

# Windows CMake C++ LIB DLL

The background image is a photograph of a city street corner. On the left is a modern building with a dark, grid-like facade and large windows. On the right is a multi-story brick building with arched windows. A street sign on the brick building indicates '567 NE' and 'ONE WAY' with an arrow pointing right. The text 'Windows CMake C++ LIB DLL' is superimposed in the center in a white serif font.

# 使用 CMake 来生成 C++ 下需要的链接库 LIB 和 DLL

- 开发环境

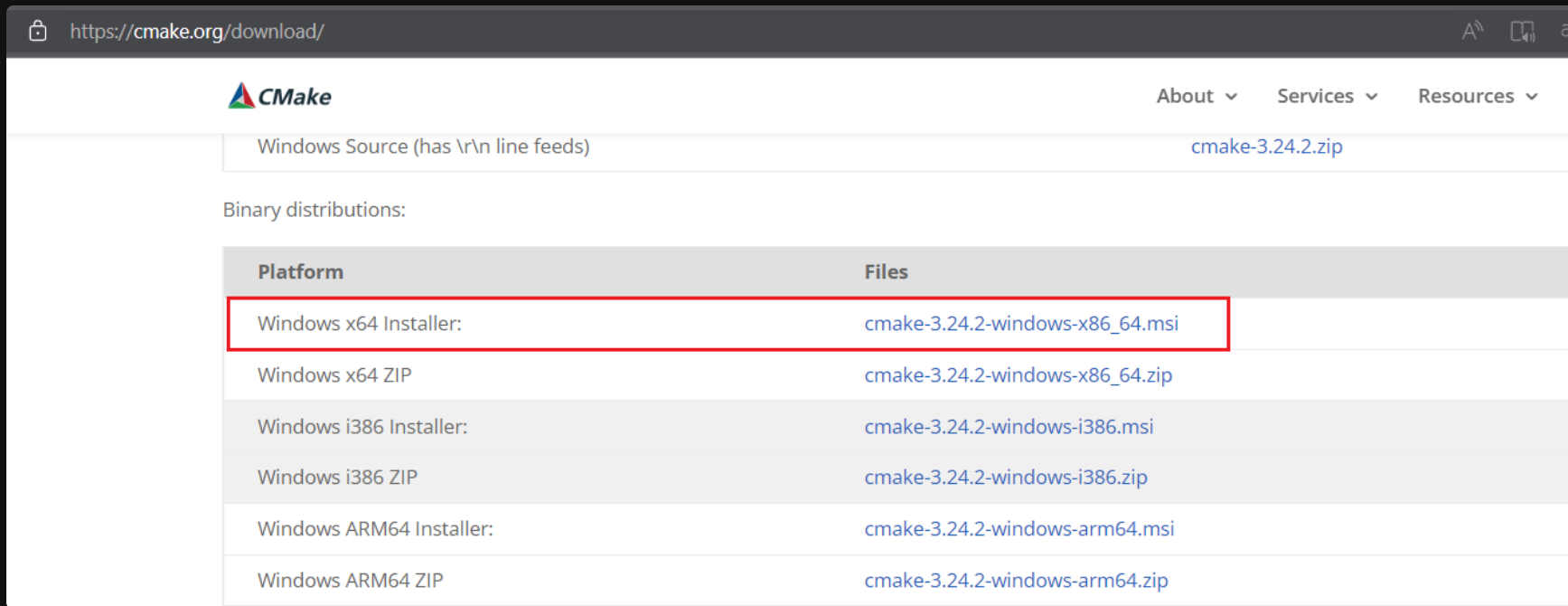
- 步骤

1. 编写对应的测试案例（函数、类）；
2. 导出包含案例代码的静态库 LIB 和动态库 DLL；
3. 编写测试主函数代码，引入这些外部库进行测试；

# 开发环境

1. 系统: Windows 10、Visual Studio 15 (msvc 14.0 编译器)

2. Windows 版本的 CMake



The screenshot shows the CMake website's download page. At the top, there's a navigation bar with the CMake logo and links for 'About', 'Services', and 'Resources'. Below this, a section for 'Windows Source (has \r\n line feeds)' includes a link to 'cmake-3.24.2.zip'. The main content area is titled 'Binary distributions:' and contains a table with two columns: 'Platform' and 'Files'. The first row of the table is highlighted with a red border and contains 'Windows x64 Installer:' and 'cmake-3.24.2-windows-x86\_64.msi'. Other rows list 'Windows x64 ZIP', 'Windows i386 Installer:', 'Windows i386 ZIP', 'Windows ARM64 Installer:', and 'Windows ARM64 ZIP' with their respective file names.

Platform	Files
Windows x64 Installer:	<a href="#">cmake-3.24.2-windows-x86_64.msi</a>
Windows x64 ZIP	<a href="#">cmake-3.24.2-windows-x86_64.zip</a>
Windows i386 Installer:	<a href="#">cmake-3.24.2-windows-i386.msi</a>
Windows i386 ZIP	<a href="#">cmake-3.24.2-windows-i386.zip</a>
Windows ARM64 Installer:	<a href="#">cmake-3.24.2-windows-arm64.msi</a>
Windows ARM64 ZIP	<a href="#">cmake-3.24.2-windows-arm64.zip</a>

# 开发环境

## 3. VSCode 及相关插件:

- CMake (必需, 语法提示)
- CMake Tools (必需, VSCode 下对 CMake 的操作工具)
- C/C++ (必需)
- Prettier (非必需, 格式化代码用)
- Git Graph (非必需, 看工程进展用)
- Vim (非必需, 一种编辑操作)

# 步骤

## 编写对应的测试案例（函数、类）

所有代码实现均在文件夹 `projects` 中

**cipher** 文件夹存放着一个类的实现

前期在这个项目文件夹下做功能性的测试

# 步骤

## 导出包含案例代码的静态库 LIB 和动态库 DLL

项目文件名为 **createlibrary**

可以利用 VSCode 中的 CMake Tools 插件，在 VSCode 中键入 `Ctrl + Shift + p` 这个快捷组合键打开 VSCode 的命令操作框；

键入 CMake，看到 `CMake: Quick Start` 的选项，选择它，选择编译器 kit，这里选择 `VisualStudio.14.0 - x86_amd64`；

之后输入你的 CMake project 的名称，会进入一个模式选择（生成库文件或者是生成执行文件），这里选择生成库文件（Create a library），CMake Tools 会自动生成可用于导出库文件的 CMakeLists.txt，同时自动执行了 `cmake -S . -B build` 这一条指令，会生成 build 文件夹，里面包含基于所选编译器以及所在开发平台（Windows）所构建的工程文件。

# 步骤

## 导出包含案例代码的静态库 LIB 和动态库 DLL

关键的是 `add_library` 这个函数，默认接受两个参数（目标导出文件名，库实现代码文件名），这里接受三个参数，中间的为导出模式选项，默认不指定为静态库，这里明确指定 `STATIC` 为静态库，若为 `SHARED` 则为动态库导出。将上一个步骤中的头文件和cpp文件放入该项目文件中，在 VSCode 最下方的状态栏选中 `build` 来执行，导出相关的库文件。

```
cmake_minimum_required(VERSION 3.0.0)
project(cipher VERSION 0.1.0)

include(CTest)
enable_testing()

add_library(hello STATIC hello.cpp)
add_library(cipher STATIC cipher.cpp)
# add_library(hello SHARED hello.cpp)
# add_library(cipher SHARED cipher.cpp)

set(CPACK_PROJECT_NAME ${PROJECT_NAME})
set(CPACK_PROJECT_VERSION ${PROJECT_VERSION})
include(CPack)
```

# 步骤

## 编写测试主函数代码，引入这些外部库进行测试

uselibrary 和 usedll 项目的生成和上一步骤基本一样，只是改为选择生成执行文件即可，CMake Tools 会自动生成对应的 CMakeLists.txt 项目配置文件。使用动态库和静态库的写法一样，只是要注意 `.dll` 文件要放在执行文件的同目录下。写法还与项目目录结构有关，可根据实际情况调整。

```
<!-- usedll -->
cmake_minimum_required(VERSION 3.0.0)
project(cipher_test VERSION 0.1.0)

include(CTest)
enable_testing()

include_directories("lib")
link_directories("lib")
add_executable(cipher_test main.cpp)
target_link_libraries(cipher_test hello cipher)

set(CPACK_PROJECT_NAME ${PROJECT_NAME})
set(CPACK_PROJECT_VERSION ${PROJECT_VERSION})
include(CPack)
```

```
<!-- uselibrary -->
cmake_minimum_required(VERSION 3.0.0)
project(cipher_test VERSION 0.1.0)

include(CTest)
enable_testing()

include_directories("lib")
link_directories("lib")
add_executable(cipher_test main.cpp)
target_link_libraries(cipher_test hello cipher)

set(CPACK_PROJECT_NAME ${PROJECT_NAME})
set(CPACK_PROJECT_VERSION ${PROJECT_VERSION})
include(CPack)
```



# 参考链接

[CMake's add\\_library – Creating Libraries With CMake](#)

[Step 2: Adding a Library](#)