1. # 第六集：在项目中使用外部共享库和头文件

在本集中，我们将学习如何在CMake项目中引用并使用外部共享库和头文件。我们将继续使用前集创建的 `libhello` 共享库，并编写一个程序来调用这个库中的函数。

准备工作

1. 在 `episode6` 目录下构建用于存放本节课的所有资源。
2. 在 `episode6` 目录下创建 `src` 目录，并在其中创建 `main.cc` 文件，内容如下：

```
#include <hello.hpp>
int main() {
    HelloFunc();
    return 0;
}
```

配置环境变量

为了让CMake能够找到库和头文件，需要设置以下环境变量：

- `CMAKE_INCLUDE_PATH`：指定额外的头文件搜索路径。
- `CMAKE_LIBRARY_PATH`：指定额外的库文件搜索路径。

例如，你可以在命令行中设置：

```
export CMAKE_INCLUDE_PATH=/usr/local/include/hello
export CMAKE_LIBRARY_PATH=/usr/local/lib
```

MAC系统

```
export DYLD_LIBRARY_PATH=/usr/local/lib:$DYLD_LIBRARY_PATH
```

Ubuntu系统

```
export LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH
```

编写CMake配置文件

在 src 目录下创建 CMakeLists.txt 文件，并配置如何构建程序：

```
cmake_minimum_required(VERSION 3.10)
project(UseHello)
add_executable(main main.c)
find_library(HELLO_LIBRARY NAMES hello PATHS
${CMAKE_LIBRARY_PATH})
if(HELLO_LIBRARY)
    target_link_libraries(main ${HELLO_LIBRARY})
endif()
```

构建程序

1. 在 目录下创建一个 build 目录，并从该目录运行 cmake .. 和 make`来构建项目。
2. 如果构建成功，执行生成的可执行文件应输出 Hello World ，表示程序正确链接到了 libhello 库。

处理潜在的错误

- 如果 cmake 或 make 过程中遇到找不到 hello.h 的错误，确保 CMAKE_INCLUDE_PATH 正确设置，并且 include_directories 指令已经正确添加到 CMakeLists.txt 中。
- 如果链接过程中遇到找不到库的错误，确保 CMAKE_LIBRARY_PATH 正确设置，并且 find_library 正确使用。

总结

通过本集的学习，我们了解了如何在CMake项目中配置和使用外部共享库和头文件。这不仅提高了项目的模块化和可重用性，还加深了对CMake工具链配置的理解。

作业

1. 试着将 libhello 库的不同版本放在不同的目录，通过修改环境变量来构建 main 程序，以便使用不同版本的库。
2. 如何链接静态库?