NDK 高级编程 (笔记)

deepwaterooo

2021 年 12 月 15 日

目录

1 深入了解 Android NDK

- 本文链接：http://gnaixx.cc/2017/07/23/20170723-ndk-pro/
- 创建项目时 Android.mk 文件的构建：Android.mk 配置参数
- ndk-build 脚本参数

```
#NDK 项目位置
ndk-build -C /path/to/the/project
# 强制重构所有代码
ndk-build -B
# 清除生成的二进制文件和目标文件
ndk-build clean
# 并行构建命令
ndk-build -j 4
```

2 用 JNI 实现与原生代码通信

jni 的开发基础知识，参考：

- NDK 开发 - JNI 开发流程
- NDK 开发 - JNI 数据类型与 Java 数据类型映射关系
- NDK 开发 - JNI 基本数据和字符串处理
- NDK 开发 - JNI 数组数据处理
- NDK 开发 - C/C++ 访问 Java 变量和方法
- NDK 开发-JNI 局部引用、全局引用和弱全局引用

3 日志、调试及故障处理

3.1 原生日志 API

```
//头文件
#include <android.h>
//Android.mk
LOCAL_LALIBS += -llog
_android_log_write(ANDROID_LOG_DEBUG, "hello-jni", "debug log.")
//多参数封装
static void logMessage(JNIEnv *env, jobject obj, const char *format, ...) {
    static jmethodID methodID = NULL;
    if (NULL == methodID) {
        jclass clazz = env->GetObjectClass(obj);
        methodID = env->GetMethodID(clazz, "logMessage", "(Ljava/lang/String;)V");
        env->DeleteLocalRef(clazz);
    }
    if(methodID != NULL) {
        char buffer[MAX_LOG_MSG_LENGTH];
        va_list ap;
        va_start(ap, format); //指向 format 后可变参数的地址
        vsnprintf(buffer, MAX_LOG_MSG_LENGTH, format, ap);
        va_end(ap);
        jstring message = env->NewStringUTF(buffer);
        if (message != NULL) {
            env->CallVoidMethod(obj, methodID, message);
            env->DeleteLocalRef(message);
        }
    }
}
```

3.2 重定向 Android 日志

```
adb shell stop
adb shell setprop log.redirect-stdio true
adb shell start
```

3.3 故障处理

3.3.1 堆栈分析

```
#ndk-stack
adb logcat | ndk-stack -sym obj/local/armeabi
#arm-linux-androideabi-addr2line
arm-linux-androideabi-addr2line -e obj/local/armeabi-v7a/libtongdun.so 0002197e
```

3.3.2 启用 **CheckJNI:**

```
adb shell setprop debug.checkjni 1
```

3.4 内存问题

3.4.1 打开 **libc** 调试模式

```
adb shell setprop libc.debug.malloc 1
adb shell stop
adb shell start
```

3.4.2 **strace** 工具

```
# 获取进程
adb shell ps | grep packagename
# 附加进程
adb shell strace -v -p <pid>
```

4 原生线程

- 源码地址: ndk-pro/threads
  - https://github.com/gnaixx/pro-ndk/tree/master/threads

4.1 **POSIX** 线程返回结果

//1. 线程句柄 2. 返回值指针 int pthread_join(pthread_t thread, void** ret_val);

4.2 **POSIX** 线程同步

```
static pthread_mutex_t mutex;
//初始化
int pthread_mutex_init(pthread_mutex_t* mutex, const pthread_mutexarrt_t* attr);
//锁定
int pthread_mutex_lock(pthread_mutex_t* mutex);
//解锁
int pthread_mutex_unlock(pthread_mutex_t* mutex);
//销毁
int pthread_mutex_destroy(pthread_mutex_t* mutex);
```

4.3 使用信号量同步 POSIX 线程

```
//头文件
#include <semaphone.h>
//初始化
extern int sem_init(sem_t* sem, int pshared, unsigned int value);
//锁定信号
extern int sem_wait(sem_t* sem);
//解锁
extern int sem_post(sem_t* sem);
//销毁
extern int sem_destroy(sem_t* sem);
```

5 POSIX Socket API

- TCP && UDP
- 源码地址: pro-ndk/echo
  - https://github.com/gnaixx/pro-ndk/tree/master/echo

6 支持 C++

6.1 支持的 C++ 运行库

- C++ 系统运行库不支持：C++ 标准库、异常支持库、RTTI 支持
- GAbi++、STLport、GUN STL

表 11-1 支持的 C++运行库比较

C++运行库	C++异常支持	C++ RTTI 支持	C++标准库
系统库	No	No	No
GAbi++	No	Yes	No
STLport	No	Yes	Yes
GNU STL	Yes	Yes	Yes

6.2 指定 C++ 运行库

```
//Application.mk
APP_STL := system //默认
APP_STL := gabi++_static
APP_STL := gabi++_shared
APP_STL := stlport_static
APP_STL := stlport_shared
APP_STL := gunstl_static
APP_STL := gunstl_shared
//1. 项目只有一个单一的原生模块时支持静态库
//2. 项目中包含多个原生模块时使用动态库
//3. 动态库使用时需要先加载
System.loadLibrary(“strport_shared”)
```

6.3 C++ 支持异常

```
//单个模块 Android.mk
LOCAL_CPP_FEATURES += exceptions
```

```
//支持所有原生模块 Application.mk
APP_CPPFLAGS += -fexceptions
```

6.4 C++ RTTI 支持（Run-Time Type Information）

- 在运行库展示对象类型信息，只要用于执行安全类型转化。

```
//单个模块 Android.mk
LOCAL_CPP_FEATURES += rtti
//支持所有原生模块 Application.mk
APP_CPPFLAGS += -frtti
```

6.5 C++ 标准库

- 容器
  - 序列
    - vector 支持随机访问，末尾常量插入删除 其他线性
    - deque 与 vector支持随机访问 选择实现队列的基础
    - list 双向列表
    - slist 单项链表
  - 关联容器
    - 排序关联容器 操作复杂度不超过对数阶
      - set 已排序，不重复
      - map 键值对，不重复
      - multiset 已排序，多重关联，允许重复
      - multimap 对拥有相同键值的元素数量不限制
    - 哈希关联容器 查询时间快
      - hashed_set 不重复
      - hash_map
      - hash_multiset 允许重复
      - hash_multimap 允许重复
  - 适配器
    - stack 堆栈 LIFO (Last In First Out)
    - queue 队列 FIFO (First In First Out)
  - String
- 迭代器

6.6 C++ 运行库调试模式

```
//GUN STL 调试模式
LOCAL_CFLAGS += -D_GLIBCXX_DEBUG
//STLport 调试模式
LOCAL_CFLAGS += -D_STLP_DEBUG
//日志重定向到 Android 日志
LOCAL_CFLAGS += -D_STLP_DEBUG
LOCAL_CFLAGS += -D_STLP_DEBUG_MESSAGE
LOCAL_LDLIBS += —llog
void __stl_debug_message(const char* format_str, …){
    va_list ap;
```

```
    va_start(ap, format_str);
    __android_log_vprint(ANDROID_LOG_FATAL, “STLport”, format_str, ap);
    va_end(ap);
}
```

7 原生图形 API

- 源码地址: ndk-pro/abiplayer
  - https://github.com/gnaixx/pro-ndk/tree/master/aviplayer

8 程序概要分析和 NEON 优化

- android-ndk-profiler
- NEON 指令：并不是所有基于 ARM-V7a 的设备都支持 NEON 指令