[conowen]大钟的专栏

Android的NDK开发(1)————Android JNI简介与调用流程

分类： Android的NDK开发 2012-04-29 18:12 33312人阅读 评论(5) 收藏 举报

android jni 虚拟 exception

```
/*****************************************************************************************
 * author：conowen@大钟
 * E-mail：conowen@hotmail.com
 *site:http://menwoo.com/
 *深圳市大望谷科技有限公司
 * http://blog.csdn.net/conowen
 * 注：本文为原创，仅作为学习交流使用，转载请标明作者及出处。
 *****************************************************************************************/
```

1、JNI简介

JNI全称为Java Native Interface（JAVA本地调用）。从Java1.1开始，JNI成为java平台的一部分，它允许Java代码和其他语言写的代码（如C&C++）进行交互。并非从Android发布才引入JNI的概念的。

2、JNI与NDK

简单来说，Android的NDK提供了一些交叉编译工具链和Android自带的库，这些Android的库可以让开发者在编写本地语言的程序时调用。而NDK提供的交叉编译工具链就对已经编写好的C&C++代码进行编译，生成库。

当然了，你也可以自己搭建交叉编译环境，而不用NDK的工具和库。然后生成库，只要规范操作，一样可以生成能让JAVA层成功调用的库文件的。

利用NDK进行编译本地语言可以参考这篇博文：http://blog.csdn.net/conowen/article/details/7522667

3、JNI 调用流程

众所周知，Android的应用层的类都是以Java写的，这些Java类编译为Dex文件之后，必须靠Dalvik虚拟机(Virtual Machine)来执行。假如在执行java程序时，需要载入C&C++函数时，Dalvik虚拟机就会去加载C&C++的库，（System.loadLibrary("libName");）让java层能顺利地调用这些本地函数。需要清楚一点，这些C&C++的函数并不是在Dalvik虚拟机中运行的，所以效率和速度要比在Dalvik虚拟机中运行得快很多。

Dalvik虚拟机成功加载库之后，就会自动地寻找库里面的JNI_OnLoad函数，这个函数用途如下：
(1)告诉Dalvik虚拟机此C库使用哪一个JNI版本。如果你的库里面没有写明JNI_OnLoad()函数，VM会默认该库使用最老的JNI 1.1版本。但是新版的JNI做了很多的扩充，也优化了一些内容，如果需要使用JNI的新版功能，就必须在JNI_OnLoad()函数声明JNI的版本。如

```
[java]
01. result = JNI_VERSION_1_4;
```

瑞芯微RK (3)
嵌入式相关 (1)
计算机相关 (2)
Android多媒体&流媒体开发 (6)
Android的NDK开发 (5)
所想所感 (1)
所想所感 ios (1)

文章存档

2015年09月 (1)
2014年11月 (4)
2014年04月 (2)
2012年08月 (6)
2012年07月 (4)
展开

阅读排行

Android学习笔记(21)—— (49089)
【整理】Android-Recove (45493)
Android的NDK开发(1)— (33300)
Android学习笔记(13)—— (26417)
Android学习笔记(12)—— (22280)
RKAndroidTool工具的各 (21335)
Android图形系统之Surfa (21228)
【整理】Libav、FFmpeg (21201)
Android的文件系统结构 (20983)
Android的NDK开发(3)— (20125)

评论排行

Android学习笔记(21)—— (74)
关于havlenapetr-FFMpe (63)
Android多媒体开发（3） (50)
Android多媒体开发（4） (46)
Android多媒体开发（5） (41)
Android学习笔记(20)—— (15)
Android学习笔记(12)—— (14)
Android多媒体开发（2） (14)
Android学习笔记(1)—— (9)
Android的NDK开发(2)— (8)

推荐文章

*HTML抽屉效果的实现与展示
*2015年校招求职之旅
* iOS9使用提示框的正确实现方式
* 你不知道的JavaScript--Item18 JScript的Bug与内存管理
Android Studio中配置及使用OpenCV示例
* 最老程序员创业开发实训4--IOS平台下MVC架构

最新评论

Android多媒体开发（5）———— guonanyun: 楼主,可以提供一下完整demo的下载地址吗?
Android的NDK开发(4)————J wangjg0317: 写得很好，谢谢楼

当没有JNI_OnLoad()函数时，Android调试信息会做出如下提示（No JNI_OnLoad found）

```
[java]
04-29 13:53:12.184: D/dalvikvm(361): Trying to load lib /data/data/com.conowen.helloworld/lib/libl
04-29 13:53:12.204: D/dalvikvm(361): Added shared lib /data/data/com.conowen.helloworld/lib/libHe
04-29 13:53:12.204: D/dalvikvm(361): No JNI_OnLoad found in /data/data/com.conowen.helloworld/lib,
```

(2)因为Dalvik虚拟机加载C库时，第一件事是调用JNI_OnLoad()函数，所以我们可以在JNI_OnLoad()里面进行一些初始化工作，如注册JNI函数等等。注册本地函数，可以加快java层调用本地函数的效率。

另外：与JNI_OnLoad()函数相对应的有JNI_OnUnload()函数，当虚拟机释放该C库时，则会调用JNI_OnUnload()函数来进行善后清除动作。

4、例子（关于jni里面的数据类型转换与常用jni方法下一篇博文介绍）

下面以havlenapetr的FFmpeg工程里面的onLoad.cpp为例详细说一下：

```cpp
//onLoad.cpp文件

#define TAG "ffmpeg_onLoad"

#include <stdlib.h>
#include <android/log.h>
#include "jniUtils.h"

extern "C" {

extern int register_android_media_FFMpegAVRational(JNIEnv *env);

#ifdef BUILD_WITH_CONVERTOR
extern int register_android_media_FFMpeg(JNIEnv *env);
#endif

extern int register_android_media_FFMpegAVFormatContext(JNIEnv *env);
extern int register_android_media_FFMpegAVInputFormat(JNIEnv *env);

}

extern int register_android_media_FFMpegAVCodecContext(JNIEnv *env);
extern int register_android_media_FFMpegUtils(JNIEnv *env);
extern int register_android_media_FFMpegAVFrame(JNIEnv *env);

#ifdef BUILD_WITH_PLAYER
extern int register_android_media_FFMpegPlayerAndroid(JNIEnv *env);
#endif

static JavaVM *sVm;

/*
 * Throw an exception with the specified class and an optional message.
 */
int jniThrowException(JNIEnv* env, const char* className, const char* msg) {
    jclass exceptionClass = env->FindClass(className);
    if (exceptionClass == NULL) {
        __android_log_print(ANDROID_LOG_ERROR,
                TAG,
                "Unable to find exception class %s",
                        className);
        return -1;
    }

    if (env->ThrowNew(exceptionClass, msg) != JNI_OK) {
        __android_log_print(ANDROID_LOG_ERROR,
                TAG,
```

```
                "Failed throwing '%s' '%s'",
                className, msg);
    }
    return 0;
}

JNIEnv* getJNIEnv() {
    JNIEnv* env = NULL;
    if (sVm->GetEnv((void**) &env, JNI_VERSION_1_4) != JNI_OK) {
        __android_log_print(ANDROID_LOG_ERROR,
                            TAG,
                            "Failed to obtain JNIEnv");
        return NULL;
    }
    return env;
}

/*
 * Register native JNI-callable methods.
 *
 * "className" looks like "java/lang/String".
 */
int jniRegisterNativeMethods(JNIEnv* env,
                             const char* className,
                             const JNINativeMethod* gMethods,
                             int numMethods)
/*从com_media_ffmpeg_FFMpegPlayer.cpp文件跳到此，完成最后的注册
 * 向 Dalvik虚拟机(即AndroidRuntime)登记传过来的参数gMethods[]所含的本地函数
 */
{
    jclass clazz;

    __android_log_print(ANDROID_LOG_INFO, TAG, "Registering %s natives\n", className);
    clazz = env->FindClass(className);
    if (clazz == NULL) {
        __android_log_print(ANDROID_LOG_ERROR, TAG, "Native registration unable to find class
        return -1;
    }
    if (env->RegisterNatives(clazz, gMethods, numMethods) < 0) {
        __android_log_print(ANDROID_LOG_ERROR, TAG, "RegisterNatives failed for '%s'\n", classI
        return -1;
    }
    return 0;
}
//Dalvik虚拟机加载C库时，第一件事是调用JNI_OnLoad()函数
jint JNI_OnLoad(JavaVM* vm, void* reserved) {
    JNIEnv* env = NULL;//定义JNI Env
    jint result = JNI_ERR;
    sVm = vm;
    /*JavaVM::GetEnv 原型为 jint (*GetEnv)(JavaVM*, void**, jint);
     * GetEnv()函数返回的  Jni 环境对每个线程来说是不同的，
     *  由于Dalvik虚拟机通常是Multi-threading的。每一个线程调用JNI_OnLoad()时，
     *  所用的JNI Env是不同的，因此我们必须在每次进入函数时都要通过vm->GetEnv重新获取
     *
     */
    //得到JNI Env
    if (vm->GetEnv((void**) &env, JNI_VERSION_1_4) != JNI_OK) {
        __android_log_print(ANDROID_LOG_ERROR, TAG, "GetEnv failed!");
        return result;
    }

    __android_log_print(ANDROID_LOG_INFO, TAG, "loading . . .");

/*开始注册
 * 传入参数是JNI env
 * 由于下面有很多class，只以register_android_media_FFMpegPlayerAndroid(env)为例子
 */

#ifdef BUILD_WITH_CONVERTOR
    if(register_android_media_FFMpeg(env) != JNI_OK) {
        __android_log_print(ANDROID_LOG_ERROR, TAG, "can't load android_media_FFMpeg");
        goto end;
    }
#endif

    if(register_android_media_FFMpegAVFormatContext(env) != JNI_OK) {
        __android_log_print(ANDROID_LOG_ERROR, TAG, "can't load android_media_FFMpegAVFormatCo
        goto end;
    }
```

```
127.
128.     if(register_android_media_FFMpegAVCodecContext(env) != JNI_OK) {
129.         __android_log_print(ANDROID_LOG_ERROR, TAG, "can't load android_media_FFMpegAVCodecCon
130.         goto end;
131.     }
132.
133.     if(register_android_media_FFMpegAVRational(env) != JNI_OK) {
134.         __android_log_print(ANDROID_LOG_ERROR, TAG, "can't load android_media_FFMpegAVRational
135.         goto end;
136.     }
137.
138.     if(register_android_media_FFMpegAVInputFormat(env) != JNI_OK) {
139.         __android_log_print(ANDROID_LOG_ERROR, TAG, "can't load android_media_FFMpegAVInputFor
140.         goto end;
141.     }
142.
143.     if(register_android_media_FFMpegUtils(env) != JNI_OK) {
144.         __android_log_print(ANDROID_LOG_ERROR, TAG, "can't load android_media_FFMpegUtils");
145.         goto end;
146.     }
147.
148.     if(register_android_media_FFMpegAVFrame(env) != JNI_OK) {
149.         __android_log_print(ANDROID_LOG_ERROR, TAG, "can't load android_media_FFMpegAVFrame");
150.         goto end;
151.     }
152.
153. #ifdef BUILD_WITH_PLAYER
154.     if(register_android_media_FFMpegPlayerAndroid(env) != JNI_OK) {//跳到----》
com_media_ffmpeg_FFMpegPlayer.cpp文件
155.         __android_log_print(ANDROID_LOG_ERROR, TAG, "can't load android_media_FFMpegPlayerAndr
156.         goto end;
157.     }
158. #endif
159.
160.     __android_log_print(ANDROID_LOG_INFO, TAG, "loaded");
161.
162.     result = JNI_VERSION_1_4;
163.
164. end:
165.     return result;
166. }
```

[cpp]

```
01. //com_media_ffmpeg_FFMpegPlayer.cpp文件
02. /*
03.  *
04.  * 由于代码量较大，com_media_ffmpeg_FFMpegPlayer.cpp开始的一部分省略，只是贴出注册函数的相关部分。
05.  */
06. static const char* const kClassPathName = "com/media/ffmpeg/FFMpegPlayer";
07. /*
08.  * 由于gMethods[]是一个<名称，函数指针>对照表，在程序执行时，
09.  * 可多次调用registerNativeMethods()函数来更换本地函数的指针，
10.  * 从而达到弹性调用本地函数的目的。
11.  */
12. static JNINativeMethod gMethods[] = {
13.     {"setDataSource",       "
(Ljava/lang/String;)V",           (void *)com_media_ffmpeg_FFMpegPlayer_setDataSource},
14.     {"_setVideoSurface",    "
(Landroid/view/Surface;)V",       (void *)com_media_ffmpeg_FFMpegPlayer_setVideoSurface},
15.     {"prepare",             "
()V",                             (void *)com_media_ffmpeg_FFMpegPlayer_prepare},
16.     {"_start",              "
()V",                             (void *)com_media_ffmpeg_FFMpegPlayer_start},
17.     {"_stop",               "
()V",                             (void *)com_media_ffmpeg_FFMpegPlayer_stop},
18.     {"getVideoWidth",       "
()I",                             (void *)com_media_ffmpeg_FFMpegPlayer_getVideoWidth},
19.     {"getVideoHeight",      "
()I",                             (void *)com_media_ffmpeg_FFMpegPlayer_getVideoHeight},
20.     {"seekTo",              "
(I)V",                            (void *)com_media_ffmpeg_FFMpegPlayer_seekTo},
21.     {"_pause",              "
()V",                             (void *)com_media_ffmpeg_FFMpegPlayer_pause},
```

```
22.     {"isPlaying",          "
    ()Z",                           (void *)com_media_ffmpeg_FFMpegPlayer_isPlaying},
23.     {"getCurrentPosition",  "
    ()I",                           (void *)com_media_ffmpeg_FFMpegPlayer_getCurrentPosition},
24.     {"getDuration",         "
    ()I",                           (void *)com_media_ffmpeg_FFMpegPlayer_getDuration},
25.     {"_release",            "
    ()V",                           (void *)com_media_ffmpeg_FFMpegPlayer_release},
26.     {"_reset",              "
    ()V",                           (void *)com_media_ffmpeg_FFMpegPlayer_reset},
27.     {"setAudioStreamType",  "
    (I)V",                          (void *)com_media_ffmpeg_FFMpegPlayer_setAudioStreamType},
28.     {"native_init",         "
    ()V",                           (void *)com_media_ffmpeg_FFMpegPlayer_native_init},
29.     {"native_setup",        "
    (Ljava/lang/Object;)V",         (void *)com_media_ffmpeg_FFMpegPlayer_native_setup},
30.     {"native_finalize",     "
    ()V",                           (void *)com_media_ffmpeg_FFMpegPlayer_native_finalize},
31.     {"native_suspend_resume", "
    (Z)I",                      (void *)com_media_ffmpeg_FFMpegPlayer_native_suspend_resume},
32. };
33. 
34. int register_android_media_FFMpegPlayerAndroid(JNIEnv *env) {
35.     return jniRegisterNativeMethods(env, kClassPathName, gMethods, sizeof(gMethods) / sizeof(g
36.     /*跳到OnLoad.cpp文件中的
37.      * jint jniRegisterNativeMethods(JNIEnv* env,
38.                                 const char* className,
39.                                 const JNINativeMethod* gMethods,
40.                                 int numMethods)
41. 
42.      */
43. }
```


上一篇 Android多媒体开发（2）————使用Android NKD编译原版FFmpeg

下一篇 Android的NDK开发(2)————利用Android NDK编写一个简单的HelloWorld

5楼 lancelee2015 2015-07-20 16:26发表

写的很好 学习了~