关于 Unity Native 插件的开发

作者：kyochow
原文链接：https://ld246.com/article/1602510440076
来源网站：链滴

构建native库的意义

1. 核心算法保密（例如随机算法，寻路算法等）
2. 效率考量(c++库效率比C#要高一个数量级)
3. 跨端需求(客户端，服务端)

目标

- 可以被用作Unity Plugin，供Unity Editor(Windows + Mac)使用
- 也可以编译为库文件，供移动端使用(Android + iOS)

准备工作

0. 本文以Mac系统为例
1. 安装 Homebrew
2. 安装CMake，一个跨平台的安装和编译工具，可以用简单的语句来描述所有平台的安装或编译过程
3. JDK SDK NDK这些可以使用Unity安装包内附带的
4. XCode
5. Visual Studio

native插件的基本要素

1. 全局头文件Define.h，定义API，为了兼顾做Unity-Plugin和普通dll库，里做一个宏定义判断(标准写法，全网通用)

```
#pragma once
// Unity native plugin API
// Compatible with C99

#if defined(__CYGWIN32__)
    #define API __declspec(dllexport) __stdcall
#elif defined(WIN32) || defined(_WIN32) || defined(__WIN32__) || defined(_WIN64) || defined(W
NAPI_FAMILY)
    #define API __declspec(dllexport) __stdcall
#elif defined(__MACH__) || defined(__ANDROID__) || defined(__linux__) || defined(__QNX__)
    #define API
#else
    #define API
#endif
```

2.具体暴露给Unity的方法如何定义？例如 Encrypt.h,标准写法如下

```
#include "Define.h"
#ifdef __cplusplus
extern "C"
```

原文链接：关于 Unity Native 插件的开发

```
{
#endif

    //Encrypt
    void API EncodeNoGC(char* params, int paramsLength);
    //Decrypt
    void API DecodeNoGC(char* params, int paramsLength);
    //The key
    char KEY[] = "abcdefg123456";
#ifdef __cplusplus
}
#endif
```

这样的代码到底是什么意思呢？首先，__cplusplus是cpp中的自定义宏，那么定义了这个宏的话表示是一段cpp的代码，也就是说，上面的代码的含义是:如果这是一段cpp的代码，那么加入extern "C"{和处理其中的代码。

要明白为何使用extern "C"，还得从cpp中对函数的重载处理开始说起。在c++中，为了支持重机制，在编译生成的汇编码中，要对函数的名字进行一些处理，加入比如函数的返回类型等等.而在C，只是简单的函数名字而已，不会加入其他的信息.也就是说:C++和C对产生的函数名字的处理是不一的.

3.接下来具体逻辑怎么写？例如 Encrypt.cpp,标准写法如下

```
#include "Encrypt.h"

#ifdef __cplusplus
extern "C"
{
#endif
    void API EncodeNoGC(char* params, int paramsLength)
    {
        //加密逻辑
    }

    void API DecodeNoGC(char* params, int paramsLength)
    {
        //解密逻辑
    }
#ifdef __cplusplus
}
#endif
```

4.这样一个最基本的native就写完里，下一步是编译，我们使用cmake，这里做cmake科普，一个CMakeLists.txt如下

```
CMAKE_MINIMUM_REQUIRED(VERSION 2.6)

PROJECT(kernal)

AUX_SOURCE_DIRECTORY(src/. SRC_LIST)
```

```
FILE(GLOB_RECURSE HEADER_LIST  src/*.h )
source_group("Header Files" FILES ${HEADER_LIST})

if ( WIN32 AND NOT CYGWIN AND NOT ( CMAKE_SYSTEM_NAME STREQUAL "WindowsStore
 ) )
    set(CMAKE_C_FLAGS_RELEASE "${CMAKE_C_FLAGS_RELEASE} /MT" CACHE STRING "")
    set(CMAKE_C_FLAGS_DEBUG "${CMAKE_C_FLAGS_DEBUG} /MTd" CACHE STRING "")
    set(CMAKE_CXX_FLAGS_RELEASE "${CMAKE_CXX_FLAGS_RELEASE} /MT" CACHE STRING "")
    set(CMAKE_CXX_FLAGS_DEBUG "${CMAKE_CXX_FLAGS_DEBUG} /MTd" CACHE STRING "")
    set(CompilerFlags
            CMAKE_CXX_FLAGS_DEBUG
            CMAKE_CXX_FLAGS_RELEASE
            CMAKE_C_FLAGS_DEBUG
            CMAKE_C_FLAGS_RELEASE
            )
    foreach(CompilerFlag ${CompilerFlags})
        string(REPLACE "/MD" "/MT" ${CompilerFlag} "${${CompilerFlag}}")
    endforeach()
endif ()

if (APPLE)
    if (IOS)
        set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -fembed-bitcode")
        set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} -fembed-bitcode")
        ADD_Library(kernal  ${HEADER_LIST} ${SRC_LIST} )
        set_xcode_property (kernal IPHONEOS_DEPLOYMENT_TARGET "7.0" "all")
    else ()
        ADD_Library(kernal  MODULE ${HEADER_LIST} ${SRC_LIST} )
        set_target_properties(kernal PROPERTIES BUNDLE TRUE)
    endif ()
elseif (ANDROID)
    ADD_Library(kernal SHARED ${HEADER_LIST} ${SRC_LIST} )
else ()
    ADD_Library(kernal MODULE ${HEADER_LIST} ${SRC_LIST} )
    set_target_properties(kernal PROPERTIES BUNDLE TRUE)
endif ()
```

这个cmake是包含各个平台的编译的，其中特别的

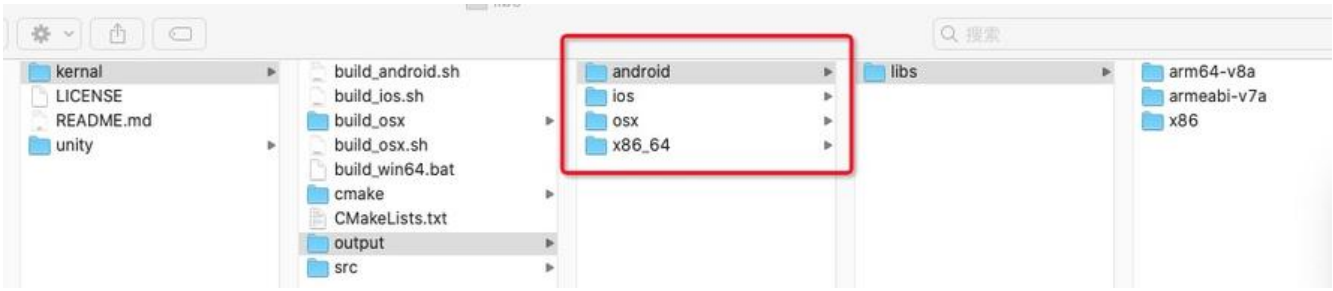
- Apple平台下，Mac出.boudle库
- Apple平台下，iOS出.a静态库
- Android 是要出SHARED包，也就是.so,静态库是不能用的
- Windows部分没什么特别(个别代码摘自xLua的编译文件)

5.有了CMakeLists.txt，各个平台执行shell或者bat包即可，仅举一个例子Android

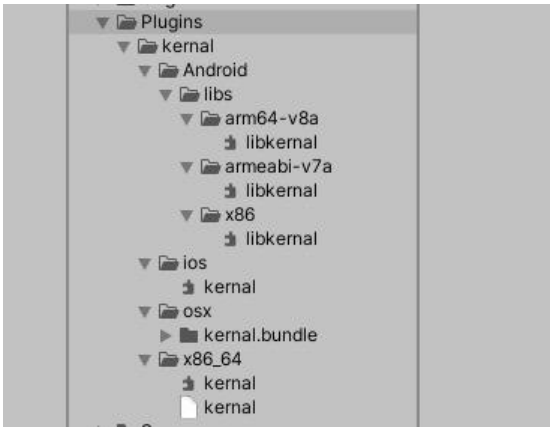
```
if [ -n "$ANDROID_NDK" ]; then
    export NDK=${ANDROID_NDK}
elif [ -n "$ANDROID_NDK_HOME" ]; then
    export NDK=${ANDROID_NDK_HOME}
elif [ -n "$ANDROID_NDK_HOME" ]; then
```

```
    export NDK=${ANDROID_NDK_HOME}
else
    export NDK=/Applications/Unity/Hub/Editor/2019.4.0f1/PlaybackEngines/AndroidPlayer/
DK
fi

if [ ! -d "$NDK" ]; then
    echo "Please set ANDROID_NDK environment to the root of NDK."
    exit 1
fi

function build() {
    API=$1
    ABI=$2
    TOOLCHAIN_ANME=$3
    BUILD_PATH=build_android_${ABI}
    cmake -H. -B${BUILD_PATH} -DANDROID_ABI=${ABI} -DCMAKE_TOOLCHAIN_FILE=${NDK}
build/cmake/android.toolchain.cmake -DANDROID_NATIVE_API_LEVEL=${API} -DANDROID_
OOLCHAIN=clang -DANDROID_TOOLCHAIN_NAME=${TOOLCHAIN_ANME}
    cmake --build ${BUILD_PATH} --config Release
    cp ${BUILD_PATH}/libkernal.so output/android/libs/${ABI}/libkernal.so
#    cp ${BUILD_PATH}/libkernal.so ../unity/Assets/Plugins/kernal/Android/libs/${ABI}/libkernal
so
}

build android-16 armeabi-v7a arm-linux-androideabi-4.9
build android-16 arm64-v8a  arm-linux-androideabi-clang
build android-16 x86 x86-4.9
```

最关键的先要指定NDK路径，然后就是按部就班的执行build了,所有的脚本执行完，得到如下output果



接下来就是原封不动的扔到Unity的Plugins里去



原文链接：关于 Unity Native 插件的开发

6.在Unity中如何调用呢，废话不多说，直接上代码Encrypt.cs

```
public class Encrypt
{
#if UNITY_IOS && !UNITY_EDITOR
    const string ENCRYPT_DLL = "__Internal";
#else
    const string ENCRYPT_DLL = "kernal";
#endif
    [DllImport(ENCRYPT_DLL, EntryPoint = "EncodeNoGC", CallingConvention = CallingConvent
on.Cdecl)]
    public static extern void EncodeNoGC(byte[] aData, int aLength);

    [DllImport(ENCRYPT_DLL, EntryPoint = "DecodeNoGC", CallingConvention = CallingConven
ion.Cdecl)]
    public static extern void DecodeNoGC(byte[] aData, int aLength);
}
```

注意这里的CallingConvention必须是CallingConvention.Cdecl，Standard调用在移动端是不被支的

Demo

上面的代码仅仅是核心代码，一些include没有放进来，完整Demo可以看如下示例

https://github.com/kyochow/xor_unity_native.git