Android 启动 app

1. Android 启动应用过程

see：https://blog.csdn.net/luoshengyang/article/details/8885792

操作：手机屏幕点击图标，即可启动该图标应用，该应用是有系统 Luncher 启动，完成运行 app，需要 Application 的加载，MainActivity 界面的加载。

1) Application 类的加载

使用 as 设置 debug，运行一个应用，再自定义的 application 中设置断点，如图

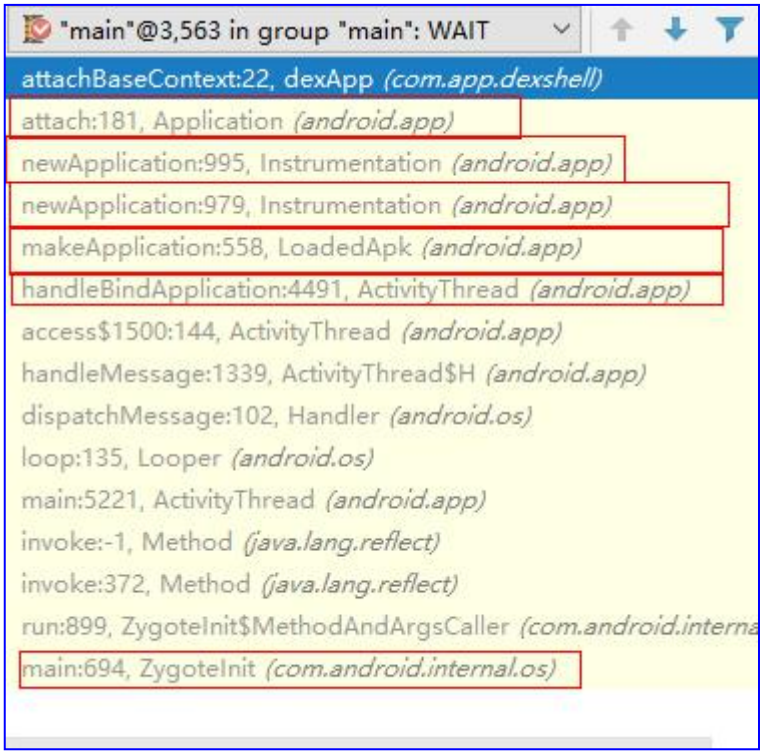


图 1

Zygote 通过 fork 一个进程，完成初始化安卓虚拟机。

在类 ActivityThread 中的 main 函数全局类 ActivityThread 初始化

```
ActivityThread:
public static void main(String[] args) {
//...
ActivityThread thread = new ActivityThread();
thread.attach(false);
//....}
```

ActivityThread: handleBindApplication 用于加载 Application

ActivityThread: performLaunchActivity 用于加载 Activity

ActivityThread: handleBindApplication

```
//参数 AppBindData data 为解析 AndroidManifest.xml 后的数据
private void handleBindApplication(AppBindData data) {
//appContext 的创建
final ContextImpl appContext = ContextImpl.createAppContext(this,
data.info);
... ...
//android 应用调试的端口设置为 8100
Debug.changeDebugPort(8100);
... ...

try {
    // If the app is being launched for full backup or restore, bring it
up in
    // a restricted environment with the base application class.
    Application app =
data.info.makeApplication(data.restrictedBackupMode, null);
    mInitialApplication = app;
```

```
    }catch (Exception e) {}
    try {
    //callApplicationOnCreate 即开发者常使用的 OnCreate()函数
    mInstrumentation.callApplicationOnCreate(app);
} catch (Exception e) {}
    ... ...
    }
    类 android.app.LoadedApk:
    public Application makeApplication(boolean forceDefaultAppClass,
        Instrumentation instrumentation) {
    //如果 mApplication==null，进入下面创建开发者自定义的
userApplication

    if (mApplication != null) {
        return mApplication;
}
Application app = null;
//AndroidManifest.xml 中开发者定义的 application name=”xxx.xx”
String appClass = mApplicationInfo.className;
if (forceDefaultAppClass || (appClass == null)) {
    appClass = "android.app.Application";
}

try {
    java.lang.ClassLoader cl = getClassLoader();
    if (!mPackageName.equals("android")) {
        initializeJavaContextClassLoader();
    }
    ContextImpl appContext =
```

```
ContextImpl.createAppContext(mActivityThread, this);
    app = mActivityThread.mInstrumentation.newApplication(
            cl, appClass, appContext);
    appContext.setOuterContext(app);
} catch (Exception e) {
    if (!mActivityThread.mInstrumentation.onException(app, e)) {
        throw new RuntimeException(
            "Unable to instantiate application " + appClass
            + ": " + e.toString(), e);
    }
}
mActivityThread.mAllApplications.add(app);
mApplication = app;
if (instrumentation != null) {
    try {
        instrumentation.callApplicationOnCreate(app);
    } catch (Exception e) {
        if (!instrumentation.onException(app, e)) {
            throw new RuntimeException(
                "Unable to create application " +
app.getClass().getName()
                    + ": " + e.toString(), e);
        }
    }
}
...
return app;
    }
```

类 android.app.Instrumentation

```
    public Application newApplication(ClassLoader cl, String className,
Context context)
            throws InstantiationException, IllegalAccessException,
            ClassNotFoundException {
    return newApplication(cl.loadClass(className), context);
}
    static public Application newApplication(Class<?> clazz, Context
context)
            throws InstantiationException, IllegalAccessException,
            ClassNotFoundException {
    Application app = (Application)clazz.newInstance();
//进入到 android.app.Application.attachBaseContext(Context context)
    该函数经常用于加固中
       app.attach(context);
    return app;
}
    类 android.app.Application
    /* package */ final void attach(Context context) {
    attachBaseContext(context);
    mLoadedApk = ContextImpl.getImpl(context).mPackageInfo;
}
```

2) MainActivity 加载

ActivityThread: performLaunchActivity

```
    private Activity performLaunchActivity(ActivityClientRecord r,
Intent customIntent) {
    ActivityInfo aInfo = r.activityInfo;//即启动了 MainActivity
```

```
    ComponentName component = r.intent.getComponent();
    if (component == null) {
        component = r.intent.resolveActivity(
            mInitialApplication.getPackageManager());
        r.intent.setComponent(component);
    }

    if (r.activityInfo.targetActivity != null) {
        component = new ComponentName(r.activityInfo.packageName,
                r.activityInfo.targetActivity);
    }

    Activity activity = null;
    try {
        java.lang.ClassLoader cl = r.packageInfo.getClassLoader();
        activity = mInstrumentation.newActivity(
                cl, component.getClassName(), r.intent);
                    r.state.setClassLoader(cl);
         } catch (Exception e) {  }

    try {
        Application app = r.packageInfo.makeApplication(false,
mInstrumentation);

                      activity.attach(appContext, this,
getInstrumentation(), r.token,
                    r.ident, app, r.intent, r.activityInfo, title,
r.parent,
                    r.embeddedID, r.lastNonConfigurationInstances,
```

```
config,
                    r.referrer, r.voiceInteractor);

            if (customIntent != null) {
                activity.mIntent = customIntent;
            }
            if (r.isPersistable()) {
                mInstrumentation.callActivityOnCreate(activity,
r.state, r.persistentState);
            } else {
                mInstrumentation.callActivityOnCreate(activity,
r.state);
            }

    }
```

2. DexClassLoader 加载 dex

应用开发中可以动态加载 jar 或 dex 文件，调用其中的类和函数。

```
    DexClassLoader loader=new DexClassLoader (
    "/sdcard/aaa.dex", getApplication ().getCacheDir
().getAbsolutePath (),null,   getClassLoader ());
try {
    Class aClass =loader.loadClass
("example.demo.ndkdemo.MainActivity");
    Log.i ("tag:loadclass",aClass.getSimpleName ());
}catch (Exception ex){
    ex.getLocalizedMessage ();
}
```

DexCLassLoader 的定义：

```
public class BaseDexClassLoader extends ClassLoader {
    private final DexPathList pathList;

    /**
     * Constructs an instance.
     *
     * @param dexPath the list of jar/apk files containing classes and
     * resources, delimited by {@code File.pathSeparator}, which
     * defaults to {@code ":"} on Android
     * @param optimizedDirectory directory where optimized dex files
     * should be written; may be {@code null}
     * @param libraryPath the list of directories containing native
     * libraries, delimited by {@code File.pathSeparator}; may be
     * {@code null}
     * @param parent the parent class loader
     */
    public BaseDexClassLoader(String dexPath, File optimizedDirectory,
            String libraryPath, ClassLoader parent) {
        super(parent);
        this.pathList = new DexPathList(this, dexPath, libraryPath, optimizedDirectory
    }
```

DexPathList 定义：

```
/*package*/ final class DexPathList {
    private static final String DEX_SUFFIX = ".dex";
    private static final String JAR_SUFFIX = ".jar";
    private static final String ZIP_SUFFIX = ".zip";
    private static final String APK_SUFFIX = ".apk";

    /** class definition context */
    private final ClassLoader definingContext;

    /**
     * List of dex/resource (class path) elements.
     * Should be called pathElements, but the Facebook app uses reflection
     * to modify 'dexElements' (http://b/7726934).
     */
    private final Element[] dexElements;

    /** List of native library directories. */
    private final File[] nativeLibraryDirectories;

    /**
```

其中可以看出 DexClassLoader 可以加载的文件类型有：dex;jar;zip;apk

makeDexElements 函数：

```
private static Element[] makeDexElements(ArrayList<File> files, File optimizedDirec
                                         ArrayList<IOException> suppressedExceptions) {
    ArrayList<Element> elements = new ArrayList<Element>();
    /*
     * Open all files and load the (direct or contained) dex files
     * up front.
     */
    for (File file : files) {
        File zip = null;
        DexFile dex = null;
        String name = file.getName();

        if (name.endsWith(DEX_SUFFIX)) {
            // Raw dex file (not inside a zip/jar).
            try {
                dex = loadDexFile(file, optimizedDirectory);
            } catch (IOException ex) {
                System.logE("Unable to load dex file: " + file, ex);
            }
        } else if (name.endsWith(APK_SUFFIX) || name.endsWith(JAR_SUFFIX)
                || name.endsWith(ZIP_SUFFIX)) {
            zip = file;

            try {
                dex = loadDexFile(file, optimizedDirectory);
            } catch (IOException suppressed) {
                /*
                 * IOException might get thrown "legitimately" by the DexFile construct
                 * zip file turns out to be resource-only (that is, no classes.dex file
```

函数 loadDexFile()继续分析到函数

```
private DexFile(String sourceName, String outputName, int flags) throws IOException {
    if (outputName != null) {
        try {
            String parent = new File(outputName).getParent();
            if (Libcore.os.getuid() != Libcore.os.stat(parent).st_uid) {
                throw new IllegalArgumentException("Optimized data directory " + parent
                        + " is not owned by the current user. Shared storage cannot protec
                        + " your application from code injection attacks.");
            }
        } catch (ErrnoException ignored) {
            // assume we'll fail with a more contextual error later
        }
    }

    mCookie = openDexFile(sourceName, outputName, flags);
    mFileName = sourceName;
    guard.open("close");
    //System.out.println("DEX FILE cookie is " + mCookie);
}
```

```
private static int openDexFile(String sourceName, String outputName,
    int flags) throws IOException {
    return openDexFileNative(new File(sourceName).getCanonicalPath(),
                             (outputName == null) ? null : new File(outputName).getCano
                             flags);
}

native private static int openDexFileNative(String sourceName, String outputName,
    int flags) throws IOException;
```

openDexFileNative 函数为 native 函数。由此，dalvik 和 art 模式下加载 dex，实现了不同的方式，dalvik 实现解释器，art 实现编译为 oat 文件，两种加载的替换参见

Android ART 运行时无缝替换 Dalvik 虚拟机的过程分析

https://blog.csdn.net/luoshengyang/article/details/18006645

1. Dalvik 加载

4.4 源代码分析：

Dalvik 加载.dex 文件：

```
    /*
     * Try to open it directly as a DEX if the name ends with ".dex".
     * If that fails (or isn't tried in the first place), try it as a
     * Zip with a "classes.dex" inside.
     */
    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;
```

函数调用 dvmRawDexFileOpen -> dexFileParse

```
DexFile* dexFileParse(const u1* data, size_t length, int flags)
{
    DexFile* pDexFile = NULL;
    const DexHeader* pHeader;
    const u1* magic;
    int result = -1;

    if (length < sizeof(DexHeader)) {
        ALOGE("too short to be a valid .dex");
        goto bail;      /* bad file format */
    }

    pDexFile = (DexFile*) malloc(sizeof(DexFile));
    if (pDexFile == NULL)
        goto bail;      /* alloc failure */
    memset(pDexFile, 0, sizeof(DexFile));

    /*
     * Peel off the optimized header.
     */
    if (memcmp(data, DEX_OPT_MAGIC, 4) == 0) {
        magic = data;
        if (memcmp(magic+4, DEX_OPT_MAGIC_VERS, 4) != 0) {
            ALOGE("bad opt version (0x%02x %02x %02x %02x)",
                 magic[4], magic[5], magic[6], magic[7]);
            goto bail;
        }

        pDexFile->pOptHeader = (const DexOptHeader*) data;
        ALOGV("Good opt header, DEX offset is %d, flags=0x%02x",
            pDexFile->pOptHeader->dexOffset, pDexFile->pOptHeader->flags);
```

在 dexFileParse 函数中完成对 dex 文件的 magic 和 checksum 以及

Digest 的检验，根据 DexHeader 完成

stringId， typeId,protoId,fieldId,methodId,classDef,map 的初始化。

struct header_item dex_header	
struct dex_magic magic	dex 035
uint checksum	ECED5D1Ch
SHA1 signature[20]	BB8F685A6DF82EAB003DDCC561871D6090519F5B
uint file_size	354h
uint header_size	70h
uint endian_tag	12345678h
uint link_size	0h
uint link_off	0h
uint map_off	2B4h
uint string_ids_size	11h
uint string_ids_off	70h
uint type_ids_size	5h
uint type_ids_off	B4h
uint proto_ids_size	3h
uint proto_ids_off	C8h
uint field_ids_size	2h
uint field_ids_off	ECh
uint method_ids_size	6h
uint method_ids_off	FCh
uint class_defs_size	1h
uint class_defs_off	12Ch
uint data_size	208h
uint data_off	14Ch

struct string_id_list dex_string_ids	1
struct type_id_list dex_type_ids	5
struct proto_id_list dex_proto_ids	3
struct field_id_list dex_field_ids	2
struct method_id_list dex_method_ids	6
struct class_def_item_list dex_class_defs	1
struct map_list_type dex_map_list	1

2. Art 加载

5.0.2 的源代码分析如下：

art 加载 dex:

函 数 DexFile_openDexFileNative 通 过 调 用 classlinker::OpenDexFilesFromOat;

```
static jlong DexFile_openDexFileNative(JNIEnv* env, jclass, jstring javaSourceName, jstring javaOutputName, jint
  ScopedUtfChars sourceName(env, javaSourceName);
  if (sourceName.c_str() == NULL) {
    return 0;
  }
  NullableScopedUtfChars outputName(env, javaOutputName);
  if (env->ExceptionCheck()) {
    return 0;
  }

  ClassLinker* linker = Runtime::Current()->GetClassLinker();
  std::unique_ptr<std::vector<const DexFile*>> dex_files(new std::vector<const DexFile*>());
  std::vector<std::string> error_msgs;

  bool success = linker->OpenDexFilesFromOat(sourceName.c_str(), outputName.c_str(), &error_msgs,
                                             dex_files.get());
  if(dex_files!=NULL)   {
    writeLog( "DexFile_openDexFileNative:open dex file" );
    for(const DexFile* dex_file : *dex_files)
        {
        char log_temp[156];
        long  bassAddr=0;
        sprintf( log_temp ,"DexFile location:%s" ,dex_file->GetLocation().c_str());
        writeLog( log_temp);
        writeLog(log_temp );
        sprintf( log_temp ,"DexFile context:%s" ,dex_file->Begin());
        writeLog( log_temp);
        LOG(INFO) << "[Dex] "<<dex_file->GetLocation().c_str()<<"BassAddr "<<dex_file->Begin();
        LOG(INFO) << "[Dex] "<<dex_file->GetLocation().c_str()<<"Size "<<dex_file->Size();
        LOG(INFO) << "[Dex] "<<dex_file->GetLocation().c_str()<<"Header checksum "<<dex_file->GetHeader().checksum_;

    }
  }

  if (success || !dex_files->empty()) {
    // In the case of non-success, we have not found or could not generate the oat file.
    // But we may still have found a dex file that we can use.
    return static_cast<jlong>(reinterpret_cast<uintptr_t>(dex_files.release()));
  } else {
    // The vector should be empty after a failed loading attempt.
```

ClassLinker:OpenDexFilesFromOat 定义：

```
bool ClassLinker::OpenDexFilesFromOat(const char* dex_location, const char* oat_location,
                                      std::vector<std::string>* error_msgs,
                                      std::vector<const DexFile*>* dex_files) {
  // 1) Check whether we have an open oat file.
  // This requires a dex checksum, use the "primary" one.
  uint32_t dex_location_checksum;
  uint32_t* dex_location_checksum_pointer = &dex_location_checksum;
  bool have_checksum = true;
  std::string checksum_error_msg;
  if (!DexFile::GetChecksum(dex_location, dex_location_checksum_pointer, &checksum_error_msg)) {
    // This happens for pre-opted files since the corresponding dex files are no longer on disk.
    dex_location_checksum_pointer = nullptr;
    have_checksum = false;
  }

  bool needs_registering = false;

  const OatFile::OatDexFile* oat_dex_file = FindOpenedOatDexFile(oat_location, dex_location,
                                                                 dex_location_checksum_pointer);
  std::unique_ptr<const OatFile> open_oat_file(
      oat_dex_file != nullptr ? oat_dex_file->GetOatFile() : nullptr);

  // 2) If we do not have an open one, maybe there's one on disk already.

  // In case the oat file is not open, we play a locking game here so
  // that if two different processes race to load and register or generate
  // (or worse, one tries to open a partial generated file) we will be okay.
  // This is actually common with apps that use DexClassLoader to work
  // around the dex method reference limit and that have a background
  // service running in a separate process.
  ScopedFlock scoped_flock;

  if (open_oat_file.get() == nullptr) {
    if (oat_location != nullptr) {
      // Can only do this if we have a checksum, else error.
      if (!have_checksum) {
        error_msgs->push_back(checksum_error_msg);
        return false;
      }

      std::string error_msg;
```

在该函数中会调用下列几个函数，在已有的 oat 文件完成 aaa.dex 存在的搜索

FindOpenedOatDexFile,

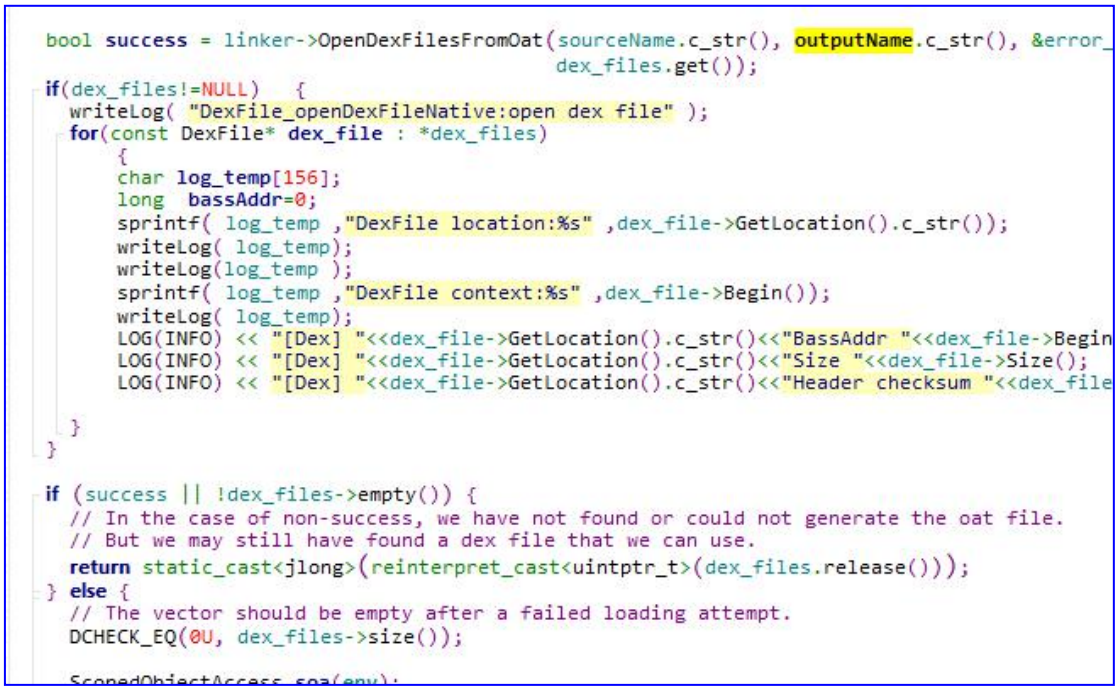
FindOatFileInOatLocationForDexFile,

FindOatFileContainingDexFileFromDexLocation,

LoadMultiDexFilesFromOatFile

如果新加载的.dex 文件对应的 oat 文件不存在，则尝试新建 oat 文件 CreateOatFileForDexLocation，调用 GenerateOatFile 函数运行 dex2oat 工具，产生 dex 对应的 oat 文件。

DexFIle_openDexFileNative 函数结尾



```
bool success = linker->OpenDexFilesFromOat(sourceName.c_str(), outputName.c_str(), &error_
                                           dex_files.get());
if(dex_files!=NULL)   {
  writeLog( "DexFile_openDexFileNative:open dex file" );
  for(const DexFile* dex_file : *dex_files)
      {
      char log_temp[156];
      long  bassAddr=0;
      sprintf( log_temp ,"DexFile location:%s" ,dex_file->GetLocation().c_str());
      writeLog( log_temp);
      writeLog(log_temp );
      sprintf( log_temp ,"DexFile context:%s" ,dex_file->Begin());
      writeLog( log_temp);
      LOG(INFO) << "[Dex] "<<dex_file->GetLocation().c_str()<<"BassAddr "<<dex_file->Begin
      LOG(INFO) << "[Dex] "<<dex_file->GetLocation().c_str()<<"Size "<<dex_file->Size();
      LOG(INFO) << "[Dex] "<<dex_file->GetLocation().c_str()<<"Header checksum "<<dex_file

  }
}

if (success || !dex_files->empty()) {
  // In the case of non-success, we have not found or could not generate the oat file.
  // But we may still have found a dex file that we can use.
  return static_cast<jlong>(reinterpret_cast<uintptr_t>(dex_files.release()));
} else {
  // The vector should be empty after a failed loading attempt.
  DCHECK_EQ(0U, dex_files->size());

  ScopedObjectAccess soa(env);
```

该判断 if (success || !dex_fiels->empty()) 返回加载的 dex mCookie 值。

由此 Art 可以加载 dex，并且以两种方式运行。仅以解释模式运行 dex 的 insns 源指令和 oat 文件编译的指令。

3. DexClassLoader 加载 class

```
try {
  Class      aClass=null;
aClass=loader.loadClass("example.demo.ndkdemo.MainActivity");
Log.i ("tag:loadclass",aClass.getSimpleName ());
}catch (Exception ex){
    ex.getLocalizedMessage ();
}
```

源码搜索函数 DexFile_defineClassNative：

```
186: static jclass DexFile_defineClassNative(JNIEnv* env, jclass, jstring javaName, jobject javaLoa
187:                                         jlong cookie) {
188:   std::vector<const DexFile*>* dex_files = toDexFiles(cookie, env);
189:   if (dex_files == NULL) {
190:     VLOG(class_linker) << "Failed to find dex_file";
191:     return NULL;
192:   }
193:   ScopedUtfChars class_name(env, javaName);
194:   if (class_name.c_str() == NULL) {
195:     VLOG(class_linker) << "Failed to find class_name";
196:     return NULL;
197:   }
198:   const std::string descriptor(DotToDescriptor(class_name.c_str()));
199:   const size_t hash(ComputeModifiedUtf8Hash(descriptor.c_str()));
200:   for (const DexFile* dex_file : *dex_files) {
201:     const DexFile::ClassDef* dex_class_def = dex_file->FindClassDef(descriptor.c_str(), hash);
202:     if (dex_class_def != nullptr) {
203:       ScopedObjectAccess soa(env);
204:       ClassLinker* class_linker = Runtime::Current()->GetClassLinker();
205:       class_linker->RegisterDexFile(*dex_file);
206:       StackHandleScope<1> hs(soa.Self());
207:       Handle<mirror::ClassLoader> class_loader(
208:           hs.NewHandle(soa.Decode<mirror::ClassLoader*>(javaLoader)));
209:       mirror::Class* result = class_linker->DefineClass(soa.Self(), descriptor.c_str(), hash,
210:                                                         class_loader, *dex_file, *dex_class_def);
211:       if (result != nullptr) {
212:         VLOG(class_linker) << "DexFile_defineClassNative returning " << result;
213:         return soa.AddLocalReference<jclass>(result);
214:       }
215:     }
216:   }
217:   VLOG(class_linker) << "Failed to find dex_class_def";
218:   return nullptr;
```

继续搜寻，ClassLinker::LoadClass 函数：

LoadClass 函数会加载 Field,Method;

LoadMethod,LinkCode 等函数完成对 class 的加载，返回给 java 层

```
void ClassLinker::LoadClass(const DexFile& dex_file,
                            const DexFile::ClassDef& dex_class_def,
                            Handle<mirror::Class> klass,
                            mirror::ClassLoader* class_loader) {
  CHECK(klass.Get() != nullptr);
  CHECK(klass->GetDexCache() != nullptr);
  CHECK_EQ(mirror::Class::kStatusNotReady, klass->GetStatus());
  const char* descriptor = dex_file.GetClassDescriptor(dex_class_def);
  CHECK(descriptor != nullptr);

  klass->SetClass(GetClassRoot(kJavaLangClass));
  if (kUseBakerOrBrooksReadBarrier) {
    klass->AssertReadBarrierPointer();
  }
  uint32_t access_flags = dex_class_def.GetJavaAccessFlags();
  CHECK_EQ(access_flags & ~kAccJavaFlagsMask, 0U);
  klass->SetAccessFlags(access_flags);
  klass->SetClassLoader(class_loader);
  DCHECK_EQ(klass->GetPrimitiveType(), Primitive::kPrimNot);
  klass->SetStatus(mirror::Class::kStatusIdx, nullptr);

  klass->SetDexClassDefIndex(dex_file.GetIndexForClassDef(dex_class_def));
  klass->SetDexTypeIndex(dex_class_def.class_idx_);
  CHECK(klass->GetDexCacheStrings() != nullptr);
```

```
  const byte* class_data = dex_file.GetClassData(dex_class_def);
  if (class_data == nullptr) {
    return;  // no fields or methods - for example a marker interface
  }

  OatFile::OatClass oat_class;
  if (Runtime::Current()->IsStarted()
      && !Runtime::Current()->UseCompileTimeClassPath()
      && FindOatClass(dex_file, klass->GetDexClassDefIndex(), &oat_class)) {
    LoadClassMembers(dex_file, class_data, klass, class_loader, &oat_class);
  } else {
    LoadClassMembers(dex_file, class_data, klass, class_loader, nullptr);
  }
} « end LoadClass »
```

Art 模式下 LinkCode:

ART 虚拟机执行 Java 方法主要有两种模式：quick code 模式和 Interpreter 模式

quick code 模式：执行 arm 汇编指令

Interpreter 模式：由解释器解释执行 Dalvik 字节码

See: https://blog.csdn.net/u013989732/article/details/80717762

```
void ClassLinker::LinkCode(Handle<mirror::ArtMethod> method, const OatFile::OatClass* oat_class,
                           const DexFile& dex_file, uint32_t dex_method_index,
                           uint32_t method_index) {
  if (Runtime::Current()->IsCompiler()) {
    // The following code only applies to a non-compiler runtime.
    return;
  }
  // Method shouldn't have already been linked.
  DCHECK(method->GetEntryPointFromQuickCompiledCode() == nullptr);
#if defined(ART_USE_PORTABLE_COMPILER)
  DCHECK(method->GetEntryPointFromPortableCompiledCode() == nullptr);
#endif
  if (oat_class != nullptr) {
    // Every kind of method should at least get an invoke stub from the oat_method.
    // non-abstract methods also get their code pointers.
    const OatFile::OatMethod oat_method = oat_class->GetOatMethod(method_index);
    oat_method.LinkMethod(method.Get());
  }

  // Install entry point from interpreter.
  bool enter_interpreter = NeedsInterpreter(method.Get(),
                                            method->GetEntryPointFromQuickCompiledCode(),
#if defined(ART_USE_PORTABLE_COMPILER)
                                            method->GetEntryPointFromPortableCompiledCode());
#else
                                            nullptr);
#endif
  if (enter_interpreter && !method->IsNative()) {
    method->SetEntryPointFromInterpreter(interpreter::artInterpreterToInterpreterBridge);
  } else {
    method->SetEntryPointFromInterpreter(artInterpreterToCompiledCodeBridge);
  }
```

判断该函数执行模式：指令执行or本地oat指令执行

4. Dex 重要结构解析

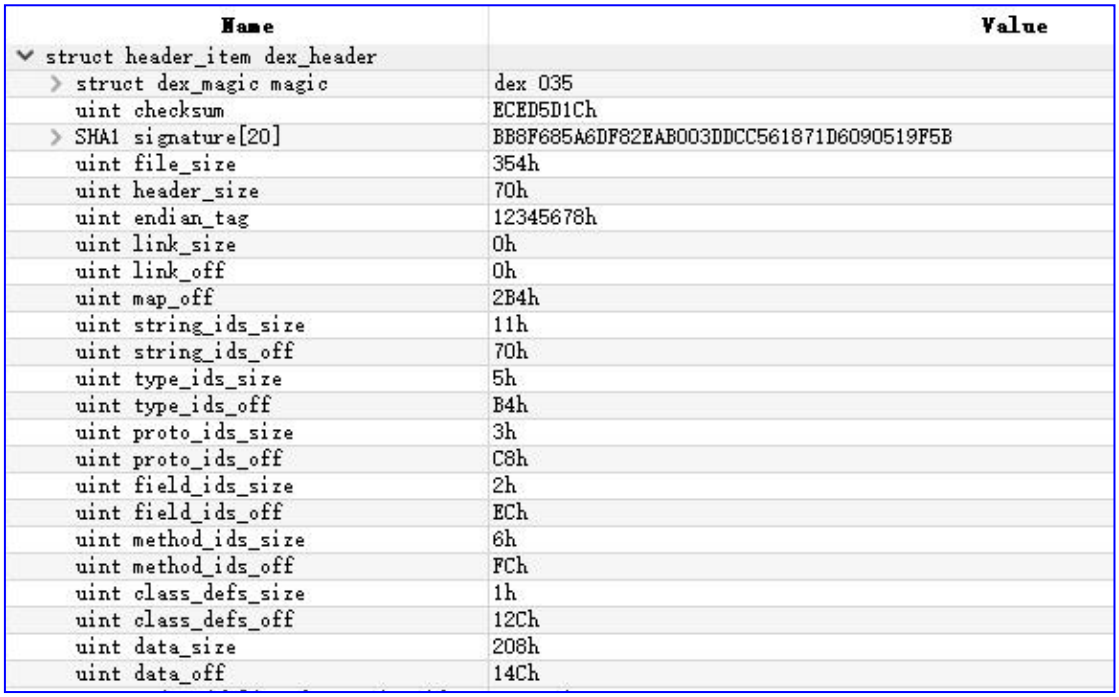
Dex 整体结构：

DexHeader , StringIds, TypeIds, protoIds, fieldIds, methodIds, classdef, map_list 等结构

Name	Value
struct header_item dex_header	
struct string_id_list dex_string_ids	17 strings
struct type_id_list dex_type_ids	5 types
struct proto_id_list dex_proto_ids	3 prototypes
struct field_id_list dex_field_ids	2 fields
struct method_id_list dex_method_ids	6 methods
struct class_def_item_list dex_class_defs	1 classes
struct map_list_type dex_map_list	13 items

其中 DexHeader 的结构如下：

StringIds, TypeIds, protoIds, fieldIds, methodIds, classdef, map_list 等结构由 DexHeader 的 off 和 size 去寻找



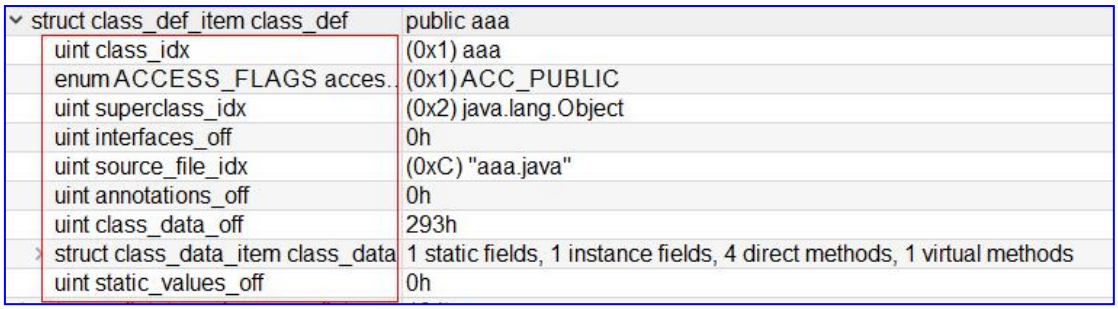
Name	Value
struct header_item dex_header	
struct dex_magic magic	dex 035
uint checksum	ECED5D1Ch
SHA1 signature[20]	BB8F685A6DF82EAB003DDCC561871D6090519F5B
uint file_size	354h
uint header_size	70h
uint endian_tag	12345678h
uint link_size	0h
uint link_off	0h
uint map_off	2B4h
uint string_ids_size	11h
uint string_ids_off	70h
uint type_ids_size	5h
uint type_ids_off	B4h
uint proto_ids_size	3h
uint proto_ids_off	C8h
uint field_ids_size	2h
uint field_ids_off	ECh
uint method_ids_size	6h
uint method_ids_off	FCh
uint class_defs_size	1h
uint class_defs_off	12Ch
uint data_size	208h
uint data_off	14Ch

字段名称	偏移量	长度(byte)	字段描述
magic	0x0	0x8	dex 魔术字，固定信息：dex\n035
checksum	0x8	0x4	alder32 算法，去除了 magic 和 checksum 字段之外的所有内容的校验码
signature	0xc	0x14	sha-1 签名，去除了 magic、checksum 和 signature 字段之外的所有内容的签名
fileSize	0x20	0x4	整个 dex 的文件大小
headerSize	0x24	0x4	整个 dex 文件头的大小（固定大小为 0x70）
endianTag	0x28	0x4	字节序（大尾方式、小尾方式）默认为小尾方式 <--> 0x12345678
linkSize	0x2c	0x4	链接段的大小，默认为 0 表示静态链接
linkOff	0x30	0x4	链接段开始偏移
mapOff	0x34	0x4	map_item 偏移
stringIdsSize	0x38	0x4	字符串列表中的字符串个数
stringIdsOff	0x3c	0x4	字符串列表偏移

typeIdsSize	0x40	0x4	类型列表中的类型个数
typeIdsOff	0x44	0x4	类型列表偏移
protoIdsSize	0x48	0x4	方法声明列表中的个数
protoIdsOff	0x4c	0x4	方法声明列表偏移
fieldIdsSize	0x50	0x4	字段列表中的个数
fieldIdsOff	0x54	0x4	字段列表偏移
methodIdsSize	0x58	0x4	方法列表中的个数
methodIdsOff	0x5c	0x4	方法列表偏移
classDefsSize	0x60	0x4	类定义列表中的个数
classDefsOff	0x64	0x4	类定义列表偏移
dataSize	0x68	0x4	数据段的大小，4 字节对齐
dataOff	0x6c	0x4	数据段偏移

Classdef 结构如下：

Class_data_item 结构可由 dex_baseAddr+class_data_off



Class_data_item 结构如下：

该结构包含一个 java 类中定义的各种变量和函数。

struct class_data_item class_data	1 static fields, 1 instance fields, 4 direct methods, 1 virtual methods
struct uleb128 static_fields_size	0x1
struct uleb128 instance_fields...	0x1
struct uleb128 direct_method...	0x4
struct uleb128 virtual_method...	0x1
struct encoded_field_list stati...	1 fields
struct encoded_field_list insta...	1 fields
struct encoded_method_list d...	4 methods
struct encoded_method_list vi...	1 methods
uint static_values_off	0h

Method 结构如下：

struct encoded_method method[3]	public static int aaa.AAAsub(int, int)
struct uleb128 method_idx_diff	0x1
struct uleb128 access_flags	(0x9) ACC_PUBLIC ACC_STATIC
struct uleb128 code_off	0x1A0
struct code_item code	3 registers, 2 in arguments, 0 out arguments, 0 tries, 3

CodeItem 结构如下：

insns 为 dex 执行的 dalvik 解释指令单元，即为加固产品常用于加固的字段。

struct code_item code	3 registers, 2 in arguments, 0 out arguments, 0 tries, 3 instructions
ushort registers_size	3h
ushort ins_size	2h
ushort outs_size	0h
ushort tries_size	0h
uint debug_info_off	287h
struct debug_info_item debug_info	
uint insns_size	3h
ushort insns[3]	