MICROOH 麦可网

Android-从程序员到架构师之路

出品人：Sundy
讲师：高焕堂（台湾）

http://www.microoh.com

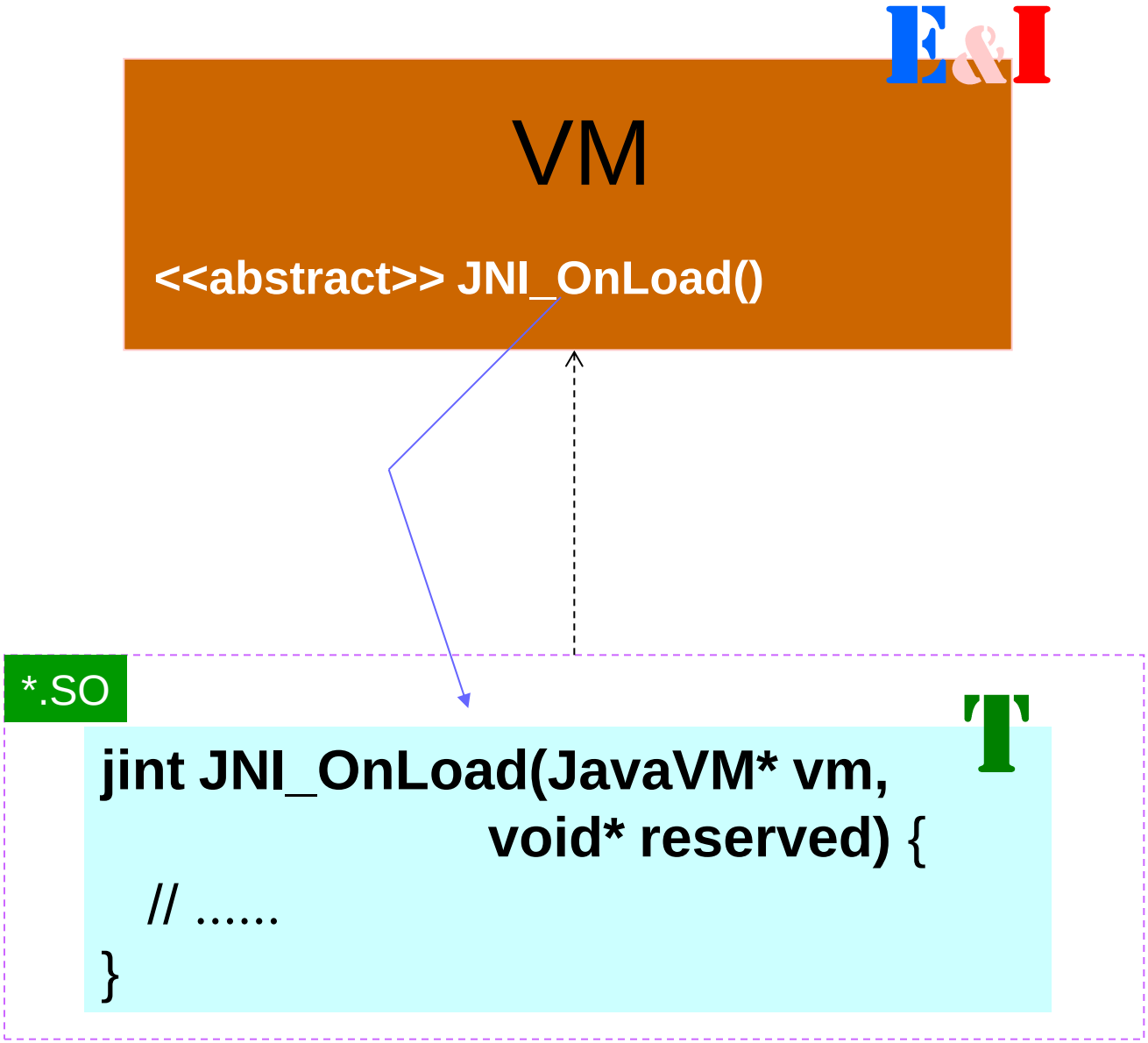
C02_e

认识JNI开发与NDK (e)

By 高煥堂

VM的接口

* .so的入口函数：
JNI_OnLoad()


E&I
VM
<<abstract>> JNI_OnLoad()
*.SO
T
jint JNI_OnLoad(JavaVM* vm,
void* reserved) {
// ……
}

- 执行**System**.loadLibrary()函数时，VM会反向调用*.so里的JNI_OnLoad()函数。用途有二：

1. VM询问此*.so使用的JNI版本编号。
2. VM要求*.so做一些初期设定工作(Initialization)，例如登记<函数名称表>。

- 例如，在Android的/system/lib/libmedia_jni.so档案里，就提供了JNI_OnLoad()函数，其程序码片段为：

```
// #define LOG_NDEBUG 0
#define LOG_TAG "MediaPlayer-JNI"
// .....
jint JNI_OnLoad(JavaVM* vm, void* reserved) {
    JNIEnv* env = NULL;
    jint result = -1;
    if (vm->GetEnv((void**) &env, JNI_VERSION_1_4) != JNI_OK) {
        LOGE("ERROR: GetEnv failed\n");       goto bail;
    }
    assert(env != NULL);
    if (register_android_media_MediaPlayer(env) < 0) {
        LOGE("ERROR: MediaPlayer native registration failed\n");
        goto bail;
    }
    /* success -- return valid version number */
    result = JNI_VERSION_1_4;
bail:    return result;
}
// KTHXBYE
```

- 此函数回传JNI_VERSION_1_4值给VM，于是VM知道了其所使用的JNI版本了。
- 此外， JNI_OnLoad()函数也做了一些初期的动作，例如指令：

```
if (register_android_media_MediaPlayer(env) < 0) {
   LOGE("ERROR: MediaPlayer native registration failed
\n");
  goto bail;
}
```

- 就将此*.so的<函数名称表>登记到VM里，以便能加快后续调用本地函数之效率。

JNI_OnUnload()函数与JNI_OnLoad()相对应的。

- 当VM释放该C模块时，则会调用JNI_OnUnload()函数来进行善后清除动作。

- Java类别透过VM而调用到本地函数。
- 一般是仰赖VM去寻找*.so里的本地函数。如果需要连续调用很多次，每次都需要寻找一遍，会多花许多时间。
- 此时，将此*.so的<函数名称表>登记到VM里。例如，在Android的/system/lib/libmedia_jni.so档案里的程序码片段如下：

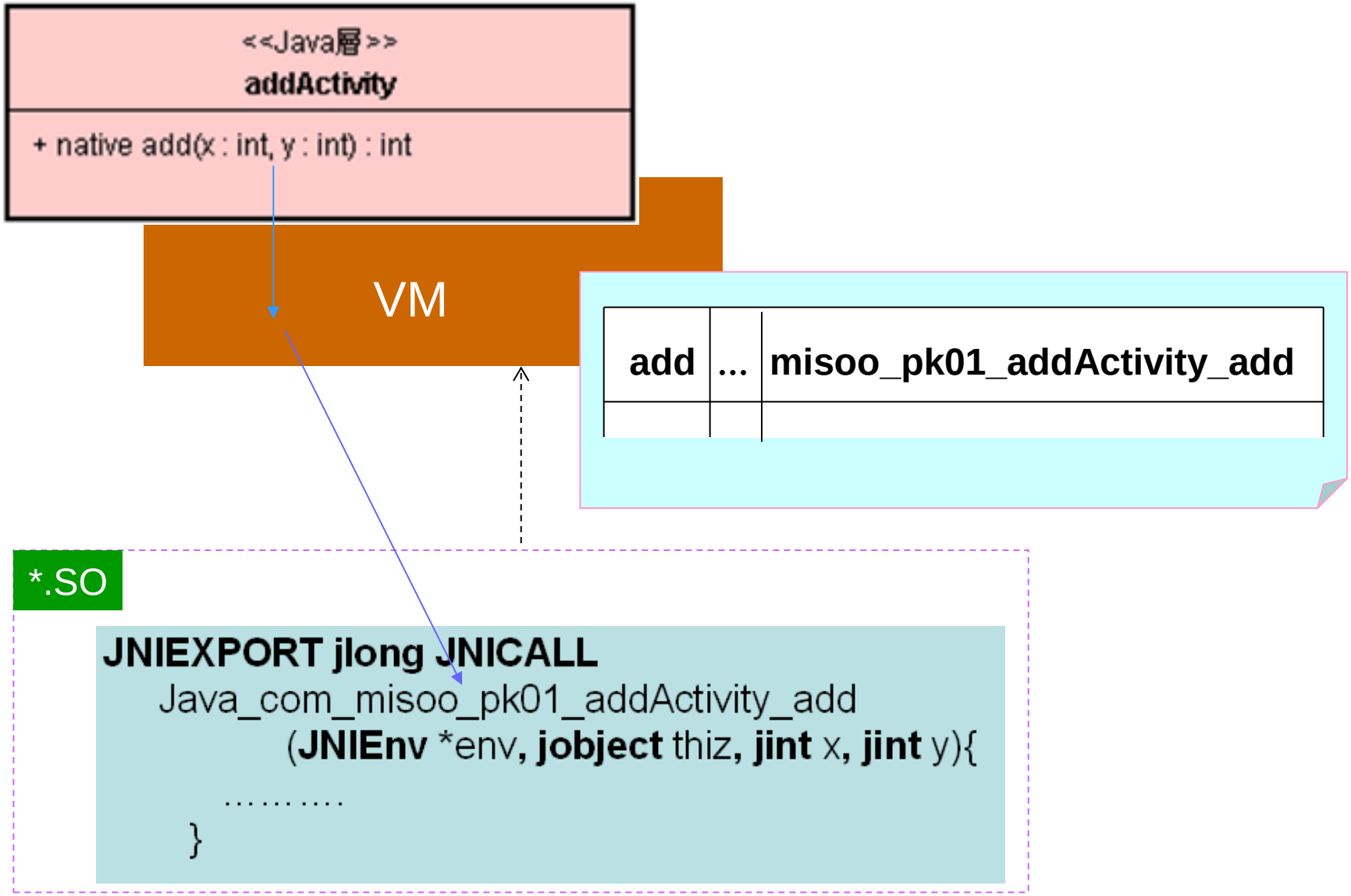
```
// #define LOG_NDEBUG 0
#define LOG_TAG "MediaPlayer-JNI"
// .........
static JNINativeMethod gMethods[] = {
    {"setDataSource",   "(Ljava/lang/String;)V",
  (void *)android_media_MediaPlayer_setDataSource},
    {"setDataSource",    "(Ljava/io/FileDescriptor;JJ)V",
(void *)android_media_MediaPlayer_setDataSourceFD},
    {"prepare",   "()V",     (void *)android_media_MediaPlayer_prepare},
    {"prepareAsync",   "()V",
                  (void *)android_media_MediaPlayer_prepareAsync},
    {  "_start",   "()V",   (void *)android_media_MediaPlayer_start},
    {  "_stop",   "()V",    (void *)android_media_MediaPlayer_stop},
     (省略)
};
```

```
// .....
static int register_android_media_MediaPlayer(JNIEnv *env) {
  .....
    return AndroidRuntime::registerNativeMethods(env,
                "android/media/MediaPlayer", gMethods, NELEM(gMethods));
}
// .....
jint JNI_OnLoad(JavaVM* vm, void* reserved){
  .....
    if (register_android_media_MediaPlayer(env) < 0) {
        LOGE("ERROR: MediaPlayer native registration failed\n");
        goto bail;
    }
  // .....
}
```

- JNI_OnLoad()调用register_android_media_MediaPlayer()函数。
- 此时，就调用到AndroidRuntime::registerNativeMethods()函数，向VM登记gMethods[]表格。

- 登记gMethods[]表格的用途有二：
  1. 更有效率去找到C函数。
  2. 可在执行期间弹性进行抽换。

- 由于gMethods[]是一个<名称，函数指针>对照表，在程序执行时，可多次调用registerNativeMethods()来更换本地函数之指针，而达到弹性抽换本地函数之目的。


<<Java層>>
addActivity
+ native add(x : int, y : int) : int
VM
add
...
misoo_pk01_addActivity_add
*.SO
JNIEXPORT jlong JNICALL
Java_com_misoo_pk01_addActivity_add
(JNIEnv *env, jobject thiz, jint x, jint y){
.....
}

创造底层变动的自由度

(没钱就改版，改版就有钱)

Thanks...

高煥堂