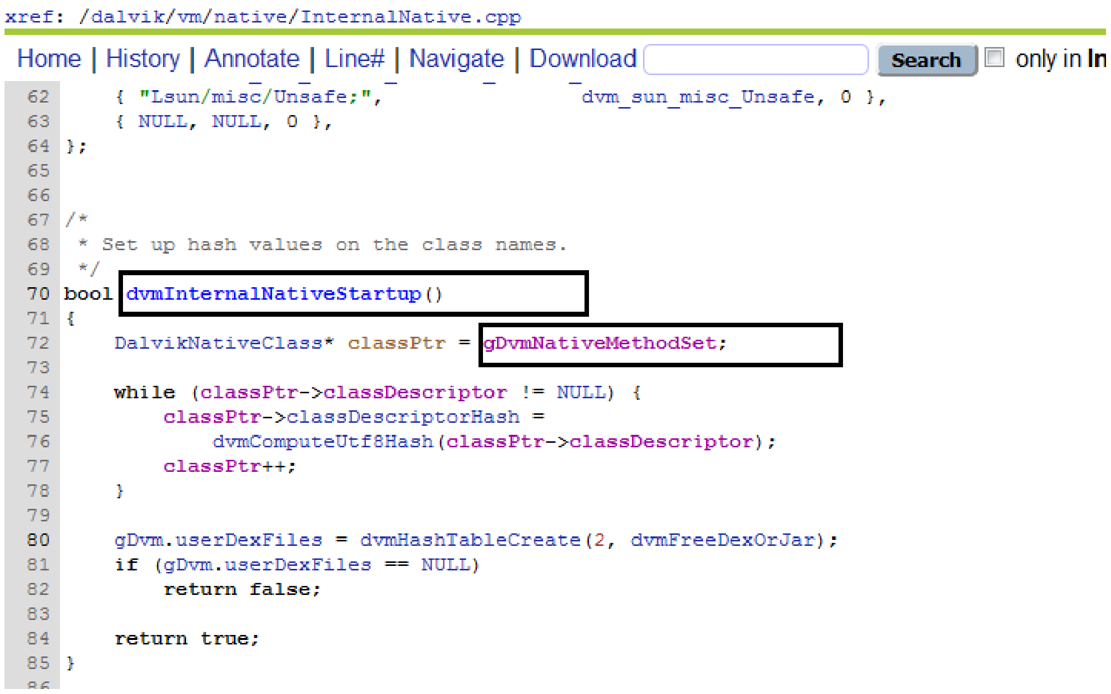
```
    /*
     * Create a JNIEnv for the main thread.  We need to have something set up
     * here because some of the class initialization we do when starting
     * up the VM will call into native code.
     */
    JNIEnvExt* pEnv = (JNIEnvExt*) dvmCreateJNIEnv(NULL);

    /* Initialize VM. */
    gDvm.initializing = true;
    std::string status =
            dvmStartup(argc, argv.get(), args->ignoreUnrecognized, (JNIEnv*)pEnv);
    gDvm.initializing = false;

    if (!status.empty()) {
        free(pEnv);
        free(pVM);
        ALOGW("CreateJavaVM failed: %s", status.c_str());
        return JNI_ERR;
    }

    /*
     * Success!  Return stuff to caller.
     */
```

//内核动态注册的本地方法要启动了



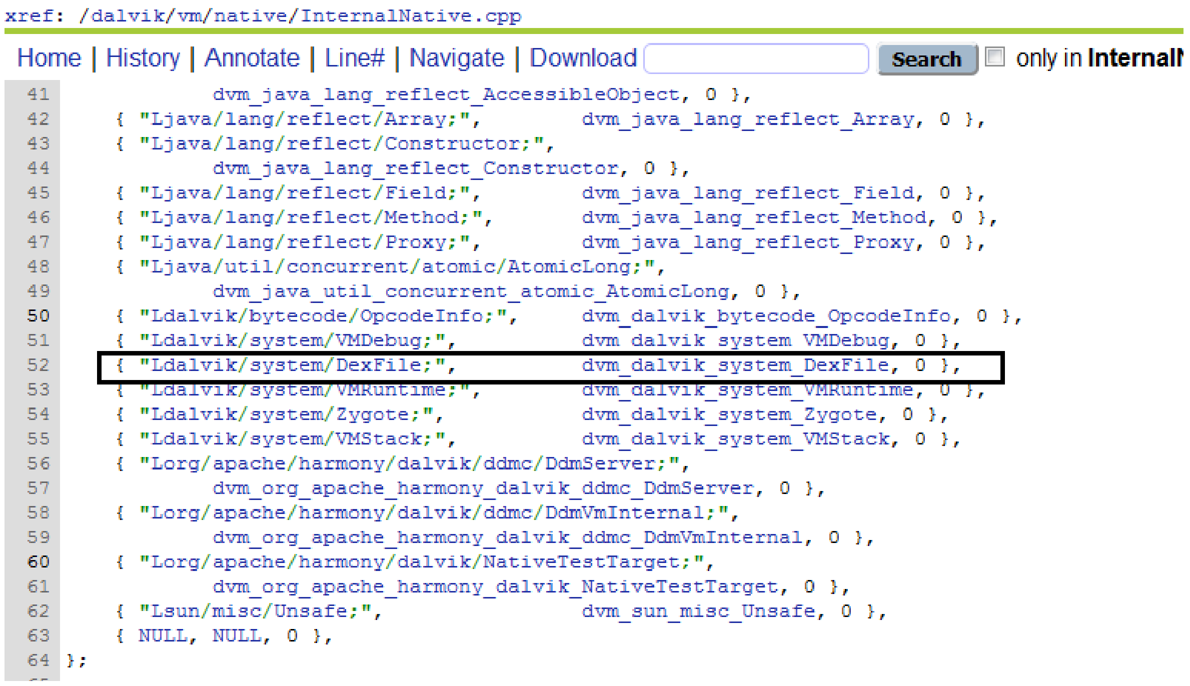
xref: /dalvik/vm/Init.cpp

```

    if (!dvmStringInternStartup()) {
        return "dvmStringInternStartup failed";
    }
    if (!dvmNativeStartup()) {
        return "dvmNativeStartup failed";
    }
    if (!dvmInternalNativeStartup()) {
        return "dvmInternalNativeStartup failed";
    }
    if (!dvmJniStartup()) {
        return "dvmJniStartup failed";
    }
    if (!dvmProfilingStartup()) {
        return "dvmProfilingStartup failed";
    }
```



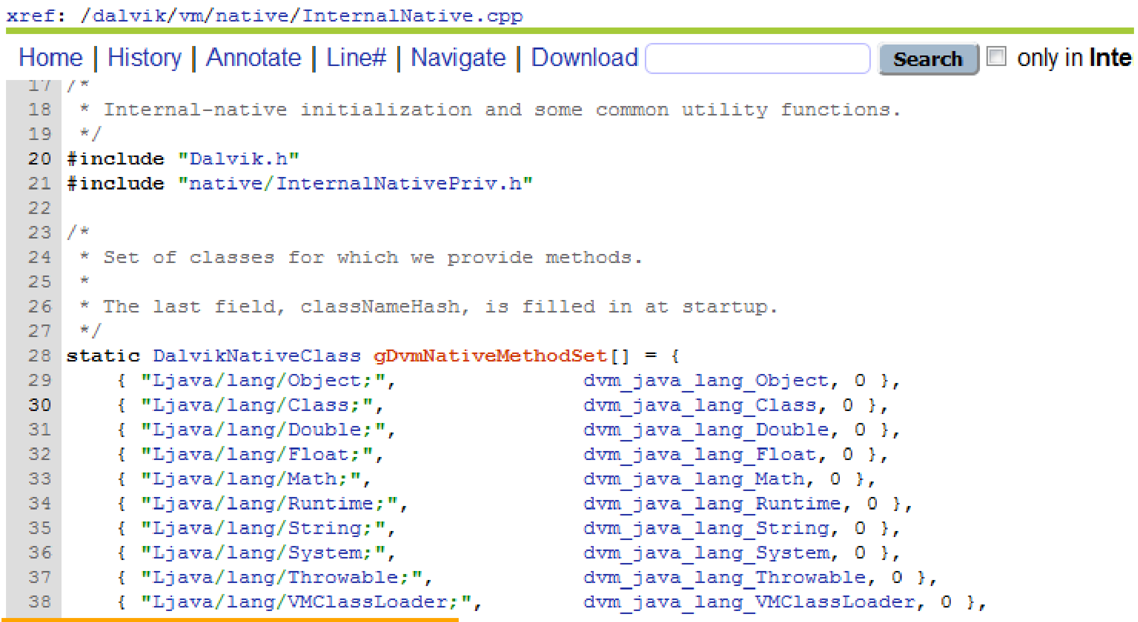
xref: /dalvik/vm/native/InternalNative.cpp

```
    { "Lsun/misc/Unsafe;",              dvm_sun_misc_Unsafe, 0 },
    { NULL, NULL, 0 },
};


/*
 * Set up hash values on the class names.
 */
bool dvmInternalNativeStartup()
{
    DalvikNativeClass* classPtr = gDvmNativeMethodSet;

    while (classPtr->classDescriptor != NULL) {
        classPtr->classDescriptorHash =
            dvmComputeUtf8Hash(classPtr->classDescriptor);
        classPtr++;
    }

    gDvm.userDexFiles = dvmHashTableCreate(2, dvmFreeDexOrJar);
    if (gDvm.userDexFiles == NULL)
        return false;

    return true;
}
```



xref: /dalvik/vm/native/InternalNative.cpp

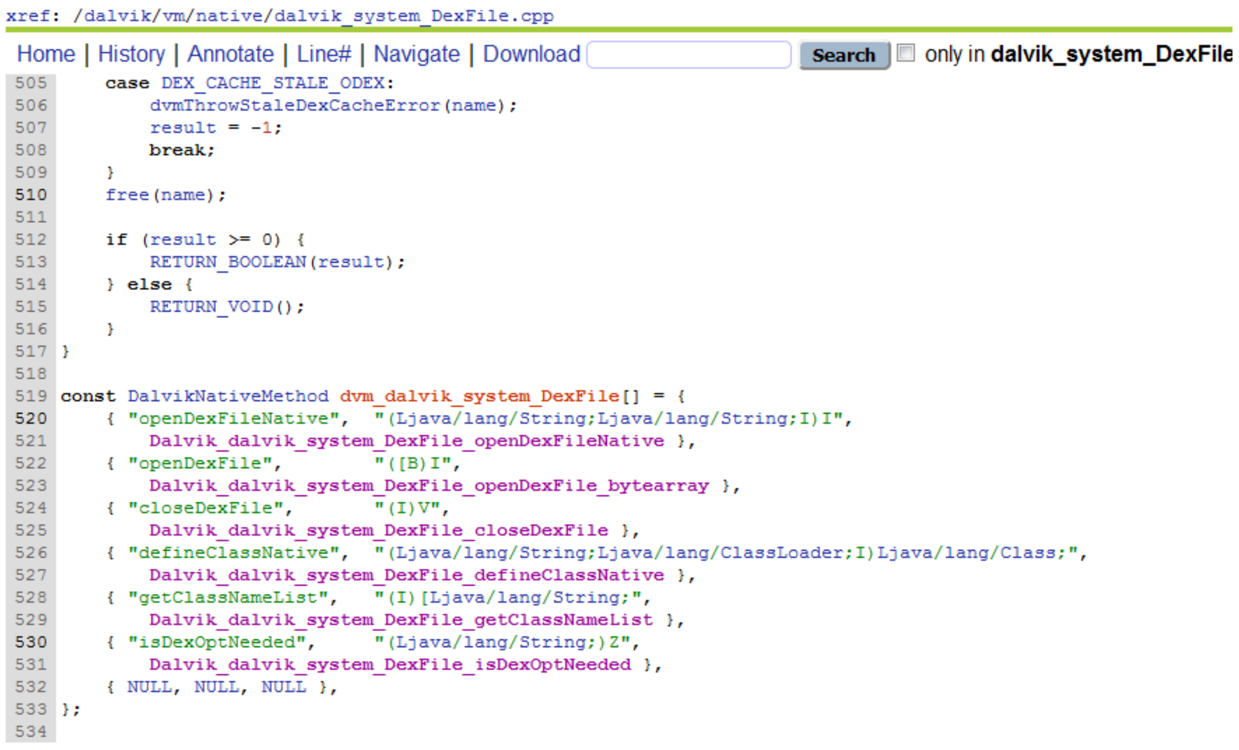
```
            dvm_java_lang_reflect_AccessibleObject, 0 },
    { "Ljava/lang/reflect/Array;",          dvm_java_lang_reflect_Array, 0 },
    { "Ljava/lang/reflect/Constructor;",
            dvm_java_lang_reflect_Constructor, 0 },
    { "Ljava/lang/reflect/Field;",          dvm_java_lang_reflect_Field, 0 },
    { "Ljava/lang/reflect/Method;",         dvm_java_lang_reflect_Method, 0 },
    { "Ljava/lang/reflect/Proxy;",          dvm_java_lang_reflect_Proxy, 0 },
    { "Ljava/util/concurrent/atomic/AtomicLong;",
            dvm_java_util_concurrent_atomic_AtomicLong, 0 },
    { "Ldalvik/bytecode/OpcodeInfo;",       dvm_dalvik_bytecode_OpcodeInfo, 0 },
    { "Ldalvik/system/VMDebug;",            dvm_dalvik_system_VMDebug, 0 },
    { "Ldalvik/system/DexFile;",            dvm_dalvik_system_DexFile, 0 },
    { "Ldalvik/system/VMRuntime;",          dvm_dalvik_system_VMRuntime, 0 },
    { "Ldalvik/system/Zygote;",             dvm_dalvik_system_Zygote, 0 },
    { "Ldalvik/system/VMStack;",            dvm_dalvik_system_VMStack, 0 },
    { "Lorg/apache/harmony/dalvik/ddmc/DdmServer;",
            dvm_org_apache_harmony_dalvik_ddmc_DdmServer, 0 },
    { "Lorg/apache/harmony/dalvik/ddmc/DdmVmInternal;",
            dvm_org_apache_harmony_dalvik_ddmc_DdmVmInternal, 0 },
    { "Lorg/apache/harmony/dalvik/NativeTestTarget;",
            dvm_org_apache_harmony_dalvik_NativeTestTarget, 0 },
    { "Lsun/misc/Unsafe;",                  dvm_sun_misc_Unsafe, 0 },
    { NULL, NULL, 0 },
};
```



xref: /dalvik/vm/native/InternalNative.cpp

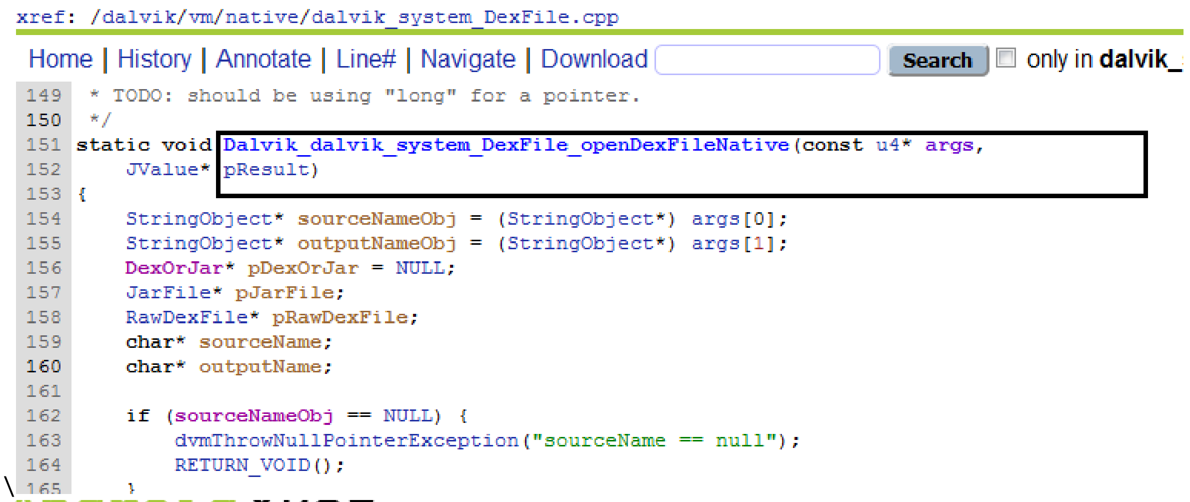
```
/*
 * Internal-native initialization and some common utility functions.
 */
#include "Dalvik.h"
#include "native/InternalNativePriv.h"

/*
 * Set of classes for which we provide methods.
 *
 * The last field, classNameHash, is filled in at startup.
 */
static DalvikNativeClass gDvmNativeMethodSet[] = {
    { "Ljava/lang/Object;",               dvm_java_lang_Object, 0 },
    { "Ljava/lang/Class;",                dvm_java_lang_Class, 0 },
    { "Ljava/lang/Double;",               dvm_java_lang_Double, 0 },
    { "Ljava/lang/Float;",                dvm_java_lang_Float, 0 },
    { "Ljava/lang/Math;",                 dvm_java_lang_Math, 0 },
    { "Ljava/lang/Runtime;",              dvm_java_lang_Runtime, 0 },
    { "Ljava/lang/String;",               dvm_java_lang_String, 0 },
    { "Ljava/lang/System;",               dvm_java_lang_System, 0 },
    { "Ljava/lang/Throwable;",            dvm_java_lang_Throwable, 0 },
    { "Ljava/lang/VMClassLoader;",        dvm_java_lang_VMClassLoader, 0 },
```

//dex相关的函数注册都在这了


Home | History | Annotate | Line# | Navigate | Download
Search
only in dalvik_system_DexFile

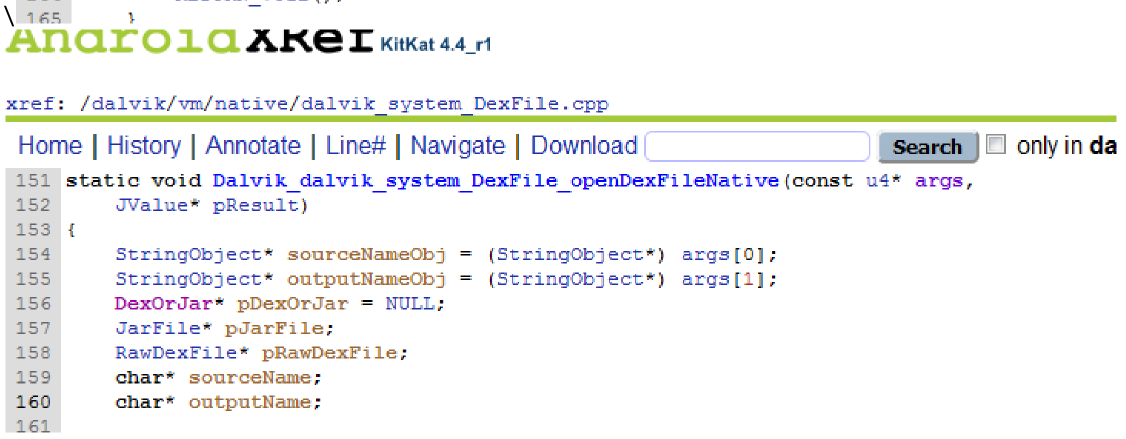
xref: /dalvik/vm/native/dalvik_system_DexFile.cpp

```
    case DEX_CACHE_STALE_ODEX:
        dvmThrowStaleDexCacheError(name);
        result = -1;
        break;
    }
    free(name);

    if (result >= 0) {
        RETURN_BOOLEAN(result);
    } else {
        RETURN_VOID();
    }
}

const DalvikNativeMethod dvm_dalvik_system_DexFile[] = {
    { "openDexFileNative",  "(Ljava/lang/String;Ljava/lang/String;I)I",
        Dalvik_dalvik_system_DexFile_openDexFileNative },
    { "openDexFile",        "([B)I",
        Dalvik_dalvik_system_DexFile_openDexFile_bytearray },
    { "closeDexFile",       "(I)V",
        Dalvik_dalvik_system_DexFile_closeDexFile },
    { "defineClassNative",  "(Ljava/lang/String;Ljava/lang/ClassLoader;I)Ljava/lang/Class;",
        Dalvik_dalvik_system_DexFile_defineClassNative },
    { "getClassNameList",   "(I)[Ljava/lang/String;",
        Dalvik_dalvik_system_DexFile_getClassNameList },
    { "isDexOptNeeded",     "(Ljava/lang/String;)Z",
        Dalvik_dalvik_system_DexFile_isDexOptNeeded },
    { NULL, NULL, NULL },
};

```



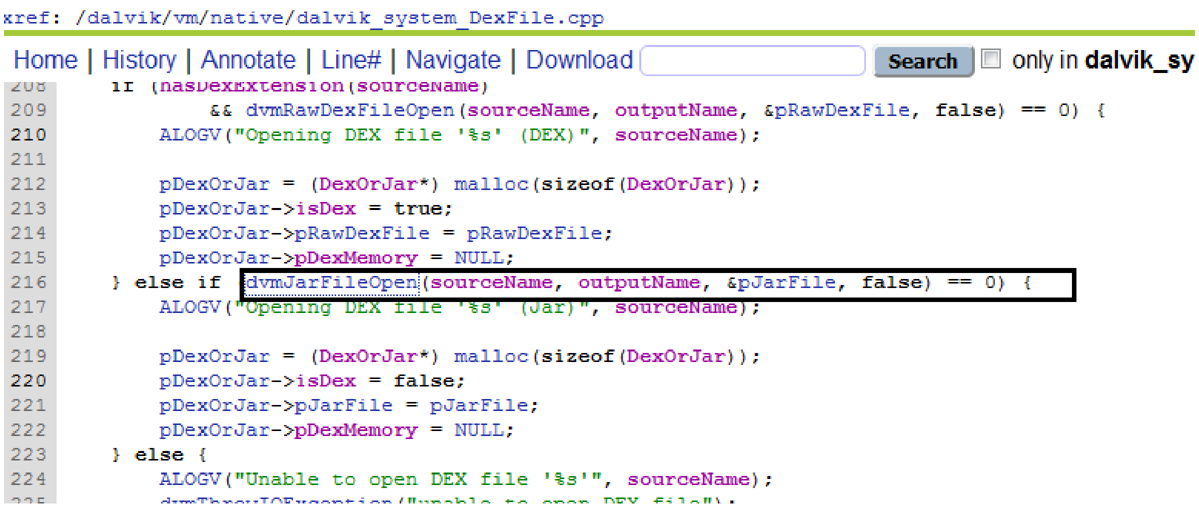
xref: /dalvik/vm/native/dalvik_system_DexFile.cpp

```
 * TODO: should be using "long" for a pointer.
 */
static void Dalvik_dalvik_system_DexFile_openDexFileNative(const u4* args,
    JValue* pResult)
{
    StringObject* sourceNameObj = (StringObject*) args[0];
    StringObject* outputNameObj = (StringObject*) args[1];
    DexOrJar* pDexOrJar = NULL;
    JarFile* pJarFile;
    RawDexFile* pRawDexFile;
    char* sourceName;
    char* outputName;

    if (sourceNameObj == NULL) {
        dvmThrowNullPointerException("sourceName == null");
        RETURN_VOID();
    }
```


AndroidXRef KitKat 4.4_r1
Home | History | Annotate | Line# | Navigate | Download
Search
only in da

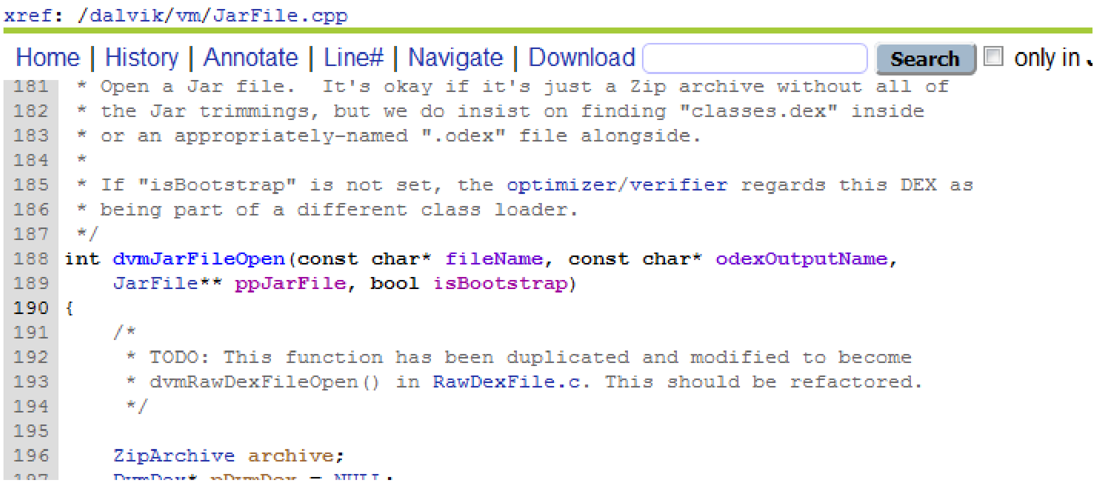
xref: /dalvik/vm/native/dalvik_system_DexFile.cpp

```
static void Dalvik_dalvik_system_DexFile_openDexFileNative(const u4* args,
    JValue* pResult)
{
    StringObject* sourceNameObj = (StringObject*) args[0];
    StringObject* outputNameObj = (StringObject*) args[1];
    DexOrJar* pDexOrJar = NULL;
    JarFile* pJarFile;
    RawDexFile* pRawDexFile;
    char* sourceName;
    char* outputName;

```



xref: /dalvik/vm/native/dalvik_system_DexFile.cpp

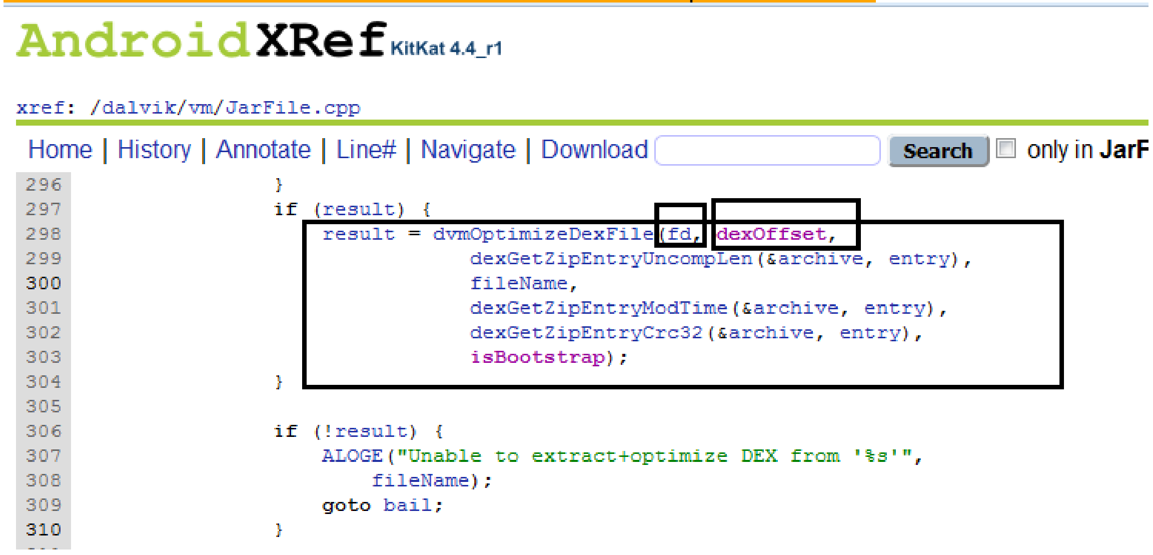
```
    if (hasDexExtension(sourceName)
            && dvmRawDexFileOpen(sourceName, outputName, &pRawDexFile, false) == 0) {
        ALOGV("Opening DEX file '%s' (DEX)", sourceName);

        pDexOrJar = (DexOrJar*) malloc(sizeof(DexOrJar));
        pDexOrJar->isDex = true;
        pDexOrJar->pRawDexFile = pRawDexFile;
        pDexOrJar->pDexMemory = NULL;
    } else if (dvmJarFileOpen(sourceName, outputName, &pJarFile, false) == 0) {
        ALOGV("Opening DEX file '%s' (Jar)", sourceName);

        pDexOrJar = (DexOrJar*) malloc(sizeof(DexOrJar));
        pDexOrJar->isDex = false;
        pDexOrJar->pJarFile = pJarFile;
        pDexOrJar->pDexMemory = NULL;
    } else {
        ALOGV("Unable to open DEX file '%s'", sourceName);
        dvmThrowIOException("Unable to open DEX file");
```



xref: /dalvik/vm/JarFile.cpp

```
 * Open a Jar file.  It's okay if it's just a Zip archive without all of
 * the Jar trimmings, but we do insist on finding "classes.dex" inside
 * or an appropriately-named ".odex" file alongside.
 *
 * If "isBootstrap" is not set, the optimizer/verifier regards this DEX as
 * being part of a different class loader.
 */
int dvmJarFileOpen(const char* fileName, const char* odexOutputName,
    JarFile** ppJarFile, bool isBootstrap)
{
    /*
     * TODO: This function has been duplicated and modified to become
     * dvmRawDexFileOpen() in RawDexFile.c. This should be refactored.
     */

    ZipArchive archive;
    DvmDex* pDvmDex = NULL;
```

//这就是我们要找到的dvmOptimizeDexFile
//第一个参数为dex文件头在内存中的指针，第二个参数为大小。即dump时，对应r0,r1的值


AndroidXRef KitKat 4.4_r1
Home | History | Annotate | Line# | Navigate | Download
Search
only in JarF

xref: /dalvik/vm/JarFile.cpp

```
                }
                if (result) {
                    result = dvmOptimizeDexFile(fd, dexOffset,
                                dexGetZipEntryUncompLen(&archive, entry),
                                fileName,
                                dexGetZipEntryModTime(&archive, entry),
                                dexGetZipEntryCrc32(&archive, entry),
                                isBootstrap);
                }

                if (!result) {
                    ALOGE("Unable to extract+optimize DEX from '%s'",
                        fileName);
                    goto bail;
                }
```

//再附加一张启动优化dex的主函数，有兴趣的进去看看（ ）

xref: /dalvik/dexopt/OptMain.cpp

Home | History | Annotate | Line# | Navigate | Download Search only in **OptMain.c**

```
    return result;
}

/*
 * Main entry point.  Decide where to go.
 */
int main(int argc, char* const argv[])
{
    set_process_name("dexopt");

    setvbuf(stdout, NULL, _IONBF, 0);

    if (argc > 1) {
        if (strcmp(argv[1], "--zip") == 0)
            return fromZip(argc, argv);
        else if (strcmp(argv[1], "--dex") == 0)
            return fromDex(argc, argv);
        else if (strcmp(argv[1], "--preopt") == 0)
            return preopt(argc, argv);
    }

    fprintf(stderr,
        "Usage:\n\n"
        "Short version: Don't use this.\n\n"
        "Slightly longer version: This system-internal tool is used to\n"
        "produce optimized dex files. See the source code for details.\n");

    return 1;
}
```