一、前言

我们在写应用程序的过程中，经常需要面对一个开发场景：编写跨平台的应用程序。

这种要求对于 Linux 系列的平台来说，还是比较好处理的，大部分情况下只需要换一个交叉编译工具链即可，涉及到硬件平台相关部分再嵌入几个内联汇编。

但是，对于 Windows 平台来说，就稍微麻烦一些。你可能会说，在 Windows 平台上用 cygwin, minGW 也可以统一编译啊，但是你能指望客户在安装你的程序时，还需要去部署兼容 Linux 的环境吗？最好的解决方式，还是使用微软自家的开发环境，比如VS等等。

这篇文章，我们以一个最简单的程序，来描述如何使用 cmake 这个构建工具，来组织一个跨平台的应用程序框架。

阅读这篇文章，您可以收获下面几个知识点：

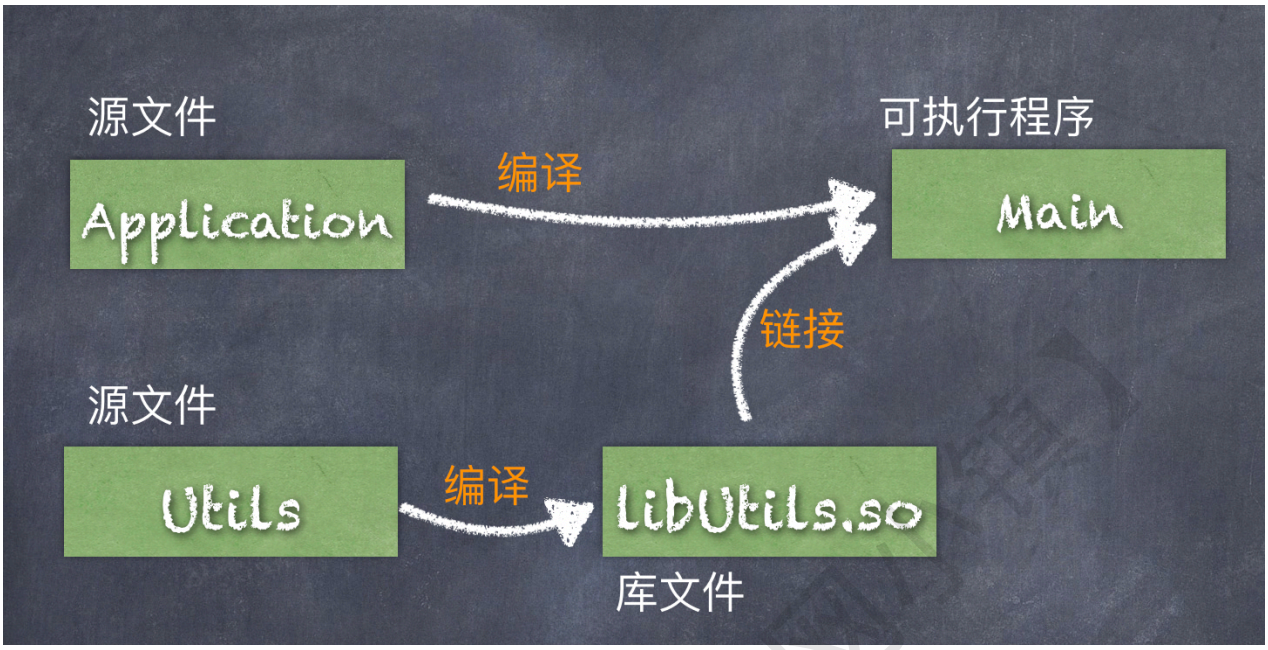
> 1. cmake 在编译库文件、应用程序中的相关指令；
> 2. Windows 系统中的动态库导出、导入写法；
> 3. 如何利用宏定义来进行跨平台编程；

在公众号后台留言【430】，可以收到示例代码。在 Linux/Windows 系统中可以直接编译、执行，拿来即用。

二、示例代码说明

1. 功能描述

示例代码的主要目的，是用来描述如何组织一个跨平台的应用程序结构。它的功能比较简单，如下图所示：



2. 文件结构

```
.
├── Application
│   ├── include
│   ├── lib
│   └── src
│       ├── CMakeLists.txt
│       ├── main.c
│       ├── Makefile
│       ├── task1.c
│       └── task1.h
├── clearall.sh
├── CMakeLists.txt
├── Common
│   ├── include
│   │   ├── Platform.h
│   │   └── Types.h
│   └── lib
│       ├── linux
│       └── win32
└── Utils
    ├── install
    └── src
        ├── CMakeLists.txt
        ├── Makefile
        ├── Utils.c
        ├── UtilsDll.h
        └── Utils.h

12 directories, 14 files
```

1. Common：放置一些开源的第三方库，例如：网络处理，json 格式解析等等;
2. Application: 应用程序，使用 Utils生成的库;
3. Uitls：放置一些工具、助手函数，例如：文件处理、字符串处理、平台相关的助手函数等等，最后会编译得到库文件（动态库 libUtils.so、静态库 libUtils.a）；
4. 如果扩展其他模块，可以按照 Utils 的文件结构复制一个即可。

3. cmake 构建步骤

在示例代码根目录下，有一个“总领” CMakeLists.txt 文件，主要用来设置编译器、编译选项，然后去 include 其他文件夹中的 CMakeLists.txt 文件，如下：

CMakeLists.txt

```
CMAKE_MINIMUM_REQUIRED(VERSION 3.0)
PROJECT(DemoApp VERSION 0.1)

if (UNIX)

# 交叉编译
#set(TOOLCHAIN_PREFIX "arm-linux-gnueabihf-")

# 编译器
set(CMAKE_C_COMPILER "${TOOLCHAIN_PREFIX}gcc")
set(CMAKE_CXX_COMPILER "${TOOLCHAIN_PREFIX}g++")
# 编译选项
set(CMAKE_CXX_FLAGS "-g -Wall -fPIC -O2")

elseif (WIN32)

set(CMAKE_C_FLAGS_DEBUG "${CMAKE_C_FLAGS_DEBUG} /MTd")
set(CMAKE_CXX_FLAGS_DEBUG "${CMAKE_CXX_FLAGS_DEBUG} /MTd")
set(CMAKE_C_FLAGS_RELEASE "${CMAKE_C_FLAGS_RELEASE} /MT")
set(CMAKE_CXX_FLAGS_RELEASE "${CMAKE_CXX_FLAGS_RELEASE} /MT")

endif()

ADD_SUBDIRECTORY(Utils/src)
ADD_SUBDIRECTORY(Application/src)
```

4. Utils 目录说明

这个目录的编译输出是库文件：

> Linux 系统：libUtils.so, libUtils.a;
>
> Windows 系统：libUtils.dll, libUtils.lib, libUtils.a;

其中的 CMakeLists.txt 文件内容如下：

```
#ifndef _UTILS_DLL_H_
#define _UTILS_DLL_H_

#if defined(linux) || defined(__linux) || defined(__linux__)
    #define UTILS_API
#else
    #ifdef UTILS_STATIC
        #define UTILS_API
    #else
        #ifdef UTILS_API_EXPORTS
            #define UTILS_API __declspec(dllexport)
        #else
            #define UTILS_API __declspec(dllimport)
        #endif
    #endif
#endif

#endif
```

目前，代码中只写了一个最简单的函数 getSystemTimestamp()，在可执行应用程序中，将会调用这个函数。

5. Application 目录说明

这个目录的编译输出是：一个可执行程序，其中调用了 libUtils 库中的函数。

CMakeLists.txt 文件内容如下：

```
# 源文件
AUX_SOURCE_DIRECTORY(. SRC)

# 头文件目录
INCLUDE_DIRECTORIES(${CMAKE_CURRENT_SOURCE_DIR})
INCLUDE_DIRECTORIES(${CMAKE_CURRENT_SOURCE_DIR}/../include)
INCLUDE_DIRECTORIES(${CMAKE_CURRENT_SOURCE_DIR}/../../Common/include)

# 库文件目录
if (UNIX)
LINK_DIRECTORIES(${CMAKE_CURRENT_SOURCE_DIR}/../lib/linux)
else()
LINK_DIRECTORIES(${CMAKE_CURRENT_SOURCE_DIR}/../lib/win32)
endif ()

# 生成可执行文件
ADD_EXECUTABLE(main ${SRC})

if (UNIX)
TARGET_LINK_LIBRARIES(main libUtils.so)
else()
TARGET_LINK_LIBRARIES(main libUtils.lib)
endif()
```

三、Linux 系统下操作步骤

1. 创建构建目录 build

```
$ mkdir build
```

在一个独立的 build 目录中编译，生成的中间代码不会污染源代码，这样对于使用 git 等版本管控工具来说非常的方便，在提交的时候只需要 ignore build 目录即可，强烈推荐按照这样的方式来处理。

2. 执行 cmake，生成 Makefile

```
$ cd build
$ cmake ..
```

```
-- The C compiler identification is GNU 5.4.0
-- The CXX compiler identification is GNU 5.4.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
```

3. 编译 Utils 库

```
$ cd Utils/src
$ make
```

```
Scanning dependencies of target libUtils_shared
[ 25%] Building C object Utils/src/CMakeFiles/libUtils_shared.dir/Utils.c.o
[ 50%] Linking C shared library libUtils.so
[ 50%] Built target libUtils_shared
Scanning dependencies of target libUtils_static
[ 75%] Building C object Utils/src/CMakeFiles/libUtils_static.dir/Utils.c.o
[100%] Linking C static library libUtils.a
[100%] Built target libUtils_static
```

在 CMakeLists.txt 中的最后部分是安装指令，把产生的库文件和头文件，安装到源码中的 install 目录下。

```
$ make install
```

```
c_app_framework/Utils/install/include/Utils.h
c_app_framework/Utils/install/include/UtilsDll.h
c_app_framework/Utils/install/lib/linux/libUtils.so
c_app_framework/Utils/install/lib/linux/libUtils.a
```

4. 编译可执行程序 Application

Application 使用到了 libUtils.so 库，因此需要手动把 libUtils.so 和头文件，复制到 Application 下面对应的 lib/linux 和 include 目录下。

当然，也可以把这个操作写在 Utils 的安装命令里。

```
$ cd build/Application/src
$ make
```

```
Scanning dependencies of target main
[ 33%] Building C object Application/src/CMakeFiles/main.dir/main.c.o
[ 66%] Building C object Application/src/CMakeFiles/main.dir/task1.c.o
[100%] Linking C executable main
[100%] Built target main
```

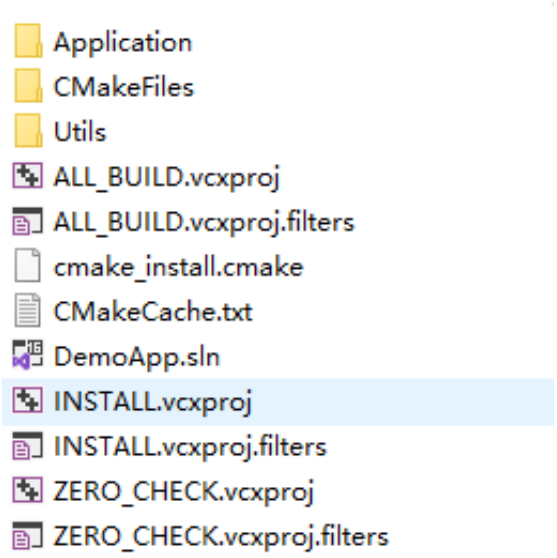
执行生成的可执行程序 main，即可看到输出结果。

四、Widnows 系统下操作步骤

1. 通过 cmake 指令生成 VS 工程

同样的道理，新建一个 build 目录，然后在其中执行 `cmake ..` 指令，生成 VS 解决方案，我使用的是 VS2019:

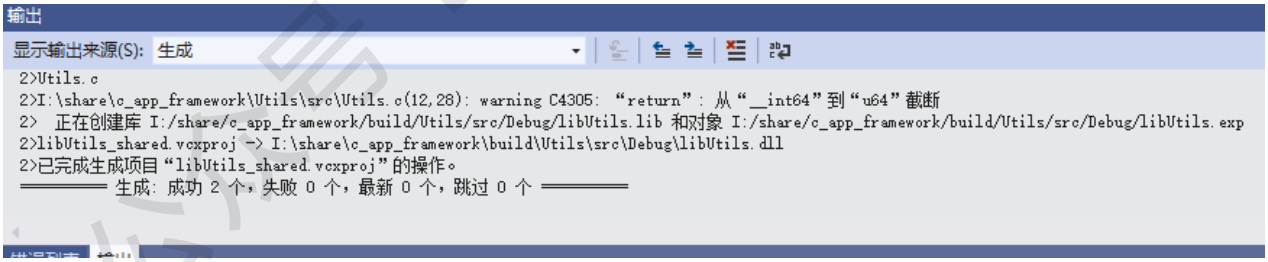
```
I:\share\c_app_framework\build>cmake ..
-- Building for: Visual Studio 16 2019
-- Selecting Windows SDK version 10.0.19041.0 to target Windows 10.0.18363.
-- The C compiler identification is MSVC 19.28.29337.0
-- The CXX compiler identification is MSVC 19.28.29337.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: C:/Program Files (x86)/Microsoft Visual Studio/2019
/Professional/VC/Tools/MSVC/14.28.29333/bin/Hostx64/x64/cl.exe - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: C:/Program Files (x86)/Microsoft Visual Studio/20
19/Professional/VC/Tools/MSVC/14.28.29333/bin/Hostx64/x64/cl.exe - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: I:/share/c_app_framework/build
```



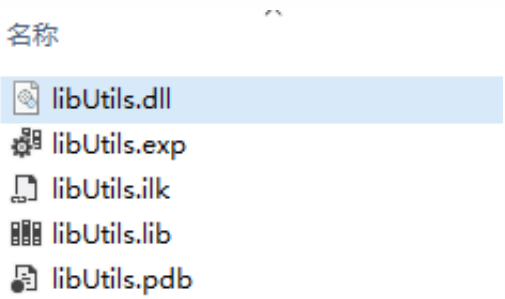
2. 编译 Utils 库文件

使用 VS2019 打开工程文件 DemoApp.sln，在右侧的解决方案中，可以看到：

在 libUtils_shared 单击右键，选择【生成】：



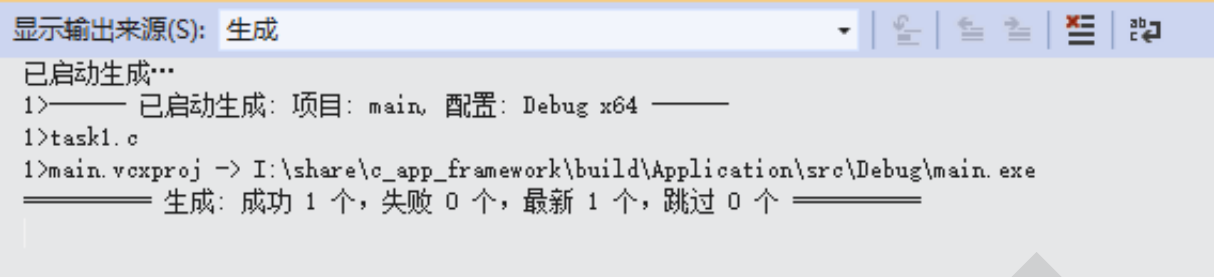
此时，在目录 build\Utils\src\Debug 下面，可以看到生成的文件：



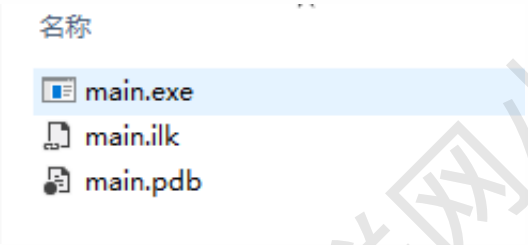
3. 编译可执行程序 Application

因为Application需要使用 Utils 生成的库，因此，需要手动把库和头文件复制到 Application 下面的 lib/win32 和 include 目录下。

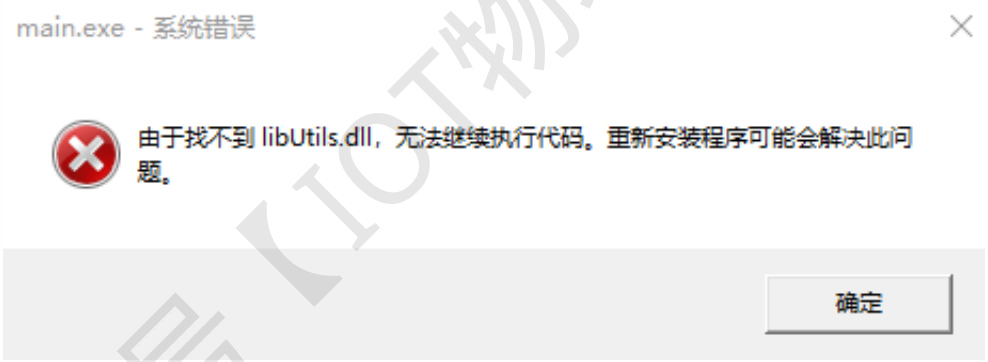
在 VS 解决方案窗口中，在 main 目标上，单击右键，选择【生成】：



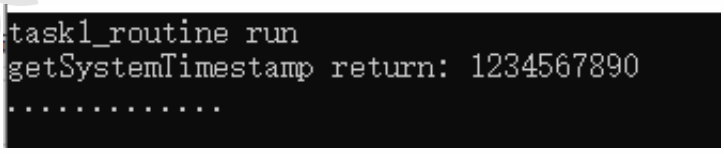
此时，在目录 build\Application\src\Debug 下可以看到生成的可执行程序:



直接单击 main.exe 执行，报错：



需要把 libUtils.dll 动态库文件复制到 main.exe 所在的目录下，然后再执行，即可成功。



五、总结

这篇文章的操作过程主要以动态库为主，如果编译、使用静态库，执行过程是一样一样的。

如果操作过程有什么问题，欢迎留言、讨论，谢谢！

在公众号后台留言【430】，可以收到示例代码。在 Linux/Windows 系统中可以直接编译、执行，拿来即用。

公众号【IOT物联网小镇】

祝您好运！

---------- End ----------

让知识流动起来，越分享，越幸运！

星标公众号，能更快找到我！

Hi~你好，我是道哥，一枚嵌入式开发老兵。

推荐阅读

1. C语言指针-从底层原理到花式技巧，用图文和代码帮你讲解透彻
2. 原来gdb的底层调试原理这么简单
3. 一步步分析-如何用C实现面向对象编程
4. 图文分析：如何利用Google的protobuf，来思考、设计、实现自己的RPC框架
5. 都说软件架构要分层、分模块，具体应该怎么做(一)
6. 都说软件架构要分层、分模块，具体应该怎么做(二)