**测试环境：Windows 10（Ubuntu）+MinGw（gcc/g++/Ninja.exe）+ Visual Studio 2019 +Clion（克利翁）**

**插图以及内容来自书籍《Modern CMake For C++》**

**辅助工具：ChatGPT**

**CMake入门03-第一个基于CMake的项目：**

本节主要的内容：

* 基本的指令和命令；
* 如何划分项目；
* 考虑合理的项目结构；
* 如何禁用内部构建；

讲解原则：我们在demo里面涉及到一个新的命令，才展开讲解；

**构建相关的最重要的两个命令：**

表格

描述已自动生成

其中-S指定要构建的源码文件路径；

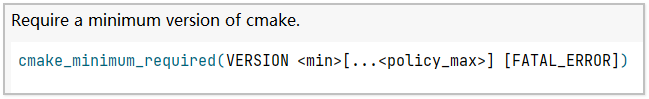
其中-B指定要生成的Build Tree路径；

一个CMakelists.txt的基本结构：

图形用户界面, 文本, 应用程序, 聊天或短信

描述已自动生成

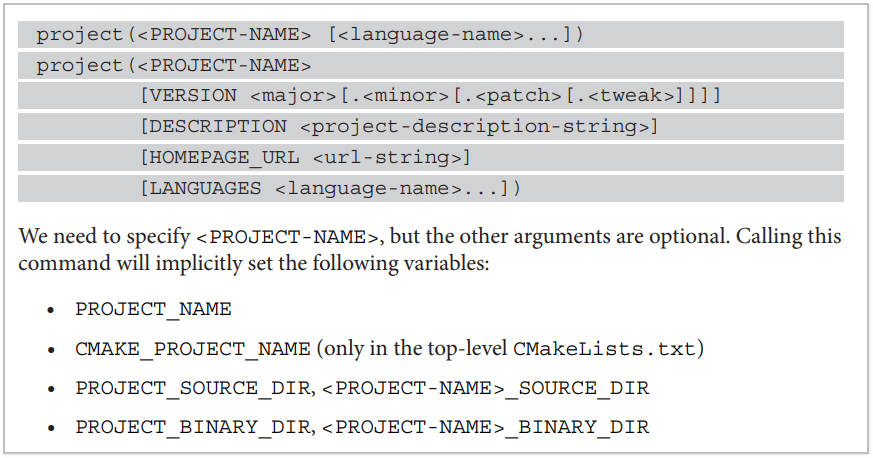
cmake\_minimum\_required命令的作用：



指定cmake需要的最低版本；

* 将检查系统是否具有正确的 CMake 版本，
* 它还将调用另一个命令 cmake\_policy（VERSION），它将告诉 CMake 用于此项目的正确策略。
* 可用于project 和脚本中添加最小版本要求；

project命令的作用： （非必须）



设置项目的名称；

在CMakeLists.txt file 文件下，CMake 将该命令隐式添加到文件顶部；

CMake 默认启用 C 和 C++，因此您可能希望仅为C++项目显式指定 CXX。

如果指定CXX，编译器将允许您在配置阶段跳过对未使用语言的任何检查来节省时间。

add\_executable命令的作用：

图形用户界面, 文本, 应用程序

描述已自动生成

添加一个Target也就是一个可执行文件；

图形用户界面, 文本, 应用程序

描述已自动生成

打开c++标准：



在 CMake 中，它的作用是告诉 CMake 编译器必须使用 C++ 标准；

对项目Project分区：

随着我们的解决方案拥有的行数和文件数量的增加，

我们慢慢明白不可避免的事情即将到来：

要么我们开始对项目进行分区，要么淹没在代码行和大量文件中。

我们可以通过两种方式解决此问题：

1. 同一个源文件，划分不同的模块和函数；
2. 把相关的代码放入子目录下；

假设一个项目的工程文件目录是这样的：

表格

描述已自动生成

方式① 直接使用路径；

图形用户界面, 文本, 应用程序

描述已自动生成

缺点：（关注点没有分离）

如果cars下多个源文件，修改后，Top目录下的CMakeLists.txt就要跟着修改，这显然违背了设计原则；

方式②，使用include命令；

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

Include命令：

图形用户界面, 文本, 应用程序

中度可信度描述已自动生成

在 CMake 中，**include** 命令用于将指定的外部 CMake 脚本文件引入当前的 CMakeLists.txt 文件中。

**CMake 会有效地将 cars\_sources 设置在与 add\_executable 相同的范围内（规则一）**

缺点：

1. 根据规则一，嵌套目录中的变量会污染顶级作用域（反之亦然）：
2. 所有子目录的编译环境都是继承自顶级目录；
3. 任何配置的修改子目录的文件都需要全部重新编译；
4. 所有路径都是相对于顶层路径。（非常重要，测试的时候说明）

方式③，使用add\_subdirectory命令；

新的文件结构：

图形用户界面

低可信度描述已自动生成

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

add\_subdirectory的命令：

图形用户界面, 文本, 应用程序, 聊天或短信

描述已自动生成

**add\_subdirectory** 是 CMake 中的一个命令，用于向当前项目添加一个子目录，并在该子目录中执行一个新的 CMakeLists.txt 文件

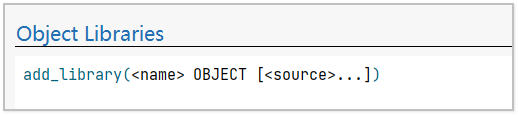
注意当使用add\_subdirectory的时候，该目录下一定要有一个CMakeLists.txt文件；

* **source\_dir**指定源文件CMakeLists.txt和代码文件所在的目录，即子目录的名称；
* **binary\_dir** 是可选的二进制目录名称或路径，用于指定生成的目标文件的输出目录。如果不指定该参数，则默认使用 **source\_dir** 目录作为输出目录

add\_library命令：

文本

中度可信度描述已自动生成



OBJECT：创建一个目标对象。源文件编译成目标文件，但不将其文件存档或链接成为库文件；

STATIC：类型的库是一组目标文件的压缩归档文件，可用于在编译链接过程中与其他目标文件一起使用，从而创建可执行程序。

SHARED：类型的库是动态链接库，它们可以在运行时加载到内存中。它们适用于创建插件或扩展已有软件的代码重复使用。

MODULE：类型的库与 SHARED 有些类似，但它们没有与其他目标链接。相反，它们使用类似于 dlopen 或 LoadLibrary 的功能在运行时动态加载。

CMake 创建的库类型取决于 BUILD\_SHARED\_LIBS 变量的值。如果该变量设置为 ON，则默认创建 SHARED 类型库。如果设置为 OFF，则默认创建 STATIC 类型库。

最后生成生成名称依赖于平台，可能如下：

静态库：such as lib<name>.a or <name>.lib)

动态库：such as lib<name>.so or <name>.dll)

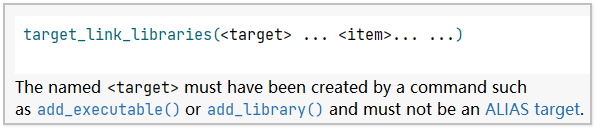
target\_include\_directories命令：

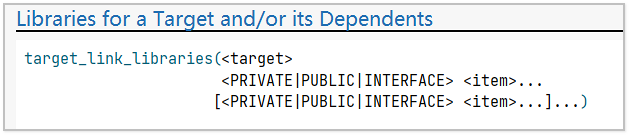
文本

描述已自动生成

**target\_include\_directories**用于向一个或多个目标添加头文件搜索路径。

target\_link\_libraries命令：





**target\_link\_libraries**用于为目标添加链接库或者链接其他目标。

文本, 信件

描述已自动生成



**我们在讲Target目标的时候详细讲这个知识点：**

我们开始使用库了吗？其实不然。由于我们使用了 OBJECT 关键字，

因此我们表示我们只对生成对象文件感兴趣（与上一个示例中完全相同）。

我们只是将它们分组在一个单一的逻辑目标（汽车）下

如您所见，我使用 add\_library（） 生成一个全局可见的目标 cars，并将 cars 目录添加到其公共包含目录中，

其中包含 target\_include\_ directory（）。

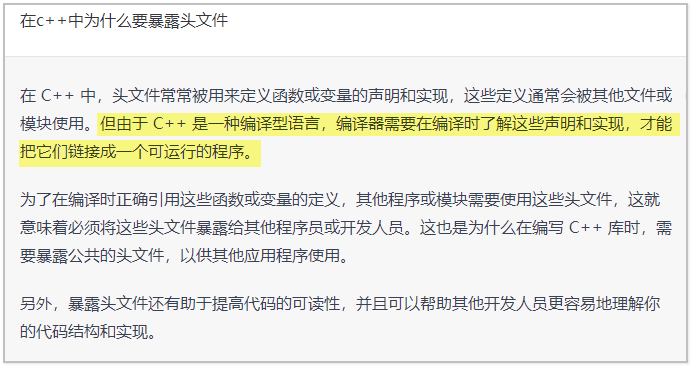
这允许 main.cpp 包含 cars.h 文件而不提供相对路径

//其他使用方式，引入头文件，直接用相对路径；

图形用户界面, 应用程序

描述已自动生成

了解一个关于C++头文件的知识点：



详细理解：

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

使用add\_subdirectory的优点：

• 变量更改隔离在子目录里面。

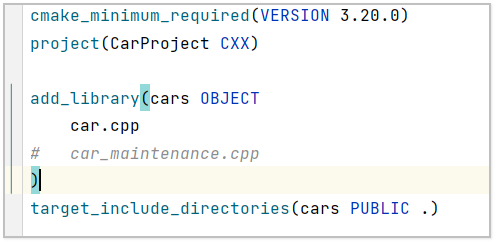
• 您可以随意配置子目录的编译配置。

• 更改子目录下的 CMakeLists.txt 文件不需要构建不相关的目标。

• 路径是目录的本地路径，如果需要，甚至可以将它们添加到父包含路径中

上面使用的是项目嵌套的方式完成了模块的划分；

使用外部项目的方式：



思考项目Project合理的结构：

* 它需要是方便浏览和易于扩展的；
* 它应该是独立的 - 例如，特定于项目的文件应该在项目目录中，而不是其他任何地方
* 抽象层次结构应通过可执行文件和二进制文件来表示

一个相对合理的工程结构：

图示

描述已自动生成

说明：

在此结构中，CMakeLists.txt 文件应存在于以下目录中：顶级项目目录、src、doc、extern 和test。

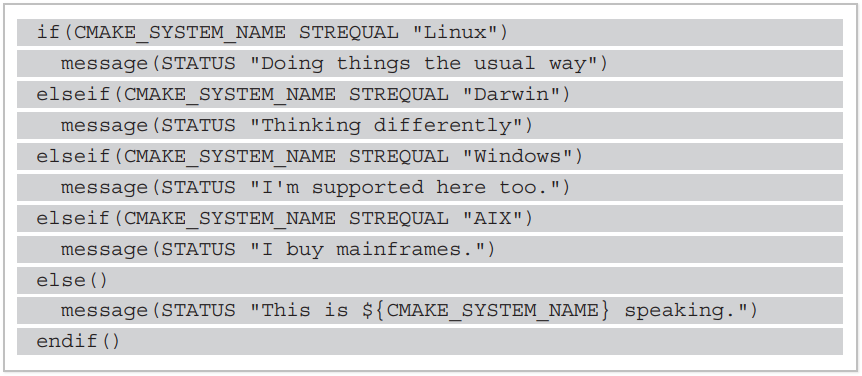
主列表文件不应该自己声明任何构建步骤，而是应该使用 add\_subdirectory() 命令来执行嵌套目录中的所有列表文件。

反过来，如果需要，这些可以将这项工作委托给更深的层

1. 执行从项目的根目录开始，即从驻留在源代码树中的列表文件开始。此文件将使用适当的策略设置所需的最低 CMake 版本，设置项目名称、支持的语言、全局变量，并包含 cmake 目录中的文件，以便其内容全局可用。
2. 下一步是通过调用 add\_subdirectory(src bin) 命令进入 src 目录的范围（我们希望将编译后的工程放在 <binary\_tree>/bin 而不是 <binary\_tree>/src 中）。
3. CMake 读取 src/CMakeLists.txt 文件并发现其唯一目的是添加四个嵌套子目录：app1、app2、lib1 和 lib2。
4. CMake进入app1的变量作用域，了解到另一个嵌套库lib3，它有自己的CMakeLists.txt文件；然后进入lib3的范围
5. lib3 库添加了一个同名的静态库目标。 CMake 返回到 app1 的父范围。
6. app1 子目录添加了一个依赖于 lib3 的可执行文件。 CMake 返回到 src 的父范围。
7. CMake 将继续进入剩余的嵌套范围并执行它们的列表文件，直到完成所有 add\_subdirectory() 调用。
8. CMake 返回到顶级范围并执行剩余的三个命令：add\_subdirectory(doc)、add\_subdirectory(extern) 和 add\_subdirectory(test)。每次，CMake 都会进入新范围并从适当的列表文件中执行命令
9. 收集所有目标并检查其正确性。 CMake 现在拥有生成构建系统所需的所有信息。

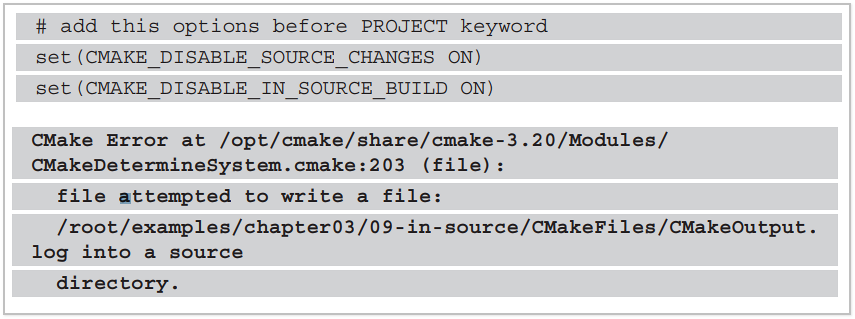
使用CMake构建的生成的静态库或动态库可以供同一构建环境下的其他目录或子目录中的代码使用。

确定操作系统：



禁止内部构建的两种方法

方法一：（不一定支持这两个参数）

：

方法二：

图形用户界面, 文本, 应用程序

描述已自动生成