总结归纳JNI和NDK相关的知识点。

转载请注明: http://blog.csdn.net/feather_wch/article/details/79599270

JNI和NDK编程

版本:2018/3/18-1(11:36

JNI和NDK简介

1、JNI是什么？作用?

> 1. Java Native Interface(java本地接口)
> 2. 方便 Java 调用 C、C++ 等本地代码所封装的一层接口

2、为什么需要JNI?

> 1. java 特点是 跨平台 ，但是会导致 本地交互能力 不够强大，一些 操作系统相关特性 Java是无法完成的-因此 Java 提供 JNI 专门用于和本地代码交互，依次增强 本地交互能力
> 2. 可以使用现有的开源库，现在很多优秀的开源库都是用 C/C++ 编写的。
> 3. 代码的保护，Android apk 的 java 代码容易被反编译，而 C/C++ 更难反编译。
> 4. 便于移植，用 C/C++ 写的库可以方便在其他嵌入式平台使用。

3、NDK是什么?

> 1. NDK 是Android所提供的 工具集合
> 2. 通过 NDK 可以在Android中更加方便地通过 JNI 来访问本地代码，如 C\C++
> 3. NDK 还提供 交叉编译器 ，只需要简单地修改 mk文件 就可以生成 特定CPU平台的动态库

4、NDK的优点

> 1. 提高代码的安全性---so库反编译比较困难
> 2. 可以方便地使用目前已有的 C/C++开源库
> 3. 便于平台间的移植---通过 C/C++ 实现的动态库可以很方便在其他平台上使用
> 4. 提高程序在某些特定情况下的执行效率，但并不能明显提升Android程序的性能。

5、AS3.0如何集成JNI功能

1. AS3.0中下载NDK,CMake(高级的编译配置工具),LLDB(高效的C/C++的调试器)
2. Java的native方法，通过javah生成对应的头文件
3. 编写C/C++代码
4. CMakeList文件进行配置

6、cpp代码中 JNIEXPORT 和 JNICALL 关键字的作用?

1. 这两个关键字是两个 宏定义
2. 用于说明该函数为 JNI函数，Java虚拟机加载的时候会链接对应的 native方法

7、Java虚拟机加载so库时，如何找到Java层中对应的hative方法？(静态注册)

1. 通过 JNI函数的函数名 去匹配： Java_PackageName_ClassName_NativeMethodName
2. 可以在 app/build/intermediates/classes/debug 通过 javah -d jni 包名.类名 在 debug目录下 生成 jni目录(-d jni参数指定)，并在其中生成对应的 H头文件

8、自动生成的JNI函数中的参数 jobject 是什么?

1. 当前和该JNI函数所连接的native方法所属的类对象(等价于Java中的this)

9、JNI的动态注册是什么?

1. 静态注册native方法的过程，是将 Java层的native方法 和 JNI函数 一一对应
2. 动态注册能让 Java层的native方法 和任意 JNI函数 连接起来。

10、JNI进行动态注册的步骤

1. Java中定义 native函数
2. C/C++中定义对应的 JNI函数(名称随意不需要对应)
3. C/C++中定义 JNINativeMethod的数据 ---指明多组 native函数和JNI函数对应关系
4. C/C++中的 JNI_OnLoad() 中进行 动态注册

```
public class JniUtils2 {
    static {
        System.loadLibrary("native-lib");
    }
    public native String getStringFromC();

    public native void dynamicFunction();

    public native void dynamicFunction2();
}
```

```
#include "com_hao_jniapp_JniUtils2.h"
#include "android/log.h"

extern "C" {
JNIEXPORT jstring JNICALL Java_com_hao_jniapp_JniUtils2_getStringFromC
        (JNIEnv *env, jobject jobject1) {
    return env->NewStringUTF("我是来自JniUtils2的Native方法");
}

// 6. JNI函数(Java层的native方法的具体实现)
static void jniDynamicLog(JNIEnv *evn, jobject obj){
    __android_log_print(ANDROID_LOG_INFO, "feather", "JNI's Log: hello Java! Im from JNI");
}
static void jniTest(JNIEnv *evn, jobject obj){
    __android_log_print(ANDROID_LOG_INFO, "feather", "JNI's Log: hello Java! Im from JNI's Test
}

/**==========================
 * 7. JNINativeMethod结构体：记录java的native方法和JNI函数的对应关系
 *==========================*/
JNINativeMethod nativeMethod[] = {  {"dynamicFunction", //1. Java层native方法的方法名
                                        "()V",      //2. Java层native方法的描述符
                                        (void*)jniDynamicLog} //3. JNI函数的指针
                  ,{"dynamicFunction2", "()V", (void*)jniTest}  //4. 另一个绑定关系的native/jr
};

/**============================================================
 * 1. JNI_OnLoad会在Java中的`System.loadLibarary()`加载so库的时候调用
 * @param jvm *jvm为Java虚拟机实例，JavaVM为一个结构体。
 *============================================================*/
JNIEXPORT jint JNICALL JNI_OnLoad(JavaVM *jvm, void *reserved) {

    JNIEnv *env;
    //2. *jvm调用GetEnv()会获得JNIEnv变量
    if (jvm->GetEnv((void**) &env, JNI_VERSION_1_4) != JNI_OK) {

        return -1;
    }
    __android_log_print(ANDROID_LOG_INFO, "feather", "JNI's Log: JNI_OnLoad...");
    //3. JNIEnv结构体指向一个函数表，获取到JniUtils对象
    jclass clz = env->FindClass("com/hao/jniapp/JniUtils2");
    //4. 将JniUtils对象和nativeMethod结构体所描述的Java层native方法和JNI函数 进行动态注册
    env->RegisterNatives(clz, nativeMethod, sizeof(nativeMethod)/sizeof(nativeMethod[0]));

    //5. return 当前使用的JNI版本
    return JNI_VERSION_1_4;
}

}
```

11、JNINativeMethod的作用

> 1. 作为一个结构体，描述了native方法和JNI函数的绑定关系

```
typedef struct {
    const char* name;//Java层native方法的名字
    const char* signature;//Java层native方法的描述符
    void*       fnPtr;//对应JNI函数的指针
} JNINativeMethod;
```

12、JNIEnv结构体的作用

> 1. 指向一个 函数表 ， 该 函数表 指向一系列 JNI函数
> 2. 通过这些 JNI函数 就能够实现 调用Java层的代码

```
//1. 获取到Java对象中某个变量的ID
jfieldID GetFieldID(jclass clazz, const char* name, const char* sig)
//2. 根据变量的ID获取到数据类型为boolean的变量
jboolean GetBooleanField(jobject obj, jfieldID fieldID)
//3. 获取到Java对象中对应方法的ID
jmethodID GetMethodID(jclass clazz, const char* name, const char* sig)
//4. 根据方法ID去调用对应对象中的方法，且该方法返回void
CallVoidMethod(jobject obj, jmethodID methodID, ...)
//5. 根据方法ID去调用对应对象中的方法，且该方法返回boolean
CallBooleanMethod(jobject obj, jmethodID methodID, ...)
```

13、JNI数据类型的作用

> 1. Java层和C/C++的数据类型和对象不能直接相互引用和使用。
> 2. C/C++的指针对于Java就是无法识别的。
> 3. 综上：JNI层定义了自己的数据类型以达到衔接的作用。

14、JNI中有哪几种数据类型?

> 1. JNI的数据类型分为两种：基本类型和引用类型

15、JNI的原始数据类型

Java Type	Native Typ	Description
boolean	jboolean	unsigned 8 bits
byte	jbyte	signed 8 bits
char	jchar	unsigned 16 bits
short	jshort	signed 16 bits
int	jint	signed 32 bits
long	jlong	signed 64 bits

Java Type	Native Typ	Description
float	jfloat	32 bits
double	jdouble	64 bits
void	void	N/A

16、JNI引用类型?

> 1. JNI定义了一些引用类型来便于 JNI层 调用

```
jobject                     (all Java objects)
|
|-- jclass                  (java.lang.Class objects)
|-- jstring                 (java.lang.String objects)
|-- jarray                  (array)
|     |--jobjectArray       (object arrays)
|     |--jbooleanArray      (boolean arrays)
|     |--jbyteArray         (byte arrays)
|     |--jcharArray         (char arrays)
|     |--jshortArray        (short arrays)
|     |--jintArray          (int arrays)
|     |--jlongArray         (long arrays)
|     |--jfloatArray        (float arrays)
|     |--jdoubleArray       (double arrays)
|
|--jthrowable
```

17、JNI中的方法ID和变量ID的作用?

> 1. 如果要在在JNI中去调用 Java层的某个方法 ， 首先需要获取它的ID，再根据ID通过JNI函数去获得该方法。

```
//变量ID
struct _jfieldID;
typedef struct _jfieldID *jfieldID;

//方法ID
struct _jmethodID;
typedef struct _jmethodID *jmethodID; /* method IDs */
```

18、JNI中类描述符的作用

> 1. 用于去获取Java的对象。
> 2. 例如 com.hao.jniapp.JniUtils2 这是该类所属于的包。需要更换为 com/hao/jniapp/JniUtils2 -这就是类的描述符(FindClass("com/hao/jniapp/JniUtils2") 中

> 常用)。
> 3.

19、方法描述符?

> 1. 就是用于确定 `native` 方法的参数和返回值。
> 2. “()V”中“V”表示返回值为空
> 3. “()V”中“()”标识为参数
> |Method Descriptor| Java Language Type|
> |---|---|
> |"()Ljava/lang/String;" |String f();
> |"(ILjava/lang/Class;)J" |long f(int i, Class c);
> |"([B)V" |String(byte[] bytes);

20、数据类型描述符

Field Desciptor	Java Language Type
Z	boolean
B	byte
C	char
S	short
I	int
J	long
F	floa
D	double

21、数组描述符

> 1. 一位数组的描述符是“[+对应的类型描述符”
> 2. 二维和三维数组，以"[["和“[[[”开头，更多维数组类似

Descriptor	Java Langauage Type
"[[I"	int[][]
"[[[D"	double[][][]

Field Desciptor	Java Language Type
"Ljava/lang/String;"	String

Field Desciptor	Java Language Type
"[Ljava/lang/Object;"	Object[]

22、JNI函数中调用Java的静态方法

> 1. Java层的native方法所在类编写一个一般方法
> 2. C/C++的文件中编写函数，参数需要有(JNIEnv *env, jobject thiz)，在内部去获得Java层方法并调用。
> 3. 在C/C++的JNI函数中去调用上者的方法。

```
//JniUtils2.Java
public class JniUtils2 {
    static {
        System.loadLibrary("native-lib");
    }
    ...
  //1. 静态方法
    public static void staticShowMsg(String msg){
        Log.i("feather", "I'm Java Static method: get JNI's msg =" + msg);
    }
}


//xxx.cpp
void callJavaMethod(JNIEnv *env, jobject thiz){
  //1. 获取到Java的class
    jclass clazz = env->FindClass("com/hao/jniapp/JniUtils2");
    if(clazz == NULL){
        printf("find class JniUtils2 error!");
        return;
    }
  //2. 获取到方法ID
    jmethodID  id = env->GetStaticMethodID(clazz, "staticShowMsg", "(Ljava/lang/String;)V");
    if(id == NULL){
        printf("find method staticShowMsg error!");
    }
  //3. 调用该静态方法并传入参数
    jstring msg = env->NewStringUTF("Hello Java! I'm JNI message");
    env->CallStaticVoidMethod(clazz, id, msg);
}
```

23、JNI函数中调用Java的非静态方法

> 重点是通过JNIEnv非静态方法去调用，并且需要将对应的Java对象的this作为参数传入。

```java
//java
public class JniUtils2 {
    static {
        System.loadLibrary("native-lib");
    }
    ...
    public void showMsg(String msg){
        Log.i("feather", "I'm Java method: get msg =" + msg + "from JNI");
    }
}
```

```
void callAnothorJavaMethod(JNIEnv *env, jobject thiz){
    jclass clazz = env->FindClass("com/hao/jniapp/JniUtils2");
    if(clazz == NULL){
        printf("find class JniUtils error!");
        return;
    }
  //1. 获取非静态方法的ID
    jmethodID  id = env->GetMethodID(clazz, "showMsg", "(Ljava/lang/String;)V");
    if(id == NULL){
        printf("find method showMsg error!");
    }
    jstring msg = env->NewStringUTF("Hello Java! I'm JNI message");
  //2. 重点是需要将代表JniUtils2对象的this指针(`jobject thiz`)
    env->CallVoidMethod(thiz, id, msg);
}
```

参考书籍

1. AS3.0的JNI和NDK开发教程
2. Android JNI基础
3. 如何在JNI中的C/C++层打印log到logcat
4. JNI的API文档