# CMake 跨平台自动化构建 c++项目宝典

## 引言

### 课程介绍

#### 重要性

- 持续集成的方法和实践
- 为以后职业提升到项目经理和 CTO 的技能
- 大量开源系统所使用的构建工具
  - QT
  - opencv
  - googletest
  - KDE
  - OGRE
  - 安卓 NDK
  - 鸿蒙 ETS NDK
- 大部分公司 C++开发所使用的构建工具、一些使用 makefile，几十家企业的培训经验

## 课程收益

能够使用 cmake 构建跨平台程序和库（Windows、 Linux 、Mac）

能通过交叉编译构建安卓、鸿蒙、嵌入式 Linux 程序

熟悉 cmake 常用语法和常用函数

能够使用 cmake 配置自动化单元测试和部署

能够使用 cmake 构建综合的大项目

## 适合人群

有部分语言基础，想要学习 Linux 平台项目开发

想要学习跨平台构建方案

想要学习自动化单元测试方法的同学

公司需要需要使用 cmake 做项目构建

## 自我介绍

华为 HDE，给上百家企业做过 c++开发培训和咨询

### 自己的历程

本科毕业工作，研究生毕业，15 年创业，20 年程序员，跨平台使用两套项目配置，十年前开始做企业培训和咨询，企业构建从 makefile 到 cmake 大家都知道，原来也很乱，现在用 cmake 之后，自动化构建，单元测试一直没有引用，在 cmake 之后引入了

# 第一章 CMake 快速入门-执行程序和动态库的构建

## 1.1 cmake 基本概念

### 一 是什么

- CMake 是用于构建、测试和软件打包的开源跨平台工具

### 二 为什么选用 cmake

- 为什么我需要一个好的构建系统
    - 你想避免硬编码路径
    - 您需要在多台计算机上构建一个包
    - 你想使用 CI（持续集成）
    - 你需要支持不同的操作系统
    - 你想支持多个编译器
    - 您想使用 IDE，但不是所有情况
    - 你想描述你的程序的逻辑结构，而不是标志和命令
    - 你想使用库
    - 您想使用其他工具来帮助您编写代码 moc　ProtoBuf
    - 你想使用单元测试

- 持续集成
    - 每次集成都通过自动化的制造（包括提交、发布、自动化测试）来验证，准确地发现集成错误。
    - 快速错误，每完成一点更新，就集成到主干，可以快速发现错误，定位错误也比较容易
    - 各种不同的更新主干，如果不经常集成，会导致集成的成本变大
    - 让产品可以快速地通过，同时保持关键测试合格
    - 自动化测试，只要有一个测试用例不通过就不能集成
    - 集成并不能删除发现的错误，而是让它们很容易和改正
- 为什么是 cmake
    - cmake 特性
        - 自动搜索可能需要的程序、库和头文件的能力
        - 独立的构建目录，可以安全清理
        - 创建复杂的自定义命令
            - 例如 qt moc uic
        - 配置时选择可选组件的能力
        - 从简单的文本文件（CMakeLists.txt）自动生成工作区和项目的能力

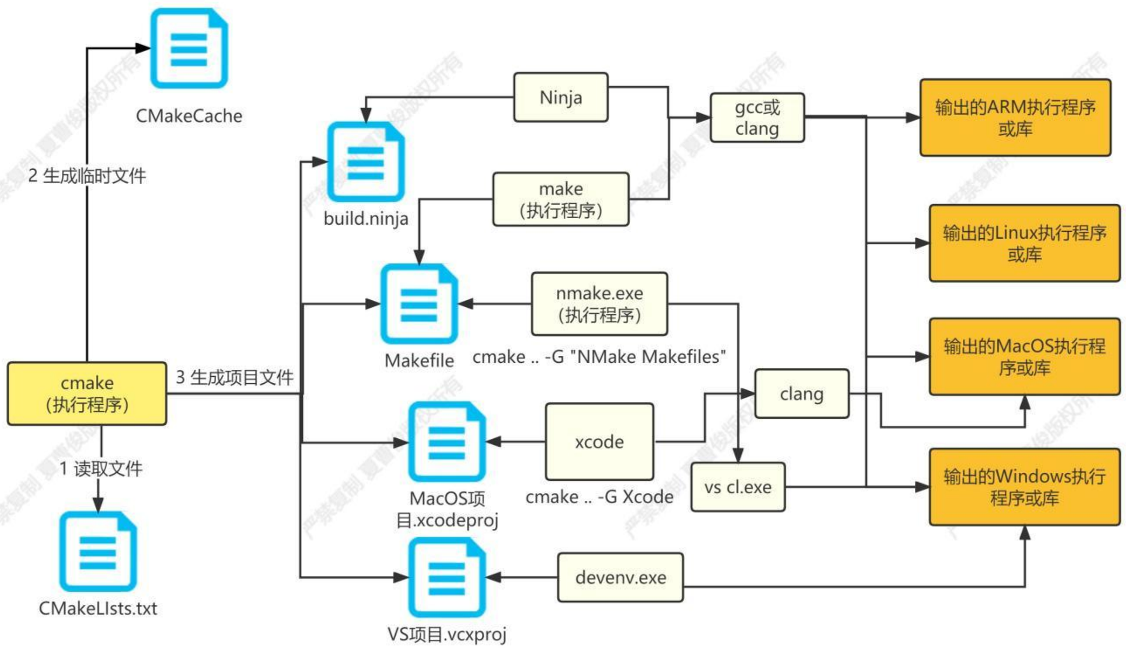
- 在静态和共享构建之间轻松切换的能力

- 在大多数平台上自动生成文件依赖项并支持并行构建

- 每个 IDE 都支持 CMake （ CMake 支持几乎所有 IDE）

- 使用 CMake 的软件包比任何其他系统都多

三 cmake 工作原理

- 

关键词

- cmake 是干什么的

- cmake 使用方法详解

## 1.2 cmake 安装

## 1.2.1 Linux 安装 CMake（ubuntu 20.04 LT）

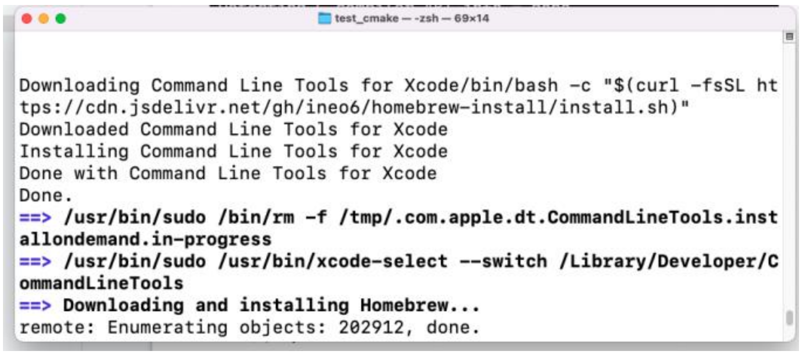
- 一　前置要求
    - 安装好 ubuntu 20.04 版本 64 位系统
        - 如果安装其他系统， 可能有不确定问题，需要微调，安装步骤是一致的
        - 系统可以是独立主机、虚拟机、wsl
    - 配置好系统网络
        - 需要在线安装编译工具
    - 准备好 cmake 源码
        - cmake-3.23.1.tar.gz
            - 提取码：1234　链接：https://pan.baidu.com/s/1AAfCb3oTA8cguIRg8wR-zg
- 二 直接安装
    - apt install cmake
    - 二进制安装
- 三 源码编译安装
    - 1 安装编译工具和依赖库

    - sudo apt install g++
    - sudo apt install make
    - apt install ninja-build
    - apt install unzip
    - apt install libssl-dev
- 2 下载解压 cmake 源码并编译
    - wget https://github.com/Kitware/CMake/releases/download/v3.23.1/cmake-3.23.1.tar.gz
    - tar -xvf cmake-3.23.1.tar.gz
    - cd cmake-3.23.1
    - ./configure
    - make -j32
- 3 安装编译好的 cmake
    - sudo make install
        - 安装路径在 /usr/local/share/cmake-3.23
- 4 设置 cmake 的运行路径
    - vi ~/.bash_profile

    - 文件中添加
    - export PATH=/usr/local/share/cmake-3.22:$PATH
  - 5 运行 cmake 查看版本
    - cmake --version
      - cmake version 3.23.1CMake suite maintained and supported by Kitware (kitware.com/cmake).

1.2.2 MacOS 安装

- 一 前置要求
  - 安装好 xcode 开发工具 （clang）
  - 确认 Command Line Tools for Xcode 已经安装
    - 命令行运行 c 程序
    - xcode-select --version
    - xcode-select --install
  - 安装好 brew
    - /bin/bash -c "$(curl -fsSL https://cdn.jsdelivr.net/gh/ineo6/homebrew-install/install.sh)"
      -

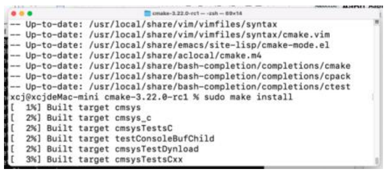


```
Downloading Command Line Tools for Xcode/bin/bash -c "$(curl -fsSL ht
tps://cdn.jsdelivr.net/gh/ineo6/homebrew-install/install.sh)"
Downloaded Command Line Tools for Xcode
Installing Command Line Tools for Xcode
Done with Command Line Tools for Xcode
Done.
==> /usr/bin/sudo /bin/rm -f /tmp/.com.apple.dt.CommandLineTools.inst
allondemand.in-progress
==> /usr/bin/sudo /usr/bin/xcode-select --switch /Library/Developer/C
ommandLineTools
==> Downloading and installing Homebrew...
remote: Enumerating objects: 202912, done.
```

- 准备好 cmake 的源码 3.23.1
  - 源码同上 linux
    - 提取码：1234　链接：https://pan.baidu.com/s/1AAfCb3oTA8cguIRg8wR-zg
    - cmake.org.cn
- macos 系统版本
  - macOS Monterey 12.3.1

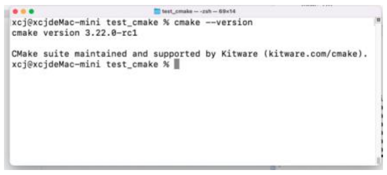
- 二 源码编译安装
  - 安装编译工具
    - brew install make
    - brew install clang
    - brew install clang++
  - 编译安装

- tar -xvf cmake-3.23.1.tar.gz
- cd cmake-3.23.1
- ./configure
- make -j16
- sudo make install
-



```
-- Up-to-date: /usr/local/share/vim/vimfiles/syntax
-- Up-to-date: /usr/local/share/vim/vimfiles/syntax/cmake.vim
-- Up-to-date: /usr/local/share/emacs/site-lisp/cmake-mode.el
-- Up-to-date: /usr/local/share/aclocal/cmake.m4
-- Up-to-date: /usr/local/share/bash-completion/completions/cmake
-- Up-to-date: /usr/local/share/bash-completion/completions/cpack
-- Up-to-date: /usr/local/share/bash-completion/completions/ctest
xcj@xcjdeMac-mini cmake-3.22.0-rc1 % sudo make install
[  1%] Built target cmsys
[  2%] Built target cmsys_c
[  2%] Built target cmsysTestsC
[  2%] Built target testConsoleBufChild
[  2%] Built target cmsysTestDynload
[  3%] Built target cmsysTestsCxx
```

- 查看安装
  - cmake --version
  -

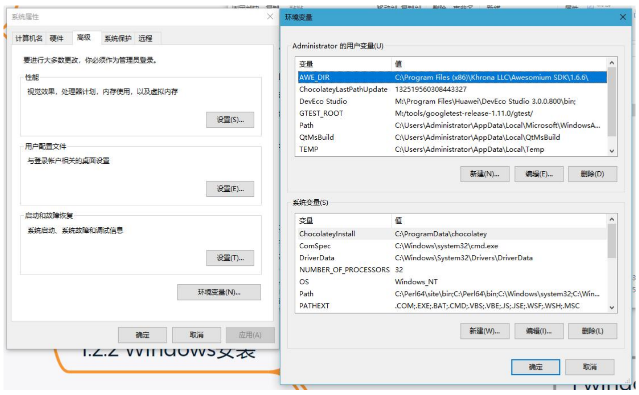


```
xcj@xcjdeMac-mini test_cmake % cmake --version
cmake version 3.22.0-rc1

CMake suite maintained and supported by Kitware (kitware.com/cmake).
xcj@xcjdeMac-mini test_cmake % 
```

### 1.2.3 Windows 安装

- 一　前置要求
  - 安装好 vs 开发工具  2017 2019 2022 都可以
  - 准备好 cmake 的源码和二进制发行包
    - 源码同上 linux

- cmake3.23.1 二进制发行包
    - 提取码：1234　链接：https://pan.baidu.com/s/1AAfCb3oTA8cguIRg8wR-zg
    - cmake.org.cn

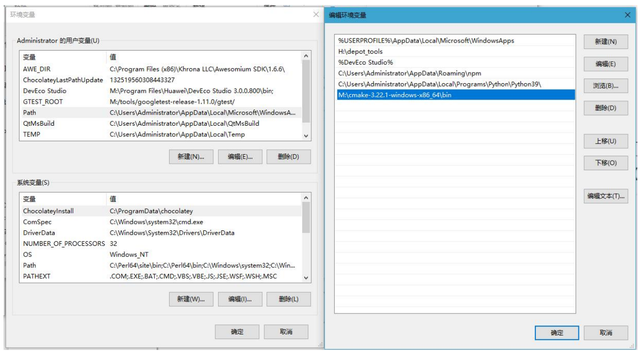
- 二　发布文件安装
    - 1 windows 系统属性-》高级-》环境变量=》设置 Path
        - 
          系统属性
          计算机名　硬件　高级　系统保护　远程
          要进行大多数更改，你必须作为管理员登录。
          性能
          视觉效果，处理器计划，内存使用，以及虚拟内存
          设置(S)...
          用户配置文件
          与登录帐户相关的桌面设置
          设置(E)...
          启动和故障恢复
          系统启动、系统故障和调试信息
          设置(T)...
          环境变量(N)...
          确定　取消　应用(A)

          环境变量
          Administrator 的用户变量(U)

          | 变量 | 值 |
          |---|---|
          | AWE_DIR | C:\Program Files (x86)\Khrona LLC\Awesomium SDK\1.6.6\ |
          | ChocolateyLastPathUpdate | 132519560308443327 |
          | DevEco Studio | M:\Program Files\Huawei\DevEco Studio 3.0.0.800\bin; |
          | GTEST_ROOT | M:/tools/googletest-release-1.11.0/gtest/ |
          | Path | C:\Users\Administrator\AppData\Local\Microsoft\WindowsA... |
          | QtMsBuild | C:\Users\Administrator\AppData\Local\QtMsBuild |
          | TEMP | C:\Users\Administrator\AppData\Local\Temp |

          新建(N)...　编辑(E)...　删除(D)
          系统变量(S)

          | 变量 | 值 |
          |---|---|
          | ChocolateyInstall | C:\ProgramData\chocolatey |
          | ComSpec | C:\Windows\system32\cmd.exe |
          | DriverData | C:\Windows\System32\Drivers\DriverData |
          | NUMBER_OF_PROCESSORS | 32 |
          | OS | Windows_NT |
          | Path | C:\Perl64\site\bin;C:\Perl64\bin;C:\Windows\system32;C:\Win... |
          | PATHEXT | .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC |

          新建(W)...　编辑(I)...　删除(L)
          确定　取消
        - 
          环境变量
          Administrator 的用户变量(U)

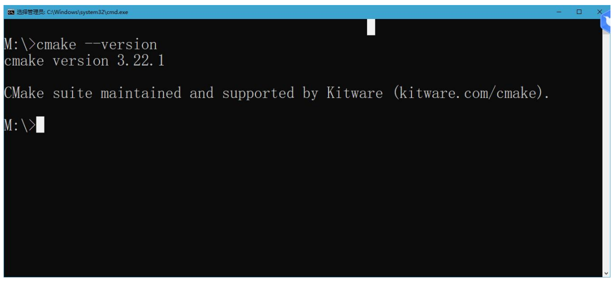
          | 变量 | 值 |
          |---|---|
          | AWE_DIR | C:\Program Files (x86)\Khrona LLC\Awesomium SDK\1.6.6\ |
          | ChocolateyLastPathUpdate | 132519560308443327 |
          | DevEco Studio | M:\Program Files\Huawei\DevEco Studio 3.0.0.800\bin; |
          | GTEST_ROOT | M:/tools/googletest-release-1.11.0/gtest/ |
          | Path | C:\Users\Administrator\AppData\Local\Microsoft\WindowsA... |
          | QtMsBuild | C:\Users\Administrator\AppData\Local\QtMsBuild |
          | TEMP | C:\Users\Administrator\AppData\Local\Temp |

          新建(N)...　编辑(E)...　删除(D)
          系统变量(S)

          | 变量 | 值 |
          |---|---|
          | ChocolateyInstall | C:\ProgramData\chocolatey |
          | ComSpec | C:\Windows\system32\cmd.exe |
          | DriverData | C:\Windows\System32\Drivers\DriverData |
          | NUMBER_OF_PROCESSORS | 32 |
          | OS | Windows_NT |
          | Path | C:\Perl64\site\bin;C:\Perl64\bin;C:\Windows\system32;C:\Win... |
          | PATHEXT | .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC |

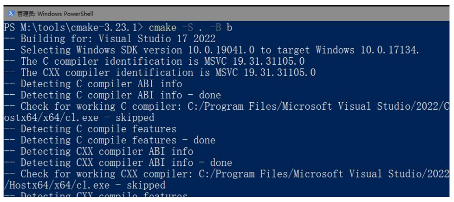
          新建(W)...　编辑(I)...　删除(L)
          确定　取消

          编辑环境变量
          ```
          %USERPROFILE%\AppData\Local\Microsoft\WindowsApps
          H:\depot_tools
          %DevEco Studio%
          C:\Users\Administrator\AppData\Roaming\npm
          C:\Users\Administrator\AppData\Local\Programs\Python\Python39\
          M:\cmake-3.22.1-windows-x86_64\bin
          ```
          新建(N)　编辑(E)　浏览(B)...　删除(D)　上移(U)　下移(O)　编辑文本(T)...
          确定　取消
        -



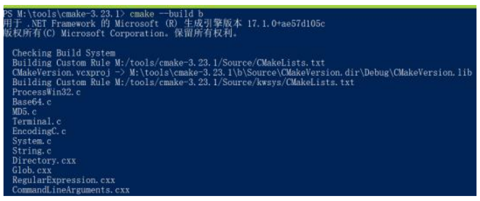
```
M:\>cmake --version
cmake version 3.22.1

CMake suite maintained and supported by Kitware (kitware.com/cmake).

M:\>
```

- 三  源码编译安装（选学）用 cmake 构建 cmake
  - 解压源码后  控制台进入 cmake 源码目录
    - 
  - 生成项目文件 cmake -S . -B b
    -



```
PS M:\tools\cmake-3.23.1> cmake -S . -B b
-- Building for: Visual Studio 17 2022
-- Selecting Windows SDK version 10.0.19041.0 to target Windows 10.0.17134.
-- The C compiler identification is MSVC 19.31.31105.0
-- The CXX compiler identification is MSVC 19.31.31105.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: C:/Program Files/Microsoft Visual Studio/2022/C
ostx64/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/Microsoft Visual Studio/2022
/Hostx64/x64/cl.exe - skipped
-- Detecting CXX compile features
```

  - 编译  cmake --build b --config Release
    -



```
PS M:\tools\cmake-3.23.1> cmake --build b
用于 .NET Framework 的 Microsoft (R) 生成引擎版本 17.1.0+ae57d105c
版权所有(C) Microsoft Corporation。保留所有权利。

  Checking Build System
  Building Custom Rule M:/tools/cmake-3.23.1/Source/CMakeLists.txt
  CMakeVersion.vcxproj -> M:\tools\cmake-3.23.1\b\Source\CMakeVersion.dir\Debug\CMakeVersion.lib
  Building Custom Rule M:/tools/cmake-3.23.1/Source/kwsys/CMakeLists.txt
  ProcessWin32.c
  Base64.c
  MD5.c
  Terminal.c
  EncodingC.c
  System.c
  String.c
  Directory.cxx
  Glob.cxx
  RegularExpression.cxx
  CommandLineArguments.cxx
```

  - 安装 cmake --install b
    - C:/Program Files (x86)/CMake/

关键词

- cmake 安装
- cmake 教程
- linux cmake 安装
- 查看 cmake 版本

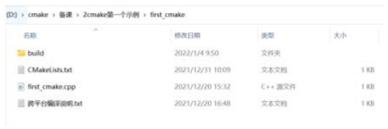
## 1.3 cmake 第一个示例 cmakelist

### 一 前置准备

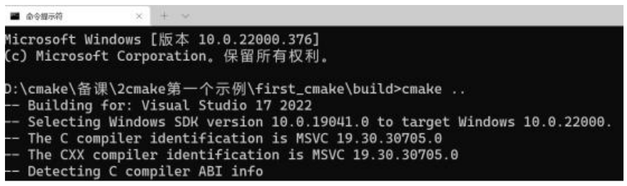
- 准备测试的 c++程序文件 first_cmake.cpp
  - //first_cmake.cpp#include <iostream>using namespace std;int main(int argc,char *argv[]){ cout<<"first cmake c++"<<endl; return 0;}
- 在源码的同目录下编写第一个 CMakeLists.txt
  - # CMakeLists.txt# 指定 cmake 的最低版本 cmake_minimum_required (VERSION 3.20)# 构建的项目名称 project (first_cmake)# 构建执行程序 add_executable(first_cmake first_cmake.cpp)

### 二 Windows 平台编译

- CMake=》vs 项目=》cl 编译
- 4 自动创建构建目录

- 生成项目文件
  - 生成 VS 项目
    - 1 源码目录下面创建一个编译目录 build，用于生成 cmake 的临时文件和项目文件，放在独立的目录方便清理和查看
    - 

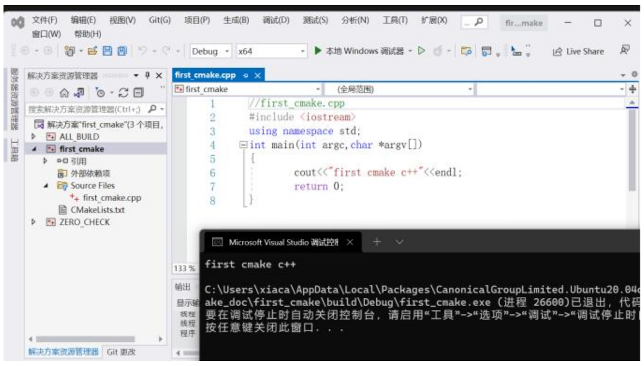
```
(D:) › cmake › 备课 › 2cmake第一个示例 › first_cmake
名称                  修改日期             类型        大小
build                2022/1/4 9:50       文件夹
CMakeLists.txt       2021/12/31 10:09    文本文档     1 KB
first_cmake.cpp      2021/12/20 15:32    C++ 源文件   1 KB
跨平台编译说明.txt     2021/12/20 16:48    文本文档     1 KB
```

    - 2 进入编译目录 build，直接运行 cmake .. 使用默认生成项目文件，下图生成的是 vs2022 的 64 位项目（文件所在的路径和目录层次不能太深，太深会找不到编译器）
    - 

```
命令提示符
Microsoft Windows [版本 10.0.22000.376]
(c) Microsoft Corporation。保留所有权利。

D:\cmake\备课\2cmake第一个示例\first_cmake\build>cmake ..
-- Building for: Visual Studio 17 2022
-- Selecting Windows SDK version 10.0.19041.0 to target Windows 10.0.22000.
-- The C compiler identification is MSVC 19.30.30705.0
-- The CXX compiler identification is MSVC 19.30.30705.0
-- Detecting C compiler ABI info
```

  - 生成 nmake 项目
    - 运行 vs 控制台编译工具 x64 Native Tools Command Prompt for VS 2022 Current
    - 进入源码目录 cmake -S . -B build -G "NMake Makefiles"
- 编译构建
  - 使用 vs 编译

- 3 进入 build 目录打开解决方案

- 

```cpp
//first_cmake.cpp
#include <iostream>
using namespace std;
int main(int argc,char *argv[])
{
        cout<<"first cmake c++"<<endl;
        return 0;
}
```

```
first cmake c++

C:\Users\xiaca\AppData\Local\Packages\CanonicalGroupLimited.Ubuntu20.04
ake_doc\first_cmake\build\Debug\first_cmake.exe (进程 26600)已退出，代码
要在调试停止时自动关闭控制台，请启用"工具"->"选项"->"调试"->"调试停止时
按任意键关闭此窗口. . .
```

- cmake 命令编译
  - cmake --build build
- nmake 编译
  - nmake

三 Linux（Ubuntu）平台编译

- 前置准备
  - 安装好 gcc 编译工具
    - sudo apt install g++
    - sudo apt install make
  - 如果需要用到 Ninja
    - sudo apt install ninja-build

- 生成项目文件
  - 生成 makefile
    - cmake -S . -B build
      -
  - 生成 Ninja 项目
    - cmake -S . -B build -G "Ninja"
- 指定项目工具
  - 在 linux 主要有两种，一种是生成 make 的 makefile 一种是生成 Ninja 的 build.ninja 生成 makefile 见上面示例生成 Ninjacmake .. -G Ninja

四 MacOS 平台编译

- 安装好 xcode 开发工具 （clang）
- 确认 Command Line Tools for Xcode 已经安装
  - 命令行运行 c 程序
  - xcode-select --version
  - xcode-select --install
- cmake -S . -B build
  - 出现错误

        - -- The CXX compiler identification is unknown
    - 解决
        - sudo xcode-select --switch /Applications/Xcode.app/
- cmake -S . -B xcode -G Xcode
    - cmake --open xcode

最后的建议

- windows 生成 vs 项目
- linux MacOS 生成 makefile

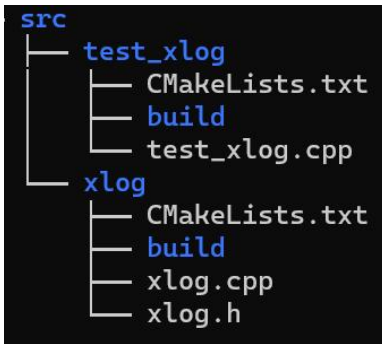
源码下载

关键词

- cmakelist
- windows cmake
- linux cmake
- ubuntu cmake
- cmake make
- visual studio cmake

## 1.4 cmake 构建静态库与动态库

前置准备

- 
```
src
├── test_xlog
│   ├── CMakeLists.txt
│   ├── build
│   └── test_xlog.cpp
└── xlog
    ├── CMakeLists.txt
    ├── build
    ├── xlog.cpp
    └── xlog.h
```
- 本节课尽量精简使用 cmake 特性，后面可以会一步步加入自动操作

动态库和静态库概念(xlog)

- 静态库
  - 文件名
    - windows
      - xlog.lib
        - xlog_d.lib
    - linux(ubuntu、Android 、鸿蒙（HarmonyOS ））
      - libxlog.a
    - macOS

         - libxlog.a
   - 基本可以理解为编译后的二进制代码，类似.o
- 动态库
   - 文件名
      - windows
         - xlog.lib
            - 函数地址索引
         - xlog.dll
            - 函数二进制代码
      - linux(ubuntu、Android 、鸿蒙（HarmonyOS ））
         - libxlog.so
      - macOS
         - libxlog.dylib
- 头文件作用
   - 函数名称和参数类型（用于索引查找函数地址）
   - 不引用，可以自己直接声明函数
   - 知道名字可以调用系统 api 查找函数

cmake 编译静态库

- # src/xlog/CMakeLists.txtcmake_minimum_required (VERSION 3.0)project (xlog)add_library(xlog STATIC xlog.cpp)

cmake 链接静态库

- #src/test_xlog/CMakeLists.txtcmake_minimum_required (VERSION 3.0)project (test_xlog)# 指定头文件路径 include_directories ("../xlog")# 指定库文件加载路径 link_directories ("../xlog/build" )add_executable(test_xlog test_xlog.cpp)# 指定加载的库 target_link_libraries (test_xlog   xlog )

cmake 动态库 编译链接

- 102cmake_lib/ ├── CMakeLists.txt ├── test_xlog│   ├── CMakeLists.txt│   └── test_xlog.cpp└── xlog    ├── CMakeLists.txt    ├── xlog.cpp    └── xlog.h
- 库和测试项目在一个 CMakeLists.txt 中配置
- 库运行时加载的路径，自动添加了编译参数
    - -Wl,-rpath,/opt/mker/poco/lib
- CMakeLists.txt
    - cmake_minimum_required (VERSION 3.20)project (xlog)include_directories ("xlog")add_executable(test_xlog ../test_xlog/test_xlog.cpp)add_library(xlog SHARED ../xlog/xlog.cpp)target_link_libraries (test_xlog xlog )
        - -Wl,-rpath,/opt/mker/poco/lib

  - xlog_EXPORTS
    - __declspec(dllexport)
    - __declspec(dllimport)
  - cmake 自动给库项目添加 库名称_EXPORTS 预处理变量
- code
  - #ifndef XLOG_H#define XLOG_H//__declspec(dllexport)//__declspec(dllexport) 导出 XLog 类的函数到 lib 文件中// xlog 库文件调用 dllexport// test_xlog 调用 dllimport#ifndef _WIN32 #define XCPP_API#else #ifdef xlog_EXPORTS #define XCPP_API __declspec(dllexport) //库项目调用 #else #define XCPP_API __declspec(dllimport) //调用库项目调用 #endif#endifclass XCPP_API XLog{public:XLog();};#endif

关键词

- cmake 链接静态库
- cmake 动态库

章节设计目的

- 为什么用 cmake

    - 环境挡住第一波
    - 涉及不同的开发平台
        - mac
        - win
        - linux
    - 快速入门
- 前置要求
    - 操作系统
    - 开发工具
        - g++
- 怎么学习

# 第二章　每个项目都会用到的-CMake 常用功能

## 2.1 cmake 注释

### 括号注释

- #[[第一行注释。第二行注释.]]message("参数 1\n" #[[中间的注释]] "参数 2")
- 3.0 之前的 CMake 版本不支持括号注释

行注释

- 行注释，一直运行到行尾

## 2.2 cmake message 详解

message 基础使用

- message（arg1 arg2 arg3 ）

message 高级使用-指定日志级别

- message([<mode>] "message text" ...)
- --log-level=<ERROR|WARNING|NOTICE|STATUS|VERBOSE|DEBUG|TRACE>
- 1 标准输出 stdout 2 错误输出 stderr
- 日志级别
    - FATAL_ERROR
        - 停止 cmake 运行和生成
            - printed to stderr
    - SEND_ERROR
        - cmake 继续运行，生成跳过
            - printed to stderr

- WARNING
    - printed to stderr
- (none) or NOTICE
    - printed to stderr
- STATUS
    - 项目用户可能感兴趣的信息
- VERBOSE
    - 针对项目用户的详细信息
- DEBUG
    - 项目本身的开发人员使用的信息
- TRACE
    - 非常低级实现细节的细粒度消息

- CMakeLists.txt
    - #生成到此终止 cmake -S . -B b --log-level ERROR #message(FATAL_ERROR "运行终止 生成终止 FATAL_ERROR")message(SEND_ERROR "继续运行生成终止 SEND_ERROR")message(WARNING "WARNING 显示行号")message(STATUS "STATUS 显示--")message(VERBOSE "VERBOSE 默认不显--")message(DEBUG "DEBUG 默认不显--")message(TRACE "TRACE 默认不显

--")#ERROR(FATAL_ERROR SEND_ERROR) > WARNING > STATUS > VERBOSE > DEBUG >TRACE

message Reporting checks 查找库日志

- Reporting checks　　message(<checkState> "message text" ...)
    - CHECK_START
        - 开始记录将要执行检查的消息
    - CHECK_PASS
        - 记录检查的成功结果
    - CHECK_FAIL
        - 记录不成功的检查结果
- 可嵌套
- STATUS 日志级别
- CMakeLists.txt
    - message("CMAKE_MESSAGE_INDENT = " ${CMAKE_MESSAGE_INDENT})set(CMAKE_MESSAGE_INDENT " ## ") #  消息对齐 message(CHECK_START "Finding xcpp")unset(miss)message(CHECK_START "Finding xlog")# ... do check, assume we find xlogmessage(CHECK_PASS "found")message(CHECK_START "Finding xthread")# ... do check, assume we don't find xthreadset(miss ${miss}[xthread])message(CHECK_FAIL "not

found")message(CHECK_START "Finding xsocket")# ... do check, assume we don't find xsocketset(miss ${miss}[xsocket])message(CHECK_FAIL "not found")set(CMAKE_MESSAGE_INDENT "")if(miss)    message(CHECK_FAIL "丢失组件: ${miss}")else()    message(CHECK_PASS "all components found")endif()

关键词

- cmake message

103message

## 2.3 cmake 变量入门

关键字

- cmake set
    - set 将一个 CMAKE 变量设置为给定值。 set(<variable> <value> ) 将变量<variable>的值设置为<value>如果没有指定<value>，那么这个变量就会被撤销而不是被设置。

104test_ver

变量语法

- set
    - 将一个 CMAKE 变量设置为给定值。 set(<variable> <value> ) 将变量<variable>的值设置为<value>如果没有指定<value>，那么这个变量就会被撤销而不是被设置。
- unset(<variable>)

变量使用

- 变量引用是值替换，如果未设置变量，返回空字符串
- 变量引用可以嵌套并从内向外求值
- 变量名大小写敏感

变量与字符串

- string(ASCII 27 Esc)
- "${VAR}变量直接在字符串中"

变量让 message 输出不同的颜色

- string(ASCII 27 Esc)
- \033[1;31;40m　　<!--1-高亮显示　31-前景色红色　　40-背景色黑色-->\033[0m　　　<!--采用终端默认设置，即取消颜色设置-->
- Windows PowerShell
- code
  - cmake_minimum_required(VERSION 3.20)project("test_v1" )string(ASCII 27 Esc)string(ASCII 70 A)message(${A})#[[格式：\33[显示方式;前景色;背景色 m 显示方式　　　　　意义-------------------------0　　　　　　　　终端默认设置 1　　　　　　　高亮显示 4　　　　　　　使用下划线 5　　　　　闪烁 7　　　　　　　反白显示 8　　　　　　　不可见\033[1;31;40m

<!--1-高亮显示 31-前景色红色 40-背景色黑色-->前景色 背景色 颜色---------------------------------------30 40 黑色 31 41 红色 32 42 绿色 33 43 黄色 34 44 蓝色 35 45 紫红色 36 46 青蓝色 37 47 白色]]message("33")set(E "${Esc}[m")set(R "${Esc}[31m")set(G "${Esc}[32m")set(Y "${Esc}[33m")set(B "${Esc}[34m")set(RB "${Esc}[4;31;40m") #红黑 message("${R}这是红色${E}")message("${G}这是绿色${E}")message("${Y}这是黄色${E}")message("${B}这是蓝色${E}")message("${RB}这是红黑${E}")

- 105message_color

cmake 内建变量

- 提供信息的变量
  - PROJECT_NAME
    - project()项目名称
- 改变行为的变量
  - BUILD_SHARED_LIBS
    - 缓存变量
    - add_library()
    - ON

            - 创建共享库
        - OFF
            - 创建静态库
- 描述系统的变量
    - MSVC
    - WIN32
    - ANDROID
    - UNIX
        - Set to True when the target system is UNIX or UNIX-like
    - CMAKE_SYSTEM_NAME
- 控制构建过程的变量
    - CMAKE_COLOR_MAKEFILE
        - 生成的 makefile 是否有颜色，默认是 ON
- 项目代码
    - CMakeLists.h
    - xlog.h

  - xlog.cpp
  - 使用 102cmake_lib 的代码
- http://cmake.org.cn/cmake_html/manual/cmake-variables.7.html

CMake 给 c++传递变量

- add_definitions(-Dxlog_STATIC)
  - 默认值是 1
- add_definitions(-DSTATIC=1)

## 2.4 cmake include

从给定的文件中读取 CMake 的列表文件。include(file　[OPTIONAL] [RESULT_VARIABLE VAR ] )　　从给定的文件中读取 CMake 的清单文件代码。在清单文件中的命令会被立即处理。如果指定了 OPTIONAL 选项，那么如果被包含文件不存在的话，不会报错。如果指定了 RESULT_VARIABLE 选项，那么 var 或者会被设置为被包含文件的完整路径，或者是

NOTFOUND，表示没有找到该文件

cmake/test_cmake.cmake

- message("in test cmake ")

CMakeLists.txt

- cmake_minimum_required (VERSION 3.0)project("test_include")message("begin include")include(cmake/test_cmake.cmake )include(cmake/test_cmake.cmake )i

nclude(cmake/test_cmake1.cmake OPTIONAL) # OPTIONAL 文件不存在，不报错 include(cmake/test_cmake1.cmake OPTIONAL RESULT_VARIABLE ret_val ) # NOTFOUNDMESSAGE("install return value is ${ret_val}")include(cmake/test_cmake.cmake OPTIONAL RESULT_VARIABLE ret_val ) # NOTFOUNDMESSAGE("install return value is ${ret_val}") # 返回文件全路径 message("after include")

107cmake_include

## 2.5 自动查找所有源码文件和头文件

项目准备 108auto_src_h

- 108auto_src_h/ ├── CMakeLists.txt ├── include│ ├── xlog.h│ └── xthread.hpp ├── main.cpp └── src ├── xlog.cpp ├── xtest.c └── xthread.cc

增加头文件和代码后不用修改 cmake

aux_source_directory

- aux_source_directory("./src" LIB_SRCS) # 当前路径下所有源码 存入 DIR_SRCS

file

- FILE(GLOB H_FILE "${INCLUDE_PATH}/xcpp/*.h")FILE(GLOB H_FILE_I "${INCLUDE_PATH}/*.h")

## 2.6 cmake 命令实现程序的分步生成

从源码到执行程序

- 多文件演示

查看所有目标

- cmake --build . --target help

预处理

- cmake --build . --target first_cmake.i

编译

- cmake --build . --target first_cmake.s

汇编

- cmake --build . --target first_cmake.o

链接

运行

- 动态库加载路径

cmake 程序分步生成、指定项目和清理

windows 下必须运行 vs 控制台

- cmake -S . -B nmake -G "NMake Makefiles"

## 2.7 cmake 命令构建指定项目和清理

cmake --build . --target help

cmake --build . --target clean

## 2.8 cmake 调试打印生成的具体指令

CMAKE_VERBOSE_MAKEFILE

- set(CMAKE_VERBOSE_MAKEFILE ON)
  - 默认是 OFF

cmake --build . -v

- 第一次运行就要加-v，不然日志不完整，可以清理后重新生成

101first_cmake

## 2.9 CMake 设置输出路径 add_subdirectory

代码准备

- 102cmake_lib
  - test_xlog
    - CMakeLists.txt
    - test_xlog.cpp

    - xlog
        - CMakeLists.txt
        - xlog.h
        - xlog.cpp
    - CMakeLists.txt
- 106cmake_system_ver
    - CMakeLists.txt
    - xlog.cpp
    - xlog.h
- 109cmake_out
    - test_xlog
        - CMakeLists.txt
        - test_xlog.cpp
        - 复制 102
    - xlog
        - CMakeLists.txt
        - xlog.cpp

        - xlog.h

        - 复制 106

    - CMakeLists.txt

库输出路径

- CMAKE_LIBRARY_OUTPUT_DIRECTORY

- linux 动态库 .so

归档输出路径

- CMAKE_ARCHIVE_OUTPUT_DIRECTORY

- windows 静态库.lib

- windows 动态库地址.lib 文件

- Linux 静态库

    - .a

执行程序输出路径

- CMAKE_RUNTIME_OUTPUT_DIRECTORY

- 执行程序和 dll 动态库

设置路径

- set(CMAKE_ARCHIVE_OUTPUT_DIRECTORY "${CMAKE_CURRENT_LIST_DIR}/lib") set(CMAKE_LIBRARY_OUTPUT_DIRECTORY "${CMAKE_CURRENT_LIST_DIR}/lib")set(CMAKE_RUNTIME_OUTPUT_DIRECTORY "${CMAKE_CURRENT_LIST_DIR}/bin")

遗留问题

- 多个项目不同输出路径
- Debug 和 Release 不同输出
- 一个项目同时要设置静态库和动态库

# 第三章 编写灵活的项目配置-cmake 语法

## 3.1. if 控制流程

语法格式

- if(<condition>) <commands>elseif(<condition>) # optional block, can be repeated <commands>else() # optional block <commands>endif()

基本表达式

- if(<constant>)
  - 1, ON, YES, TRUE,Y 或非零数（包括浮点数），则为真

  - 0, OFF, NO, FALSE, N, IGNORE, NOTFOUND, 空字符串，或以-NOTFOUND 结尾则为假

- if(<variable>)
  - 非假值常量为真。未定义和其他为假
  - 环境变量总为假
  - 宏参数不是变量
- if(<string>)
  - 字符串的值是真正的常量真
    - 其他带引号的字符串始终计算为 false

逻辑操作符

- NOT AND OR
  - if(NOT <condition>)
  - if(<cond1> AND <cond2>)
  - if(<cond1> OR <cond2>)
  - if((condition) AND (condition OR (condition)))

if 判断语句

- 一元判断

    - EXISTS
    - COMMAND
    - DEFINED
- 二元判断
    - EQUAL
    - EQUAL, LESS, LESS_EQUAL, GREATER, GREATER_EQUAL
    - STREQUAL, STRLESS, STRLESS_EQUAL, STRGREATER, STRGREATER_EQUAL
    - VERSION_EQUAL, VERSION_LESS, VERSION_LESS_EQUAL, VERSION_GREATER, VERSION_GREATER_EQUAL
    - MATCHES
        - if(<variable|string> MATCHES regex)
- 存在性检查
    - if(COMMAND command-name)如果给定名称是可以调用的命令、宏或函数，则为真。if(POLICY policy-id)如果给定名称是现有策略（形式为 CMP<NNNN>），则为真。if(TARGET target-name)如果给定名称是由调用创建的现有逻辑目标名称，则为真 add_executable(),add_library()， 或者 add_custom_target()已经调用的命令（在任何目录中）。if(TEST test-name)3.3 版新功能：如果给定名称是由 add_test()命令。if(DEFINED <name>|CACHE{<name>}|ENV{<name>})如果定义了给定的变量、缓存变量或环境变量，则为真<name>。变量的值无关紧要。请注意以下警告：宏参数

不是变量。无法直接测试<name>是否为非缓存变量。如果存在缓存或非缓存变量，则表达式将评估为真。相比之下，只有存在缓存变量时，表达式才会计算为真。如果您需要知道是否存在非缓存变量，则需要测试这两个表达式： .if(DEFINED someName)someNameif(DEFINED CACHE{someName})someNameif(DEFINED someName AND NOT DEFINED CACHE{someName})3.14 新版功能：增加了对 CACHE{<name>}变量的支持。if(<variable|string> IN_LIST <variable>)3.3 新版功能：如果给定元素包含在命名列表变量中，则为真。

- 文件操作

  - if(EXISTS path-to-file-or-directory)如果指定的文件或目录存在，则为真。行为仅针对显式完整路径进行了明确定义（前导~/不扩展为主目录，并且被视为相对路径）。解析符号链接，即如果指定的文件或目录是符号链接，如果符号链接的目标存在，则返回 true。if(file1 IS_NEWER_THAN file2)file1 如果两个文件更新 file2 或两个文件之一不存在，则为真。行为仅针对完整路径进行了明确定义。如果文件时间戳完全相同，则 IS_NEWER_THAN 比较返回 true，以便在出现平局时发生任何相关的构建操作。这包括为 file1 和 file2 传递相同文件名的情况。if(IS_DIRECTORY path-to-directory)如果给定名称是目录，则为真。行为仅针对完整路径进行了明确定义。if(IS_SYMLINK file-name)如果给定名称是符号链接，则为真。行为仅针对完整路径进行了明确定义。if(IS_ABSOLUTE path)如果给定路径是绝对路径，则为真。请注意以下特殊情况：一个空的 path 评估为假。在 Windows 主机上，任何 path 以驱动器号和冒号（例如 C:）、正斜杠或反斜杠开头的都将评估为真。这意味着路径 likeC:no\base\dir 将评估为 true，即使路径的非驱动部分是相对的。在非 Windows 主机上，任何 path 以波浪号（~）开头的都计算为真。

- 比较

- if(<variable|string> MATCHES regex)如果给定的字符串或变量的值与给定的正则表达式匹配，则为真。有关正则表达式格式，请参阅正则表达式规范。3.9 版中的新功能：()组被捕获在 CMAKE_MATCH_<n>变量。if(<variable|string> LESS <variable|string>)如果给定字符串或变量的值是有效数字且小于右侧的数字，则为真。if(<variable|string> GREATER <variable|string>)如果给定的字符串或变量的值是有效数字并且大于右边的数字，则为真。if(<variable|string> EQUAL <variable|string>)如果给定字符串或变量的值是有效数字并且等于右侧的数字，则为真。if(<variable|string> LESS_EQUAL <variable|string>)3.7 版新功能：如果给定字符串或变量的值是有效数字且小于或等于右侧的数字，则为真。if(<variable|string> GREATER_EQUAL <variable|string>)3.7 新版功能：如果给定字符串或变量的值是有效数字并且大于或等于右侧的数字，则为真。if(<variable|string> STRLESS <variable|string>)如果给定字符串或变量的值按字典顺　　序小于右侧的字符串或变量，则为真。if(<variable|string> STRGREATER <variable|string>)如果给定字符串或变量的值按字典顺　　序大于右侧的字符串或变量，则为真。if(<variable|string> STREQUAL <variable|string>)如果给定字符串或变量的值在字典上等于右侧的字符串或变量，则为真。if(<variable|string> STRLESS_EQUAL <variable|string>)3.7 版中的新功能：如果给定字符串或变量的值按字典顺　　序小于或等于右侧的字符串或变量，则为真。if(<variable|string> STRGREATER_EQUAL <variable|string>)3.7 新版功能：如果给定字符串或变量的值在字典上大于或等于右侧的字符串或变量，则为真。

- 版本比较

  - if(<variable|string> VERSION_LESS <variable|string>)组件整数版本号比较（版本格式为 major[.minor[.patch[.tweak]]]，省略的组件被视为零）。任何非整数版本组件或版本组件的非整数尾随部分都会在该点有效地截断字符

串。if(<variable|string> VERSION_GREATER <variable|string>)组件整数版本号比较（版本格式为 major[.minor[.patch[.tweak]]]，省略的组件被视为零）。任何非整数版本组件或版本组件的非整数尾随部分都会在该点有效地截断字符串。if(<variable|string> VERSION_EQUAL <variable|string>)组件整数版本号比较（版本格式为 major[.minor[.patch[.tweak]]]，省略的组件被视为零）。任何非整数版本组件或版本组件的非整数尾随部分都会在该点有效地截断字符串。if(<variable|string> VERSION_LESS_EQUAL <variable|string>)3.7 版中的新功能：组件方式的整数版本号比较（版本格式为 major[.minor[.patch[.tweak]]]，省略的组件被视为零）。任何非整数版本组件或版本组件的非整数尾随部分都会在该点有效地截断字符串。if(<variable|string> VERSION_GREATER_EQUAL <variable|string>)3.7 版中的新功能：组件方式的整数版本号比较（版本格式为 major[.minor[.patch[.tweak]]]，省略的组件被视为零）。任何非整数版本组件或版本组件的非整数尾随部分都会在该点有效地截断字符串。

遗留问题

- 判断语句过长
- 无法嵌入到其他功能函数中

## 3.2. 变量和缓存

202cmake_cache

- 202cmake_cache/ ├── CMakeLists.txt ├── sub1│ └── CMakeLists.txt└── sub2 └── CMakeLists.txt

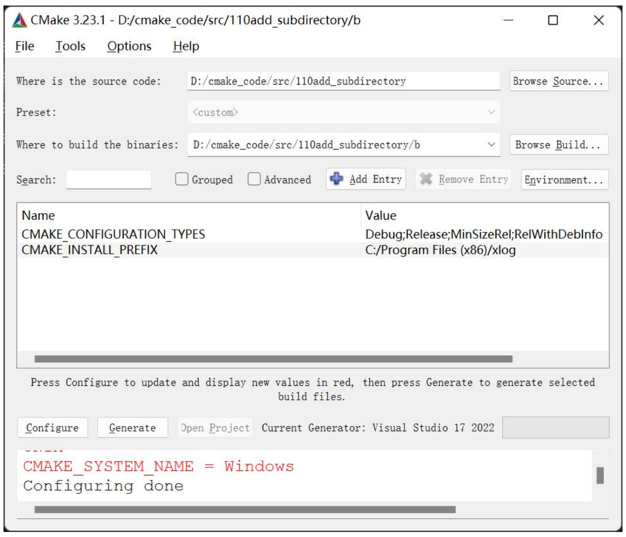
缓存变量的基础语法和使用

- set(<variable> <value>... CACHE <type> <docstring> [FORCE])
  - type
    - BOOL
      - ON/OFF 选择框
    - FILEPATH
      - 文件选择
    - PATH
      - 目录选择
    - STRING
      - A line of text. cmake-gui(1) offers a text field or a drop-down selection if the STRINGS cache entry property is set.
    - INTERNAL
      - A line of text. cmake-gui(1) does not show internal entries. They may be used to store variables persistently across runs. Use of this type implies FORCE.
  - docstring
    - The <docstring> must be specified as a line of text providing a quick summary of the option for presentation to cmake-gui(1) users.
  - FORCE

- If the cache entry does not exist prior to the call or the FORCE option is given then the cache entry will be set to the given value.

缓存变量对应 cmake-gui 和 ccmake

- cmake-gui
  - configure
    - Generate
  -


CMake 3.23.1 - D:/cmake_code/src/110add_subdirectory/b

File　Tools　Options　Help

Where is the source code: D:/cmake_code/src/110add_subdirectory　Browse Source...

Preset: <custom>

Where to build the binaries: D:/cmake_code/src/110add_subdirectory/b　Browse Build...

Search:　☐ Grouped　☐ Advanced　Add Entry　Remove Entry　Environment...

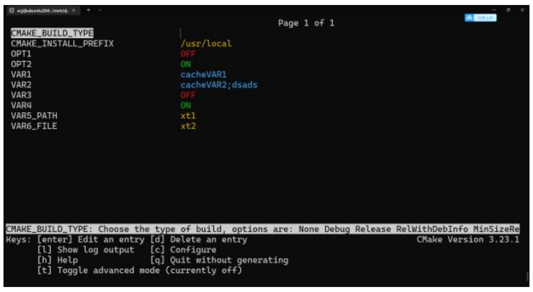
| Name | Value |
|---|---|
| CMAKE_CONFIGURATION_TYPES | Debug;Release;MinSizeRel;RelWithDebInfo |
| CMAKE_INSTALL_PREFIX | C:/Program Files (x86)/xlog |

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Configure　Generate　Open Project　Current Generator: Visual Studio 17 2022

```
CMAKE_SYSTEM_NAME = Windows
Configuring done
```

- ccmake
  - cmake -S . -B build
  - ccmake build
    -



```
                                                Page 1 of 1
CMAKE_BUILD_TYPE
CMAKE_INSTALL_PREFIX             /usr/local
OPT1                             OFF
OPT2                             ON
VAR1                             cacheVAR1
VAR2                             cacheVAR2;dsads
VAR3                             OFF
VAR4                             ON
VAR5_PATH                        xt1
VAR6_FILE                        xt2




CMAKE_BUILD_TYPE: Choose the type of build, options are: None Debug Release RelWithDebInfo MinSizeRe
Keys: [enter] Edit an entry [d] Delete an entry                                 CMake Version 3.23.1
      [l] Show log output   [c] Configure
      [h] Help              [q] Quit without generating
      [t] Toggle advanced mode (currently off)
```

- 修改缓存

- 分类型展示

- option(<variable> "<help_text>" [value])

CMake CACHE 覆盖策略设置

- CMP01263.21 版中的新功能。当此政策设置为 NEW 时，set(CACHE)命令不会从当前范围中删除任何同名的普通变量。在以下情况下，该 OLD 行为会从当前作用域中删除任何同名的普通变量：

  - 以前不存在该名称的缓存变量。该名称的缓存变量以前存在，但它没有类型。当变量在命令行上使用类似的形式而不是.cmake -DMYVAR=blahcmake -DMYVAR:STRING=blah 设置缓存变量时使用了 FORCEorINTERNAL 关键字。

- cmake_policy(SET CMP0126 NEW)

  - NEW

    - 不会删除同名的普通变量

  - OLD

    - 删除同名的普通变量

- $CACHE{NVAR1}

-D 传递缓存变量

- cmake -S . -B build -D PARA1=para001

CMake 内置缓存变量

- BUILD_SHARED_LIBS
- set(BUILD_SHARED_LIBS OFF CACHE BOOL "lib" )
- message("BUILD_SHARED_LIBS = ${BUILD_SHARED_LIBS}")

## 3.3. 属性与变量

CMake 变量和属性有什么区别

- 一种简短的说明是，属性是作用域为目标的变量。
- global property can be a useful uncached global variable

属性语法

- set_property
  - 语法
    - set_property(<GLOBAL |
      DIRECTORY [<dir>] | TARGET
      [<target1> ...] | SOURCE [<src1> ...]
      [DIRECTORY <dirs> ...] [TARGET_DIRECTORY

<targets> ...] | INSTALL [<file1> ...] | TEST [<test1> ...] | CACHE [<entry1> ...] > [APPEND] [APPEND_STRING] PROPERTY <name> [<value1> ...])

- 示例
  - set_property(GLOBAL PROPERTY TEST_GLOBAL " test4")
  - set_property(GLOBAL APPEND PROPERTY TEST_GLOBAL " test string2")
    - APPEND 列表将附加到任何现有的属性值（除了空值被忽略且不附加）
  - set_property(GLOBAL APPEND_STRING PROPERTY TEST_GLOBAL " test string3")
    - 如果 APPEND_STRING 字符串将作为字符串附加到任何现有属性值，更长的字符串而不是字符串列表。
- get_property
  - 语法
    - get_property(<variable> <GLOBAL | DIRECTORY [<dir>] | TARGET <target> | SOURCE <source> [DIRECTORY <dir> | TARGET_DIRECTORY <target>] | INSTALL <file> | TEST <test> | CACHE <entry> | VARIABLE > PROPERTY <name> [SET | DEFINED | BRIEF_DOCS | FULL_DOCS])

    - TARGET_DIRECTORY <target>源文件属性将从 <target>创建的目录范围中读取（<target>因此必须已经存在）
    - DIRECTORY <dir>源文件属性将从<dir>目录的范围中读取

- define_property
  - define_property(<GLOBAL | DIRECTORY | TARGET | SOURCE | TEST | VARIABLE | CACHED_VARIABLE> PROPERTY <name> [INHERITED] [BRIEF_DOCS <brief-doc> [docs...]] [FULL_DOCS <full-doc> [docs...]] [INITIALIZE_FROM_VARIABLE <variable>])

属性分类

- 全局属性
  - 语法
    - set_property(GLOBAL PROPERTY TEST_GLOBAL "test global 001")
    - get_property(val GLOBAL PROPERTY TEST_GLOBAL)
  - 示例
    - add_subdirectory("sub1")
      - sub1/CMakeLists.txt
        - set_property(GLOBAL PROPERTY SUB1_GLOBAL "SUB1_GLOBAL 001")

    - get_property(val GLOBAL PROPERTY SUB1_GLOBAL)message("SUB1_GLOBAL value is ${val}")

- 目录属性
  - 语法
    - set_property(DIRECTORY .　PROPERTY DIR_VAR1 "dir_var1 001")
    - get_property(var DIRECTORY . PROPERTY DIR_VAR1)
  - 示例
    - sub1/CMakeLists.txt
      - set_property(DIRECTORY .　PROPERTY SUB1_DIR_VAR1 "SUB1_DIR_VAR1 001")
    - get_property(var DIRECTORY sub1 PROPERTY SUB1_DIR_VAR1)
- 文件属性
  - 语法
    - set_property(SOURCE main.cpp　PROPERTY FILE_PRO "FILEPRO001")get_property(var SOURCE main.cpp PROPERTY FILE_PRO)
  - 示例
    - set_property(SOURCE main.cpp　PROPERTY COMPILE_DEFINITIONS "PARA1=1234")

- 目标属性
  - 语法
    - set_property(TARGET ${PROJECT_NAME} PROPERTY OBJ_VAR "TARGET 001")get_property(var TARGET ${PROJECT_NAME} PROPERTY OBJ_VAR)
  - 示例
    - set_property(SOURCE main.cpp PROPERTY COMPILE_DEFINITIONS "PARA1=1234")

打印属性

- include(CMakePrintHelpers)
- cmake_print_properties
  - cmake_print_properties([TARGETS target1 .. targetN] [SOURCES source1 .. sourceN] [DIRECTORIES dir1 .. dirN] [TESTS test1 .. testN] [CACHE_ENTRIES entry1 .. entryN] PROPERTIES prop1 .. propN
  - cmake_print_properties(TARGETS foo bar PROPERTIES LOCATION INTERFACE_INCLUDE_DIRECTORIES)
- cmake_print_variables(var1 var2 .. varN)

CMake 预置属性

- 全局属性

  - 代码

    - ALLOW_DUPLICATE_CUSTOM_TARGETS AUTOGEN_SOURCE_GROUP AUTOGEN_TARGETS_FOLDER AUTOMOC_SOURCE_GROUP AUTOMOC_TARGETS_FOLDER AUTORCC_SOURCE_GROUP AUTOUIC_SOURCE_GROUP CMAKE_C_KNOWN_FEATURES CMAKE_CUDA_KNOWN_FEATURES CMAKE_CXX_KNOWN_FEATURES CMAKE_ROLE DEBUG_CONFIGURATIONS DISABLED_FEATURES ECLIPSE_EXTRA_CPROJECT_CONTENTS ECLIPSE_EXTRA_NATURES ENABLED_FEATURES ENABLED_LANGUAGES FIND_LIBRARY_USE_LIB32_PATHS FIND_LIBRARY_USE_LIB64_PATHS FIND_LIBRARY_USE_LIBX32_PATHS FIND_LIBRARY_USE_OPENBSD_VERSIONING GENERATOR_IS_MULTI_CONFIG GLOBAL_DEPENDS_DEBUG_MODE GLOBAL_DEPENDS_NO_CYCLES IN_TRY_COMPILE JOB_POOLS PACKAGES_FOUND PACKAGES_NOT_FOUND PREDEFINED_TARGETS_FOLDER REPORT_UNDEFINED_PROPERTIES RULE_LAUNCH_COMPILE RULE_LAUNCH_CUSTOM RULE_LAUNCH_LINK RULE_MESSAGES TARGET_ARCHIVES_MAY_BE_SHARED_LIBS TARGET_MESSAGES TARGET_SUPPORTS_SHARED_LIBS USE_FOLDERS XCODE_EMIT_EFFECTIVE_PLATFORM_NAME

  - 示例

- get_property(var GLOBAL PROPERTY GENERATOR_IS_MULTI_CONFIG)message("GENERATOR_IS_MULTI_CONFIG = ${var}")

- 目录属性
  - 代码
    - ADDITIONAL_CLEAN_FILES BINARY_DIR BUILDSYSTEM_TARGETS CACHE_VARIABLES CLEAN_NO_CUSTOM CMAKE_CONFIGURE_DEPENDS COMPILE_DEFINITIONS COMPILE_OPTIONS DEFINITIONS EXCLUDE_FROM_ALL IMPLICIT_DEPENDS_INCLUDE_TRANSFORM IMPORTED_TARGETS INCLUDE_DIRECTORIES INCLUDE_REGULAR_EXPRESSION INTERPROCEDURAL_OPTIMIZATION INTERPROCEDURAL_OPTIMIZATION_<CONFIG> LABELS LINK_DIRECTORIES LINK_OPTIONS LISTFILE_STACK MACROS PARENT_DIRECTORY RULE_LAUNCH_COMPILE RULE_LAUNCH_CUSTOM RULE_LAUNCH_LINK SOURCE_DIR SUBDIRECTORIES TESTS TEST_INCLUDE_FILES VARIABLES VS_GLOBAL_SECTION_POST_<section> VS_GLOBAL_SECTION_PRE_<section> VS_STARTUP_PROJECT
  - 示例
    - add_subdirectory(sub2)get_property(var DIRECTORY . PROPERTY SUBDIRECTORIES)message("SUBDIRECTORIES = ${var}")

- 目标属性

- 代码

    - ADDITIONAL_CLEAN_FILES AIX_EXPORT_ALL_SYMBOLS ALIAS_GLOBAL ALIASED_TARGET ANDROID_ANT_ADDITIONAL_OPTIONS ANDROID_API ANDROID_API_MIN ANDROID_ARCH ANDROID_ASSETS_DIRECTORIES ANDROID_GUI ANDROID_JAR_DEPENDENCIES ANDROID_JAR_DIRECTORIES ANDROID_JAVA_SOURCE_DIR ANDROID_NATIVE_LIB_DEPENDENCIES ANDROID_NATIVE_LIB_DIRECTORIES ANDROID_PROCESS_MAX ANDROID_PROGUARD ANDROID_PROGUARD_CONFIG_PATH ANDROID_SECURE_PROPS_PATH ANDROID_SKIP_ANT_STEP ANDROID_STL_TYPE ARCHIVE_OUTPUT_DIRECTORY ARCHIVE_OUTPUT_DIRECTORY_<CONFIG> ARCHIVE_OUTPUT_NAME ARCHIVE_OUTPUT_NAME_<CONFIG> AUTOGEN_BUILD_DIR AUTOGEN_ORIGIN_DEPENDS AUTOGEN_PARALLEL AUTOGEN_TARGET_DEPENDS AUTOMOC AUTOMOC_COMPILER_PREDEFINES AUTOMOC_DEPEND_FILTERS AUTOMOC_EXECUTABLE AUTOMOC_MACRO_NAMES AUTOMOC_MOC_OPTIONS AUTOMOC_PATH_PREFIX AUTORCC AUTORCC_EXECUTABLE AUTORCC_OPTIONS AUTOUIC AUTOUIC_EXECUTABLE AUTOUIC_OPTIONS AUTOUIC_SEARCH_PATHS BINARY_DIR BUILD_RPATH BUILD_RPATH_USE_ORIGIN BUILD_WITH_INSTALL_NAME_DIR BUILD_WITH_INSTALL_RPATH BUNDLE BUNDLE_EXTENSION C_EXTENSIONS C_STANDARD C_STANDARD_REQUIRED COMMON_LANGUAGE_RUNTIME COMPATIBLE_INTERFACE_BOOL COMPATIBLE_INTERFACE_NUMBER_MAX

COMPATIBLE_INTERFACE_NUMBER_MIN

COMPATIBLE_INTERFACE_STRING    COMPILE_DEFINITIONS

COMPILE_FEATURES    COMPILE_FLAGS    COMPILE_OPTIONS

COMPILE_PDB_NAME    COMPILE_PDB_NAME_<CONFIG>

COMPILE_PDB_OUTPUT_DIRECTORY

COMPILE_PDB_OUTPUT_DIRECTORY_<CONFIG>

<CONFIG>_OUTPUT_NAME    <CONFIG>_POSTFIX

CROSSCOMPILING_EMULATOR    CUDA_ARCHITECTURES

CUDA_EXTENSIONS    CUDA_PTX_COMPILATION

CUDA_RESOLVE_DEVICE_SYMBOLS    CUDA_RUNTIME_LIBRARY

CUDA_SEPARABLE_COMPILATION    CUDA_STANDARD

CUDA_STANDARD_REQUIRED    CXX_EXTENSIONS    CXX_STANDARD

CXX_STANDARD_REQUIRED    DEBUG_POSTFIX    DEFINE_SYMBOL

DEPLOYMENT_ADDITIONAL_FILES    DEPLOYMENT_REMOTE_DIRECTORY

DEPRECATION    DISABLE_PRECOMPILE_HEADERS    DOTNET_SDK

DOTNET_TARGET_FRAMEWORK

DOTNET_TARGET_FRAMEWORK_VERSION    EchoString

ENABLE_EXPORTS    EXCLUDE_FROM_ALL

EXCLUDE_FROM_DEFAULT_BUILD

EXCLUDE_FROM_DEFAULT_BUILD_<CONFIG>

EXPORT_COMPILE_COMMANDS    EXPORT_NAME

EXPORT_PROPERTIES    FOLDER

Fortran_BUILDING_INSTRINSIC_MODULES    Fortran_FORMAT

Fortran_MODULE_DIRECTORY    Fortran_PREPROCESS    FRAMEWORK

FRAMEWORK_MULTI_CONFIG_POSTFIX_<CONFIG>

FRAMEWORK_VERSION    GENERATOR_FILE_NAME

GHS_INTEGRITY_APP    GHS_NO_SOURCE_GROUP_FILE    GNUtoMS

HAS_CXX    HEADER_DIRS    HEADER_DIRS_<NAME>    HEADER_SET

HEADER_SET_<NAME> HEADER_SETS HIP_ARCHITECTURES

HIP_EXTENSIONS HIP_STANDARD HIP_STANDARD_REQUIRED

IMPLICIT_DEPENDS_INCLUDE_TRANSFORM IMPORTED

IMPORTED_COMMON_LANGUAGE_RUNTIME

IMPORTED_CONFIGURATIONS IMPORTED_GLOBAL

IMPORTED_IMPLIB IMPORTED_IMPLIB_<CONFIG>

IMPORTED_LIBNAME IMPORTED_LIBNAME_<CONFIG>

IMPORTED_LINK_DEPENDENT_LIBRARIES

IMPORTED_LINK_DEPENDENT_LIBRARIES_<CONFIG>

IMPORTED_LINK_INTERFACE_LANGUAGES

IMPORTED_LINK_INTERFACE_LANGUAGES_<CONFIG>

IMPORTED_LINK_INTERFACE_LIBRARIES

IMPORTED_LINK_INTERFACE_LIBRARIES_<CONFIG>

IMPORTED_LINK_INTERFACE_MULTIPLICITY

IMPORTED_LINK_INTERFACE_MULTIPLICITY_<CONFIG>

IMPORTED_LOCATION IMPORTED_LOCATION_<CONFIG>

IMPORTED_NO_SONAME IMPORTED_NO_SONAME_<CONFIG>

IMPORTED_NO_SYSTEM IMPORTED_OBJECTS

IMPORTED_OBJECTS_<CONFIG> IMPORTED_SONAME

IMPORTED_SONAME_<CONFIG> IMPORT_PREFIX IMPORT_SUFFIX

INCLUDE_DIRECTORIES INSTALL_NAME_DIR

INSTALL_REMOVE_ENVIRONMENT_RPATH INSTALL_RPATH

INSTALL_RPATH_USE_LINK_PATH INTERFACE_AUTOUIC_OPTIONS

INTERFACE_COMPILE_DEFINITIONS INTERFACE_COMPILE_FEATURES

INTERFACE_COMPILE_OPTIONS INTERFACE_HEADER_SETS

INTERFACE_INCLUDE_DIRECTORIES INTERFACE_LINK_DEPENDS

INTERFACE_LINK_DIRECTORIES INTERFACE_LINK_LIBRARIES

INTERFACE_LINK_OPTIONS

INTERFACE_POSITION_INDEPENDENT_CODE

INTERFACE_PRECOMPILE_HEADERS INTERFACE_SOURCES

INTERFACE_SYSTEM_INCLUDE_DIRECTORIES

INTERPROCEDURAL_OPTIMIZATION

INTERPROCEDURAL_OPTIMIZATION_<CONFIG>

IOS_INSTALL_COMBINED ISPC_HEADER_DIRECTORY

ISPC_HEADER_SUFFIX ISPC_INSTRUCTION_SETS

JOB_POOL_COMPILE JOB_POOL_LINK

JOB_POOL_PRECOMPILE_HEADER LABELS <LANG>_CLANG_TIDY

<LANG>_COMPILER_LAUNCHER <LANG>_CPPCHECK

<LANG>_CPPLINT <LANG>_EXTENSIONS

<LANG>_INCLUDE_WHAT_YOU_USE <LANG>_LINKER_LAUNCHER

<LANG>_STANDARD <LANG>_STANDARD_REQUIRED

<LANG>_VISIBILITY_PRESET LIBRARY_OUTPUT_DIRECTORY

LIBRARY_OUTPUT_DIRECTORY_<CONFIG> LIBRARY_OUTPUT_NAME

LIBRARY_OUTPUT_NAME_<CONFIG> LINK_DEPENDS

LINK_DEPENDS_NO_SHARED LINK_DIRECTORIES LINK_FLAGS

LINK_FLAGS_<CONFIG> LINK_INTERFACE_LIBRARIES

LINK_INTERFACE_LIBRARIES_<CONFIG>

LINK_INTERFACE_MULTIPLICITY

LINK_INTERFACE_MULTIPLICITY_<CONFIG> LINK_LIBRARIES

LINK_LIBRARIES_ONLY_TARGETS LINK_OPTIONS

LINK_SEARCH_END_STATIC LINK_SEARCH_START_STATIC

LINK_WHAT_YOU_USE LINKER_LANGUAGE LOCATION

LOCATION_<CONFIG> MACHO_COMPATIBILITY_VERSION

MACHO_CURRENT_VERSION MACOSX_BUNDLE

MACOSX_BUNDLE_INFO_PLIST MACOSX_FRAMEWORK_INFO_PLIST

MACOSX_RPATH MANUALLY_ADDED_DEPENDENCIES

MAP_IMPORTED_CONFIG_<CONFIG> MSVC_RUNTIME_LIBRARY

NAME NO_SONAME NO_SYSTEM_FROM_IMPORTED

OBJC_EXTENSIONS OBJC_STANDARD OBJC_STANDARD_REQUIRED

OBJCXX_EXTENSIONS OBJCXX_STANDARD

OBJCXX_STANDARD_REQUIRED OPTIMIZE_DEPENDENCIES

OSX_ARCHITECTURES OSX_ARCHITECTURES_<CONFIG>

OUTPUT_NAME OUTPUT_NAME_<CONFIG> PCH_WARN_INVALID

PCH_INSTANTIATE_TEMPLATES PDB_NAME

PDB_NAME_<CONFIG> PDB_OUTPUT_DIRECTORY

PDB_OUTPUT_DIRECTORY_<CONFIG> POSITION_INDEPENDENT_CODE

PRECOMPILE_HEADERS PRECOMPILE_HEADERS_REUSE_FROM

PREFIX PRIVATE_HEADER PROJECT_LABEL PUBLIC_HEADER

RESOURCE RULE_LAUNCH_COMPILE RULE_LAUNCH_CUSTOM

RULE_LAUNCH_LINK RUNTIME_OUTPUT_DIRECTORY

RUNTIME_OUTPUT_DIRECTORY_<CONFIG> RUNTIME_OUTPUT_NAME

RUNTIME_OUTPUT_NAME_<CONFIG> SKIP_BUILD_RPATH

SOURCE_DIR SOURCES SOVERSION STATIC_LIBRARY_FLAGS

STATIC_LIBRARY_FLAGS_<CONFIG> STATIC_LIBRARY_OPTIONS

SUFFIX Swift_DEPENDENCIES_FILE Swift_LANGUAGE_VERSION

Swift_MODULE_DIRECTORY Swift_MODULE_NAME TYPE

UNITY_BUILD UNITY_BUILD_BATCH_SIZE

UNITY_BUILD_CODE_AFTER_INCLUDE

UNITY_BUILD_CODE_BEFORE_INCLUDE UNITY_BUILD_MODE

UNITY_BUILD_UNIQUE_ID VERSION VISIBILITY_INLINES_HIDDEN

VS_CONFIGURATION_TYPE VS_DEBUGGER_COMMAND

VS_DEBUGGER_COMMAND_ARGUMENTS

VS_DEBUGGER_ENVIRONMENT VS_DEBUGGER_WORKING_DIRECTORY

VS_DESKTOP_EXTENSIONS_VERSION

VS_DOTNET_DOCUMENTATION_FILE

VS_DOTNET_REFERENCE_<refname>

VS_DOTNET_REFERENCEPROP_<refname>_TAG_<tagname>

VS_DOTNET_REFERENCES VS_DOTNET_REFERENCES_COPY_LOCAL

VS_DOTNET_TARGET_FRAMEWORK_VERSION VS_DPI_AWARE

VS_GLOBAL_KEYWORD VS_GLOBAL_PROJECT_TYPES

VS_GLOBAL_ROOTNAMESPACE VS_GLOBAL_<variable>

VS_IOT_EXTENSIONS_VERSION VS_IOT_STARTUP_TASK

VS_JUST_MY_CODE_DEBUGGING VS_KEYWORD

VS_MOBILE_EXTENSIONS_VERSION VS_NO_SOLUTION_DEPLOY

VS_PACKAGE_REFERENCES VS_PLATFORM_TOOLSET

VS_PROJECT_IMPORT VS_SCC_AUXPATH VS_SCC_LOCALPATH

VS_SCC_PROJECTNAME VS_SCC_PROVIDER VS_SDK_REFERENCES

VS_SOLUTION_DEPLOY VS_SOURCE_SETTINGS_<tool>

VS_USER_PROPS VS_WINDOWS_TARGET_PLATFORM_MIN_VERSION

VS_WINRT_COMPONENT VS_WINRT_EXTENSIONS

VS_WINRT_REFERENCES WIN32_EXECUTABLE

WINDOWS_EXPORT_ALL_SYMBOLS XCODE_ATTRIBUTE_<an-attribute>

XCODE_EMBED_FRAMEWORKS_CODE_SIGN_ON_COPY

XCODE_EMBED_FRAMEWORKS_REMOVE_HEADERS_ON_COPY

XCODE_EMBED_<type>

XCODE_EMBED_<type>_CODE_SIGN_ON_COPY

XCODE_EMBED_<type>_PATH

XCODE_EMBED_<type>_REMOVE_HEADERS_ON_COPY

XCODE_EXPLICIT_FILE_TYPE XCODE_GENERATE_SCHEME

XCODE_LINK_BUILD_PHASE_MODE XCODE_PRODUCT_TYPE

XCODE_SCHEME_ADDRESS_SANITIZER

XCODE_SCHEME_ADDRESS_SANITIZER_USE_AFTER_RETURN

XCODE_SCHEME_ARGUMENTS　　XCODE_SCHEME_DEBUG_AS_ROOT

XCODE_SCHEME_DEBUG_DOCUMENT_VERSIONING

XCODE_SCHEME_ENABLE_GPU_FRAME_CAPTURE_MODE

XCODE_SCHEME_DISABLE_MAIN_THREAD_CHECKER

XCODE_SCHEME_DYNAMIC_LIBRARY_LOADS

XCODE_SCHEME_DYNAMIC_LINKER_API_USAGE

XCODE_SCHEME_ENVIRONMENT　　XCODE_SCHEME_EXECUTABLE

XCODE_SCHEME_GUARD_MALLOC

XCODE_SCHEME_MAIN_THREAD_CHECKER_STOP

XCODE_SCHEME_MALLOC_GUARD_EDGES

XCODE_SCHEME_MALLOC_SCRIBBLE

XCODE_SCHEME_MALLOC_STACK

XCODE_SCHEME_THREAD_SANITIZER

XCODE_SCHEME_THREAD_SANITIZER_STOP

XCODE_SCHEME_UNDEFINED_BEHAVIOUR_SANITIZER

XCODE_SCHEME_UNDEFINED_BEHAVIOUR_SANITIZER_STOP

XCODE_SCHEME_WORKING_DIRECTORY

XCODE_SCHEME_ZOMBIE_OBJECTS　　XCTEST

- 示例
    - BINARY_DIR
- 源码属性
    - 代码
        - ABSTRACT　　AUTORCC_OPTIONS　　AUTOUIC_OPTIONS COMPILE_DEFINITIONS　　COMPILE_FLAGS　　COMPILE_OPTIONS

EXTERNAL_OBJECT Fortran_FORMAT Fortran_PREPROCESS

GENERATED HEADER_FILE_ONLY INCLUDE_DIRECTORIES

KEEP_EXTENSION LABELS LANGUAGE LOCATION

MACOSX_PACKAGE_LOCATION OBJECT_DEPENDS

OBJECT_OUTPUTS SKIP_AUTOGEN SKIP_AUTOMOC

SKIP_AUTORCC SKIP_AUTOUIC SKIP_PRECOMPILE_HEADERS

SKIP_UNITY_BUILD_INCLUSION Swift_DEPENDENCIES_FILE

Swift_DIAGNOSTICS_FILE SYMBOLIC UNITY_GROUP

VS_COPY_TO_OUT_DIR VS_CSHARP_<tagname>

VS_DEPLOYMENT_CONTENT VS_DEPLOYMENT_LOCATION

VS_INCLUDE_IN_VSIX VS_RESOURCE_GENERATOR VS_SETTINGS

VS_SHADER_DISABLE_OPTIMIZATIONS VS_SHADER_ENABLE_DEBUG

VS_SHADER_ENTRYPOINT VS_SHADER_FLAGS VS_SHADER_MODEL

VS_SHADER_OBJECT_FILE_NAME VS_SHADER_OUTPUT_HEADER_FILE

VS_SHADER_TYPE VS_SHADER_VARIABLE_NAME

VS_TOOL_OVERRIDE VS_XAML_TYPE WRAP_EXCLUDE

XCODE_EXPLICIT_FILE_TYPE XCODE_FILE_ATTRIBUTES

XCODE_LAST_KNOWN_FILE_TYPE

- 示例

  - COMPILE_DEFINITIONS
  - COMPILE_FLAGS
  - INCLUDE_DIRECTORIES
  - OBJECT_OUTPUTS

## 3.4. 环境变量

环境变量语法

- set(ENV{<variable>} [<value>])
- $ENV{<variable>}

环境变量特性

- 只影响当前的 CMake 进程，不影响调用 CMake 的进程，也不影响整个系统环境，也不影响后续构建或测试进程的环境。
- 环境变量与全局属性
  - 基本类似 全局属性可以加说明
  - 环境变量访问简单
- Environment Variables are like ordinary Variables, with the following differences:Scope Environment variables have global scope in a CMake process. They are never cached.

环境变量类型

- cmake 预置
  - CMAKE_APPLE_SILICON_PROCESSOR CMAKE_BUILD_PARALLEL_LEVEL CMAKE_BUILD_TYPE CMAKE_CONFIGURATION_TYPES CMAKE_CONFIG_TYPE CMAKE_EXPORT_COMPILE_COMMANDS CMAKE_GENERATOR CMAKE_GENERATOR_INSTANCE CMAKE_GENERATOR_PLATFORM CMAKE_GENERATOR_TOOLSET CMAKE_INSTALL_MODE

CMAKE_<LANG>_COMPILER_LAUNCHER

CMAKE_<LANG>_LINKER_LAUNCHER　　CMAKE_MSVCIDE_RUN_PATH

CMAKE_NO_VERBOSE　　CMAKE_OSX_ARCHITECTURES

CMAKE_TOOLCHAIN_FILE　　DESTDIR　　LDFLAGS

MACOSX_DEPLOYMENT_TARGET　　<PackageName>_ROOT　　VERBOSE

- ASM<DIALECT>　　ASM<DIALECT>FLAGS　　CC　　CFLAGS
CSFLAGS　　CUDAARCHS　　CUDACXX　　CUDAFLAGS　　CUDAHOSTCXX
CXX　　CXXFLAGS　　FC　　FFLAGS　　HIPCXX　　HIPFLAGS　　ISPC
ISPCFLAGS　　OBJC　　OBJCXX　　RC　　RCFLAGS　　SWIFTC

- 自定义环境变量

- 系统变量

## 3.5 cmake math 数学运算

math(EXPR <variable> "<expression>" [OUTPUT_FORMAT <format>])

"5 * (10 + 13)".  支持 +, -, *, /, %, |, &, ^, ~, <<, >>

结果必须是 64 位有符号整数

输出格式

- HEXADECIMAL
    - 0x
        - 0x3e8

- DECIMAL
    - 十进制数

## 3.6 cmake string 字符串处理

语法

- 搜索和替换
    - string(FIND <string> <substring> <out-var> [...])
    - string(REPLACE <match-string> <replace-string> <out-var> <input>...)
    - string(REGEX MATCH <match-regex> <out-var> <input>...)
    - string(REGEX MATCHALL <match-regex> <out-var> <input>...)
    - string(REGEX REPLACE <match-regex> <replace-expr> <out-var> <input>...)
- 操作
    - string(APPEND <string-var> [<input>...])
    - string(PREPEND <string-var> [<input>...])
    - string(CONCAT <out-var> [<input>...])
    - string(JOIN <glue> <out-var> [<input>...])
    - string(TOLOWER <string> <out-var>)

  - string(TOUPPER <string> <out-var>)
  - string(LENGTH <string> <out-var>)
  - string(SUBSTRING <string> <begin> <length> <out-var>)
  - string(STRIP <string> <out-var>)
  - string(GENEX_STRIP <string> <out-var>)
  - string(REPEAT <string> <count> <out-var>)
- 比较
  - string(COMPARE <op> <string1> <string2> <out-var>)
- 哈希值
  - string(<HASH> <out-var> <input>)
- 生成
  - string(ASCII <number>... <out-var>)
  - string(HEX <string> <out-var>)
  - string(CONFIGURE <string> <out-var> [...])
  - string(MAKE_C_IDENTIFIER <string> <out-var>)
  - string(RANDOM [<option>...] <out-var>)

- string(TIMESTAMP <out-var> [<format string>] [UTC])
- string(UUID <out-var> ...)

- JSON
  - string(JSON <out-var> [ERROR_VARIABLE <error-var>] {GET | TYPE | LENGTH | REMOVE} <json-string> <member|index> [<member|index> ...])
  - string(JSON <out-var> [ERROR_VARIABLE <error-var>] MEMBER <json-string> [<member|index> ...] <index>)
  - string(JSON <out-var> [ERROR_VARIABLE <error-var>] SET <json-string> <member|index> [<member|index> ...] <value>)
  - string(JSON <out-var> [ERROR_VARIABLE <error-var>] EQUAL <json-string1> <json-string2>)

## 3.7. list 基础语法

set(srcs a.c b.c c.c) # sets "srcs" to "a.c;b.c;c.c"

CMake 中存储所有值都是字符串，有”；"间隔符的字符串被拆分为列表

set(x a "b;c") # sets "x" to "a;b;c", not "a;b\;c"

语法

- Reading list(LENGTH <list> <out-var>) list(GET <list> <element index> [<index> ...] <out-var>) list(JOIN <list> <glue> <out-var>) list(SUBLIST <list> <begin> <length> <out-var>)Search list(FIND <list> <value>

<out-var>)Modification list(APPEND <list> [<element>...]) list(FILTER <list> {INCLUDE | EXCLUDE} REGEX <regex>) list(INSERT <list> <index> [<element>...]) list(POP_BACK <list> [<out-var>...]) list(POP_FRONT <list> [<out-var>...]) list(PREPEND <list> [<element>...]) list(REMOVE_ITEM <list> <value>...) list(REMOVE_AT <list> <index>...) list(REMOVE_DUPLICATES <list>) list(TRANSFORM <list> <ACTION> [...])Ordering list(REVERSE <list>) list(SORT <list> [...])

code

- set(src "a" "b" "c;d")list(APPEND src "e")list(APPEND src "f")list(APPEND src "ca1")list(APPEND src "ca2")list(APPEND src "test")message("src = ${src}")#list(APPEND ENV{PATH} "/code")#message($ENV{PATH})list(LENGTH src length)message("src length ${length}")# list(GET <list> <element index> [<element index> ...] <output variable>)list(GET src 1 var)message("src 1 = ${var}")list(GET src 12 var)message("src 12 = ${var}")list(GET src -1 var)message("src -1 = ${var}")list(GET src -2 var)message("src -2 = ${var}")#list(JOIN <list> <glue> <output variable>)#a|b|c|d|e|flist(JOIN src "|" var)message("JOIN = ${var}")list(JOIN src "" var)message("JOIN = ${var}")#list(SUBLIST <list> <begin> <length> <output variable>)list(SUBLIST src 0 3 var)message("SUBLIST = ${var}")#list(FIND <list> <value> <output variable>)#全字匹配 list(FIND src "ca1" var)message("FIND = ${var}")# list(INSERT <list> <element_index> <element> [<element> ...])list(INSERT src 1 "ff")list(INSERT src 3 "ff")message("src = ${src}")list(POP_BACK src var)# list(POP_BACK <list> [<out-var>...])message("POP_BACK = ${var}")# list(POP_FRONT <list> [<out-var>...])list(POP_FRONT src var)message("POP_FRONT = ${var}")message("src = ${src}")# list(SORT <list> [COMPARE <compare>] [CASE <case>] [ORDER <order>])#[[使用 COMPARE 关键字选择排序的比较方法。该<compare>选项应该是以下之一：STRING：按字母顺序对字符串列表进行排

序。COMPARE 如果未给出该选项，这是默认行为。FILE_BASENAME：按文件的基本名称对文件的路径名列表进行排序。NATURAL：使用自然顺序对字符串列表进行排序（参见 strverscmp(3)手册），即将连续数字作为整数进行比较。例如：以下列表 10.0 1.1 2.1 8.0 2.0 3.1 如果 选择了比较，则将 排序为 1.1 2.0 2.1 3.1 8.0 10.0 ，与比较将排序为 1.1 10.0 2.0 2.1 3.1 8.0 。NATURALSTRINGCASE 关键字选择区分大小写或不区分大小写的排序模式。该<case>选项应该是以下之一：SENSITIVE：列表项以区分大小写的方式排序。CASE 如果未给出该选项，这是默认行为。INSENSITIVE：列表项不区分大小写。未指定仅大写/小写不同的项目的顺序。要控制排序顺序，ORDER 可以给出关键字。该<order>选项应该是以下之一：ASCENDING：按升序对列表进行排序。ORDER 这是未给出选项时的默认行为。DESCENDING：按降序对列表进行排序]]list(SORT src )message("SORT src = ${src}")#[[list(REMOVE_ITEM <list> <value> [<value> ...])]]list(REMOVE_DUPLICATES src)message("REMOVE_DUPLICATES src = ${src}")list(REMOVE_ITEM src f)message("REMOVE_ITEM f src = ${src}")list(REMOVE_AT src 2)message("REMOVE_AT 2 src = ${src}")

## 3.8. CMake foreach 循环语句

语法

- foreach(<loop_var> <items>)　<commands>endforeach()

RANGE

- foreach(<loop_var> RANGE <stop>)
  - 0,1,2,3...
- foreach(<loop_var> RANGE <start> <stop> [<step>])

IN

- LISTS
  - foreach(<loop_var> IN [LISTS [<lists>]] )
- ITEMS
  - foreach(<loop_var> IN [ITEMS [<items>]])
  - list 的取值
    - ${list}
- ZIP_LISTS
  - foreach(<loop_var>... IN ZIP_LISTS <lists>)
  - 3.17 中的新功能。
  - foreach(num IN ZIP_LISTS arr1 arr2)　　message(STATUS "num_0=${num_0}, num_1=${num_1}")endforeach()foreach(v1 v2 IN ZIP_LISTS arr1 arr2)　　message(STATUS "v1=${v1}, v2=${v2}")endforeach()

break()

- if(var GREATER 50)　　　　break()　　endif()

continue()

- if(NOT re)　　　message(${var})　　　continue()　　endif()

code

- foreach(var RANGE 100) #string(APPEND out ${var} " ") math(EXPR re "${var} % 3") if(NOT re) message(${var}) continue() endif() if(var GREATER 50) break() endif() message(".")endforeach()message("end for")

- #[[foreach(<loop_var> <items>) <commands>endforeach()]]#foreach(<loop_var> RANGE <stop>)# var 0 , 1, 2 ,3 ,4 ..10string(out "")foreach(var RANGE 10) string(APPEND out ${var} " ") message(${var})endforeach()message("out = ${out}")# foreach(<loop_var> RANGE <start> <stop> [<step>])foreach(var RANGE 0 10 2) #string(APPEND out ${var} " ") message(${var})endforeach()# foreach(<loop_var> IN [LISTS [<lists>]] [ITEMS [<items>]])set(args a b c d e)foreach(var IN LISTS args)message(${var})endforeach()set(A 0;1)set(B 2 3)set(C "4 5")set(D 6;7 8)set(E "")foreach(X IN LISTS A B C D E) message(STATUS "X=${X}")endforeach()list(APPEND English one two three four)list(APPEND Bahasa satu dua tiga)# 同步遍历两组数组 foreach(num IN ZIP_LISTS English Bahasa) message(STATUS "num_0=${num_0}, num_1=${num_1}")endforeach()foreach(en ba IN ZIP_LISTS English Bahasa) message(STATUS "en=${en}, ba=${ba}")endforeach()

## 3.9. CMake while 循环语句

while(<condition>) <commands>endwhile()

code

- while(var) message(${var}) math(EXPR var "${var}+1") if(var GREATER 100) set(var 0) endif()endwhile()

## 3.10 CMake 宏

### 基本语法

- macro(foo)　<commands>endmacro()
- 宏名称大小写不敏感
  - foo()Foo()FOO()cmake_language(CALL foo)

### 普通参数

- 必需的参数
  - macro(foo arg1 arg2)
- ARGC
  - 参数个数
- ARGN
  - 参数数组
- ARGV0　　ARGV1　ARGV2
- 参数不是变量
  - 无法使用如下代码 if(ARGV1) if(DEFINED ARGV2) if(ARGC GREATER 2)foreach(loop_var IN LISTS ARGN)

- 如果在调用宏的范围内有一个同名的变量，则使用未引用的名称将使用现有变量而不是参数

属性式参数

- cmake_parse_arguments
  - cmake_parse_arguments(<prefix> <options> <one_value_keywords> <multi_value_keywords> <args>...)
  - <prefix>
    - 生成变量的前缀
  - options
    - 设置了就是 TRUE 没有设置就是 FALSE  不用赋值
  - one_value_keywords
    - 单个值的变量
  - multi_value_keywords
    - 多个值的变量
  - _UNPARSED_ARGUMENTS
    - 传递了错误的值
  - _KEYWORDS_MISSING_VALUES
    - 没有设定值

- code

  - macro(mfun)set(re "001")message("in macro tmp = ${tmp}")endmacro()function(fun)message("in function tmp = ${tmp}")set(re "fun re")endfunction()set(tmp "003")fun()message("re = ${re}")mfun()message("re = ${re}")macro(my_install) set(options OPTIONAL FAST) set(oneValueArgs DESTINATION RENAME) set(multiValueArgs TARGETS CONFIGURATIONS) cmake_parse_arguments("" "${options}" "${oneValueArgs}" "${multiValueArgs}" ${ARGN} ) message("ARGN = ${ARGN}") message("MY_INSTALL_OPTIONAL = ${_OPTIONAL}") message("TARGETS = ${_TARGETS}") message("DESTINATION = ${_DESTINATION}") message("RENAME = ${_RENAME}") message("FAST = ${_FAST}") message("_UNPARSED_ARGUMENTS = ${_UNPARSED_ARGUMENTS}") message("_KEYWORDS_MISSING_VALUES = ${_KEYWORDS_MISSING_VALUES}")endmacro()my_install(TARGETS foo bar DESTINATION bin OPTIONAL CONFIGURATIONS)

- my_macro(TARGETS foo bar DESTINATION bin )

code

- macro(foo) set(foo_var "foovar") #ARGN, ARGC,ARGV 等 ARGV0 不是变量 # 通 常 宏 使 用 全 小 写 的 名 称 message(" ${ARGC} ${ARGV} " ) message("ARGV0 = ${ARGV0}") message("ARGV1 = ${ARGV1}") message("ARGV2 = ${ARGV2}") message("ARGV3 = ${ARGV3}") message("macro(foo)") message("para1 = ${para1}") foreach(arg IN LISTS ARGN) message("arg = ${arg}") endforeach()endmacro()foo(1)Foo(33)FOO(44 "tt" 111)

## 3.11 CMake 函数

函数的参数是变量

函数内部设置的普通变量作用域只在函数内

- set(fun_var2 "fun2 var value" PARENT_SCOPE)

函数可以用 return 返回

- return()
  - 宏是在原地展开的，因此无法处理 return
  - 从函数、目录或文件返回

code

- function(fun arg1 arg2)# 通常函数使用全小写的名称 set(fun_var "fun var value")set(fun_var2 "fun2 var value" PARENT_SCOPE)message("call fun")message(" ${ARGC} ${ARGV} " )message("ARGV0 = ${ARGV0}")message("ARGV1 = ${ARGV1}")message("ARGV2 = ${ARGV2}")message("ARGV3 = ${ARGV3}")endif()

- set(testm "001")macro(TestM) set(testm "002")endmacro()function(TestF) set(testm "003")endfunction()TestM()TestF()

# 第四章 简化 cmake 的多版本配置语法-生成器表达式

# 301cmake_exp

## 什么是表达式

替代复杂的 if

在构建系统生成期间评估生成器表达式以生成特定于每个构建配置的信息

大多数 CMake 命令在配置的时候执行

如果你想要他们在构建或者安装的时候运行呢

## 使用场景

修改目标配置

- target_link_libraries
- 例如 LINK_LIBRARIES,INCLUDE_DIRECTORIES, COMPILE_DEFINITIONS
- 例如 target_link_libraries(), target_include_directories(),target_compile_definitions()

## 布尔生成器表达式

逻辑运算符

- $<BOOL:string>

  - 转换 string 为 0 或 1。评估 0 以下任何一项是否为真：string 是空的，string 是不区分大小写的等于 0, FALSE, OFF, N, NO, IGNORE, or NOTFOUND, orstring 以后缀结尾-NOTFOUND（区分大小写）。否则计算为 1。

  - target_compile_definitions(testexp PUBLIC "$<$<BOOL:${LIB}>:LIBSTR=123>")

  - $<condition:true_string>

- $<NOT:condition>

  - target_compile_definitions(testexp PUBLIC "$<$<NOT:$<BOOL:${BUILD_SHARED_LIBS}>>:STATIC>")

- $<AND:conditions>

  - target_include_directories(testexp PUBLIC "$<AND:$<BOOL:${XFILE}>,$<BOOL:OFF>>")

- $<OR:conditions>

  - target_include_directories(testexp PUBLIC "$<NOT:$<OR:$<AND:1,1>,0>>")

字符串比较

- $<STREQUAL:string1,string2>

  - $<STREQUAL:${CMAKE_BUILD_TYPE},Debug>

  - target_include_directories(testexp PUBLIC "$<STREQUAL:string1,string1>")

- $<EQUAL:value1,value2>

- $<EQUAL:123,123>
- target_include_directories(testexp PUBLIC "$<EQUAL:1,12>")

变量查询

- $<CONFIG:cfgs>
  - 如果 config 是 cfgs 逗号分隔的列表中的任何一项为 1，否则 0。比较不区分大小写
  - Degbug Release 章节实践
- $<PLATFORM_ID:platform_ids>
  - CMake 的平台 ID
    - CMAKE_SYSTEM_NAME
  - message("CMAKE_SYSTEM_NAME = ${CMAKE_SYSTEM_NAME}")target_include_directories(testexp PUBLIC $<PLATFORM_ID:Windows,Linux> )

## 字符串值生成器表达式

条件表达式

- $<condition:true_string>
- $<IF:condition,true_string,false_string>

字符串转换

- $<LOWER_CASE:string>
- $<UPPER_CASE:string>

变量查询

- $<CONFIG>
- $<PLATFORM_ID>

目标相关查询

- 查询引用一个目标 tgt。可以是任何运行时目标，如：由 add_executable()创建的可执行目标由 add_library()创建的共享库目标（.so，.dll 但不是它们的.lib 导入库）由 add_library()创建的静态库目标
- $<TARGET_NAME_IF_EXISTS:tgt>
- $<TARGET_FILE:tgt>
  - Full path to the tgt binary file.
- $<TARGET_PROPERTY:tgt,prop>
  - Value of the property prop on the target tgt.

## Debugging

add_custom_target(genexdebug COMMAND ${CMAKE_COMMAND} -E echo "$<...>")

set_target_properties(test PROPERTIES VS_DEBUGGER_WORKING_DIRECTORY $<TARGET_FILE_DIR:test>)

# 第五章 CMake 跨平台 c++编译特性设置

## cmake 构建参数

### target_include_directories 包含目录详解

- 基本语法
  - target_include_directories(<target> [SYSTEM] [AFTER|BEFORE] <INTERFACE|PUBLIC|PRIVATE> [items1...] [<INTERFACE|PUBLIC|PRIVATE> [items2...] ...])
  - SYSTEM
    - 告诉编译器路径是可能是系统路径，解决一些平台的警告信息
  - AFTER
    - 最后
  - BEFORE
    - 前面
- 命令概述
  - 指定目标要使用的包含目录
  - 名称<target>必须是由命令创建的，不能是 Alias Targets 别名目标
  - 例如 add_executable()或者 add_library()

- 参数流转
    - INTERFACE
        - 只有依赖者引用
        - INTERFACE_INCLUDE_DIRECTORIES
    - PUBLIC
        - 依赖者和自己都引用
        - INCLUDE_DIRECTORIES
        - INTERFACE_INCLUDE_DIRECTORIES
    - PRIVATE
        - 只有自己用
        - INCLUDE_DIRECTORIES

target_link_libraries 导入依赖库

- target_link_libraries(<target> <PRIVATE|PUBLIC|INTERFACE> <item>... [<PRIVATE|PUBLIC|INTERFACE> <item>...]...)
- $<TARGET_OBJECTS:xlog> 依赖的头文件路径、宏定义等没有
- INTERFACE

  - 只有依赖者引用

- PUBLIC
  - 依赖者和自己都引用
- PRIVATE
  - 只有自己用
- code
  - file(WRITE a.cpp [=[ void A(){}]=])file(WRITE b.cpp [=[ void B(){}]=])file(WRITE c.cpp [=[ void C(){}]=])file(WRITE d.cpp [=[ void D(){}]=])file(WRITE main.cpp [=[ int main(){return 0;}]=])add_library(A a.cpp)target_include_directories(A PUBLIC "A_INCLUDE")target_include_directories(A PRIVATE "A_PRIVATE")target_include_directories(A INTERFACE "A_INTERFACE")add_library(B b.cpp)target_include_directories(B PUBLIC "B_INCLUDE")add_library(C c.cpp)target_include_directories(C PUBLIC "C_INCLUDE")add_library(D d.cpp)target_include_directories(C PUBLIC "D_INCLUDE")target_link_libraries(D PUBLIC A)target_link_libraries(D PRIVATE B)target_link_libraries(D INTERFACE C)add_executable(main main.cpp)target_link_libraries(main PRIVATE D)

target_compile_definitions()编译传递宏

- COMPILE_DEFINITIONS
- INTERFACE_COMPILE_DEFINITIONS

- target_compile_definitions(foo PUBLIC FOO)target_compile_definitions(foo PUBLIC -DFOO)  # -D removedtarget_compile_definitions(foo PUBLIC "" FOO) # "" ignoredtarget_compile_definitions(foo PUBLIC -D FOO) # -D becomes "", then ignored

- target_compile_definitions(<target> <INTERFACE|PUBLIC|PRIVATE> [items1...] [<INTERFACE|PUBLIC|PRIVATE> [items2...] ...])

target_compile_features c++ 11 14 17 20 22

- target_compile_features(<target> <PRIVATE|PUBLIC|INTERFACE> <feature> [...])

- foreach(var IN LISTS CMAKE_CXX_COMPILE_FEATURES) message(${var})endforeach()

- vs2022

  - cxx_std_98cxx_template_template_parameterscxx_std_11cxx_alias_templatescxx_alignascxx_alignofcxx_attributescxx_auto_typecxx_constexprcxx_decltypecxx_decltype_incomplete_return_typescxx_default_function_template_argscxx_defaulted_functionscxx_defaulted_move_initializerscxx_delegating_constructorscxx_deleted_functionscxx_enum_forward_declarationscxx_explicit_conversionscxx_extended_friend_declarationscxx_extern_templatescxx_finalcxx_func_identifiercxx_generalized_initializerscxx_inheriting_constructorscxx_inline_namespacescxx_lambdascxx_local_type_template_argscxx_long_long_typecxx_noexceptcxx_nonstatic_member_initcxx_nullptrcxx_overridecxx_range_forcxx_raw_string_literalscxx_reference_qualified_functionscxx_right_angle_bracketscxx_rvalue_referencescxx_sizeof_membercxx_static_assertcxx_strong_enumscxx_thread_localcxx_trailing_return_typescxx_unicode_literalscxx_uniform_initi alizationcxx_unrestricted_unionscxx_user_literalscxx_variadic_macroscxx_variadic_templatescxx_std_14cxx_aggregate_default_initializerscxx_attribute_deprecatedcxx_binary_literalscxx_contextual_conversionscxx_decltype_autocxx_digit_separatorscxx_generic_lambdascxx_lambda_init_capturescxx_relaxed_constexprcxx_return_type_deductioncxx_variable_templatescxx_std_17cxx_std_20cxx_std_23

- gcc
  -

    cxx_std_98cxx_template_template_parameterscxx_std_11cxx_alias_templatescxx_alignascxx_alignofcxx_attributescxx_auto_typecxx_constexprcxx_decltypecxx_decltype_incomplete_return_typescxx_default_function_template_argscxx_defaulted_functionscxx_defaulted_move_initializerscxx_delegating_constructorscxx_deleted_functionscxx_enum_forward_declarationscxx_explicit_conversionscxx_extended_friend_declarationscxx_extern_templatescxx_finalcxx_func_identifiercxx_generalized_initializerscxx_inheriting_constructorscxx_inline_namespacescxx_lambdascxx_local_type_template_argscxx_long_long_typecxx_noexceptcxx_nonstatic_member_initcxx_nullptrcxx_overridecxx_range_forcxx_raw_string_literalscxx_reference_qualified_functionscxx_right_angle_bracketscxx_rvalue_referencescxx_sizeof_membercxx_static_assertcxx_strong_enumscxx_thread_localcxx_trailing_return_typescxx_unicode_literalscxx_uniform_initializationcxx_unrestricted_unionscxx_user_literalscxx_variadic_macroscxx_variadic_templatescxx_std_14cxx_aggregate_default_initializerscxx_attribute_deprecatedcxx_binary_literalscxx_contextual_conversionscxx_decltype_autocxx_digit_separatorscxx_generic_lambdascxx_lambda_init_capturescxx_relaxed_constexprcxx_return_type_deductioncxx_variable_templatescxx_std_17cxx_std_20

- 属性说明

- cxx_std_98Compiler mode is at least C++ 98.cxx_std_11Compiler mode is at least C++ 11.cxx_std_14Compiler mode is at least C++ 14.cxx_std_17Compiler mode is at least C++ 17.cxx_std_20New in version 3.12.Compiler mode is at least C++ 20.cxx_std_23New in version 3.20.Compiler mode is at least C++ 23.Note If the compiler's default standard level is at least that of the requested feature, CMake may omit the -std= flag. The flag may still be added if the compiler's default extensions mode does not match the <LANG>_EXTENSIONS target property, or if the <LANG>_STANDARD target property is set.Low level individual compile featuresFor C++ 11 and C++ 14, compilers were sometimes slow to implement certain language features. CMake provided some individual compile features to help projects determine whether specific features were available. These individual features are now less relevant and projects should generally prefer to use the high level meta features instead. Individual compile features are not provided for C++ 17 or later.See the cmake-compile-features(7) manual for further discussion of the use of individual compile features.Individual features from C++ 98cxx_template_template_parametersTemplate template parameters, as defined in ISO/IEC 14882:1998.Individual features from C++ 11cxx_alias_templatesTemplate aliases, as defined in N2258.cxx_alignasAlignment control alignas, as defined in N2341.cxx_alignofAlignment control alignof, as defined in N2341.cxx_attributesGeneric attributes, as defined in N2761.cxx_auto_typeAutomatic type deduction, as defined in N1984.cxx_constexprConstant expressions, as defined in N2235.cxx_decltype_incomplete_return_typesDecltype on incomplete return types, as defined in N3276.cxx_decltypeDecltype, as defined in N2343.cxx_default_function_template_argsDefault template arguments for function templates, as defined in DR226cxx_defaulted_functionsDefaulted functions, as defined in N2346.cxx_defaulted_move_initializersDefaulted move initializers, as defined in N3053.cxx_delegating_constructorsDelegating constructors, as defined in N1986.cxx_deleted_functionsDeleted functions, as defined in N2346.cxx_enum_forward_declarationsEnum forward declarations, as defined in N2764.cxx_explicit_conversionsExplicit conversion operators, as defined in N2437.cxx_extended_friend_declarationsExtended friend declarations, as defined in N1791.cxx_extern_templatesExtern templates, as defined in N1987.cxx_finalOverride control final keyword, as defined in N2928, N3206 and N3272.cxx_func_identifierPredefined __func__ identifier, as defined in N2340.cxx_generalized_initializersInitializer lists, as defined in N2672.cxx_inheriting_constructorsInheriting constructors, as defined in N2540.cxx_inline_namespacesInline namespaces, as defined in N2535.cxx_lambdasLambda functions, as defined in N2927.cxx_local_type_template_argsLocal and unnamed types as template arguments, as defined in N2657.cxx_long_long_typelong long type, as defined in N1811.cxx_noexceptException specifications, as defined in N3050.cxx_nonstatic_member_initNon-static data member initialization, as defined in N2756.cxx_nullptrNull pointer, as defined in N2431.cxx_overrideOverride control override keyword, as defined in N2928, N3206 and N3272.cxx_range_forRange-based for, as defined in N2930.cxx_raw_string_literalsRaw string literals, as defined in N2442.cxx_reference_qualified_functionsReference qualified functions, as defined in N2439.cxx_right_angle_bracketsRight angle bracket parsing, as defined in N1757.cxx_rvalue_referencesR-value references, as defined in N2118.cxx_sizeof_memberSize of non-static data members, as defined in N2253.cxx_static_assertStatic assert, as defined in N1720.cxx_strong_enumsStrongly typed enums, as defined in N2347.cxx_thread_localThread-local variables, as defined in

N2659.cxx_trailing_return_typesAutomatic function return type, as defined in
N2541.cxx_unicode_literalsUnicode string literals, as defined in
N2442.cxx_uniform_initializationUniform initialization, as defined in
N2640.cxx_unrestricted_unionsUnrestricted unions, as defined in
N2544.cxx_user_literalsUser-defined literals, as defined in
N2765.cxx_variadic_macrosVariadic macros, as defined in
N1653.cxx_variadic_templatesVariadic templates, as defined in
N2242.Individual features from C++
14cxx_aggregate_default_initializersAggregate default initializers, as defined in
N3605.cxx_attribute_deprecated[[deprecated]] attribute, as defined in
N3760.cxx_binary_literalsBinary literals, as defined in
N3472.cxx_contextual_conversionsContextual conversions, as defined in
N3323.cxx_decltype_autodecltype(auto) semantics, as defined in
N3638.cxx_digit_separatorsDigit separators, as defined in
N3781.cxx_generic_lambdasGeneric lambdas, as defined in
N3649.cxx_lambda_init_capturesInitialized lambda captures, as defined in
N3648.cxx_relaxed_constexprRelaxed constexpr, as defined in
N3652.cxx_return_type_deductionReturn type deduction on normal functions, as defined in N3386.cxx_variable_templatesVariable templates, as defined in N3651.

## 调试属性方法

- set(CMAKE_DEBUG_TARGET_PROPERTIES　INCLUDE_DIRECTORIES)
- cmake_print_properties(TARGETS　xlog　PROPERTIES　INCLUDE_DIRECTORIES INTERFACE_INCLUDE_DIRECTORIESINTERFACE_SOURCESSOURCES)1
- set(CMAKE_VERBOSE_MAKEFILE ON)

- 看生成的 g++、cl 语句

file

- file(READ <filename> <out-var> [...])
- 安装那一章再讲
- file({WRITE | APPEND} <filename> <content>...)
- file({REMOVE | REMOVE_RECURSE } [<files>...])
- file(SIZE <filename> <out-var>)
- file(COPY_FILE <oldname> <newname> [...])
- file({COPY | INSTALL} <file>... DESTINATION <dir> [...])
- file(DOWNLOAD <url> [<file>] [...])
- file(UPLOAD <file> <url> [...])
- file(ARCHIVE_CREATE OUTPUT <archive> PATHS <paths>... [...])
- file(ARCHIVE_EXTRACT INPUT <archive> [...])

## add_library 详细配置

二进制对象库 OBJECT 的编译和依赖配置

- 分 obj 编译

  - obj 用于两个执行文件，用于测试
  - add_library(archive OBJECT archive.cpp zip.cpp lzma.cpp)
- -fPIC
  - set(CMAKE_POSITION_INDEPENDENT_CODE ON)
  - set_target_properties(lib1 PROPERTIES POSITION_INDEPENDENT_CODE ON)

### 带版本号的库符号链接

- NO_SONAME
  - ON 不产生动态库的符号链接
- VERSION
  - 1.0.1
- SOVERSION
  - 10
- set_target_properties(A PROPERTIES VERSION "1.0.1"SOVERSION "10")

## CMake Debug Release 配置

### Debug/Release Mode

- -O，-O1

    - 不影响编译速度的前提下，尽量采用一些优化算法降低代码大小和可执行代码的运行速度

- -O2

    - 牺牲部分编译速度，除了执行-O1 所执行的所有优化之外，还会采用几乎所有的目标配置支持的优化算法，用以提高目标代码的运行速度

- -O3

    - 执行-O2 所有的优化选项，采取很多向量化算法，提高代码的并行执行程度，利用现代 CPU 中的流水线，Cache

config 对应的优化

- Debug
    - -g
- Release
    - -O2
- RelWithDebInfo
    - -O2 -g
- MinSizeRel
    - -O3

Windows 配置

- CMAKE_CONFIGURATION_TYPES =　Debug;Release;MinSizeRel;RelWithDebInfo
- 生成时指定发布配置
  - cmake --build build --config Release

Linux 配置

- 配置时指定
  - cmake .. -D CMAKE_BUILD_TYPE=MinSizeRel
  - set(CMAKE_BUILD_TYPE Debug)

配置 Debug Release 不同输出路径

- 执行程序和 dll 输出
  - RUNTIME_OUTPUT_DIRECTORY_<CONFIG>
    - RUNTIME_OUTPUT_DIRECTORY_DEBUG
    - RUNTIME_OUTPUT_DIRECTORY_RELEASE
- lib 和.a 库输出
  - ARCHIVE_OUTPUT_DIRECTORY_<CONFIG>
- .so 动态库输出
  - LIBRARY_OUTPUT_DIRECTORY_<CONFIG>

- pdb 文件输出
    - PDB_OUTPUT_DIRECTORY_<CONFIG>

debug 库名加后缀

- set_target_properties(${name}　PROPERTIES　DEBUG_POSTFIX "d")

pdb 文件的配置

- set_target_properties(${name}　PROPERTIES　PDB_NAME "${name}" PDB_NAME_DEBUG "${name}${pdb_debug_postfix}"　COMPILE_PDB_NAME "${name}"　COMPILE_PDB_NAME_DEBUG "${name}${pdb_debug_postfix}")
- PDB_OUTPUT_DIRECTORY ${CMAKE_RUNTIME_OUTPUT_DIRECTORY}/pdbPDB_OUTPUT_DIRECTORY_DEBUG ${CMAKE_RUNTIME_OUTPUT_DIRECTORY}/pdb/debug

使用生成表达式设置 vs 调试 debug 和 release 的不同路径

- if(MSVC)set_target_properties(${PROJECT_NAME}　PROPERTIES #RUNTIME_OUTPUT_DIRECTORY_DEBUG ${CMAKE_RUNTIME_OUTPUT_DIRECTORY}/debugVS_DEBUGGER_WORKING_DIRECTORY　$<IF:$<CONFIG:Debug>,debug,release>)endif()

## VS 编译特性添加

target_compile_options 编译参数

- target_compile_options(myexe PRIVATE /bigobj)

    - 这些标志将在此源文件构建时添加

- 这些标志将在此源文件构建时添加

    - if (MSVC)　　# warning level 4 and all warnings as errors
      add_compile_options(/W4 /WX)else()　　# lots of warnings and all warnings as errors　　add_compile_options(-Wall -Wextra -pedantic -Werror)endif() 1

- COMPILE_OPTIONS

- INTERFACE_COMPILE_OPTIONS

调试、MD

- MSVC_RUNTIME_LIBRARY

    - set_property(TARGET ${PROJECT_NAME}_OBJ PROPERTY
      MSVC_RUNTIME_LIBRARY "MultiThreaded$<$<CONFIG:Debug>:Debug>")else()
      string(APPEND CMAKE_CXX_COMPILE_OBJECT " -fPIC")endif()

    - MultiThreaded

        - Compile with -MT or equivalent flag(s) to use a multi-threaded statically-linked runtime library.

    - MultiThreadedDLL

        - Compile with -MD or equivalent flag(s) to use a multi-threaded dynamically-linked runtime library.

    - MultiThreadedDebug

    - Compile with -MTd or equivalent flag(s) to use a multi-threaded statically-linked runtime library.

  - MultiThreadedDebugDLL

    - Compile with -MDd or equivalent flag(s) to use a multi-threaded dynamically-linked runtime library.

vs 分组

- source_group

  - source_group(<name> [FILES <src>...] [REGULAR_EXPRESSION <regex>])

  - source_group(TREE <root> [PREFIX <prefix>] [FILES <src>...])

    - root 后面的 src 路径会去掉 root 的内容，显示剩下的路径

  - code

    - file(WRITE a2.cpp [=[ #include <iostream>using namespace std;void A2(){ cout<<"Call A function!"<<endl;}]=])file(WRITE a3.cpp [=[ #include <iostream>using namespace std;void A3(){ cout<<"Call A function!"<<endl;}]=])file(WRITE a4.cpp [=[ #include <iostream>using namespace std;void A4(){ cout<<"Call A function!"<<endl;}]=])add_executable(${PROJECT_NAME} ${PROJECT_NAME}.cpp a1.cpp a2.cpp a3.cpp a4.cpp)

    - source_group(src1 test_lib.cpp)source_group(TREE . PREFIX src/inc FILES a1.cpp )source_group(TREE . PREFIX src2/inc FILES a2.cpp )source_group(TREE . PREFIX src2/inc2 FILES a3.cpp )

# 第六章 CMake install 部署项目

## 测试代码准备

### 源码

- include/slib.h
  - file(WRITE include/slib.h [=[void SLib();]=])
- include/slib_pri.h
  - file(WRITE include/slib_pri.h [=[#define PRI]=])
- src/slib.cpp
  - file(WRITE src/slib.cpp [=[#include <iostream>#include "slib.h"void SLib(){ std::cout<<"In Slib\n";}]=])
- src/dlib.cpp
  - file(WRITE src/dlib.cpp [=[#include <iostream>#ifdef _WIN32__declspec(dllexport) #endifvoid DLib(){ std::cout<<"In Dlib\n";}]=])
- main.cpp
  - file(WRITE main.cpp [=[#include <iostream>#include "slib.h"int main(){ void DLib(); DLib(); SLib(); std::cout<<"In main\n"; return 0;}]=])

### 静态库

- add_library(slib STATIC src/slib.cpp)

- set_target_properties(slib PROPERTIES PUBLIC_HEADER include/slib.h)set_target_properties(slib PROPERTIES PRIVATE_HEADER include/slib_pri.h)

动态库

- add_library(dlib SHARED src/dlib.cpp)

执行程序

- add_executable(${PROJECT_NAME} main.cpp)

## 安装命令

### 指定安装路径

- cmake build -D CMAKE_INSTALL_PREFIX=./

cmake --install build

## 安装目标

### 语法

- install(TARGETS targets... [EXPORT <export-name>] [RUNTIME_DEPENDENCIES args...|RUNTIME_DEPENDENCY_SET <set-name>] [[ARCHIVE|LIBRARY|RUNTIME|OBJECTS|FRAMEWORK|BUNDLE| PRIVATE_HEADER|PUBLIC_HEADER|RESOURCE|FILE_SET <set-name>] [DESTINATION <dir>] [PERMISSIONS permissions...]

[CONFIGURATIONS [Debug|Release|...]] [COMPONENT <component>] [NAMELINK_COMPONENT <component>] [OPTIONAL] [EXCLUDE_FROM_ALL] [NAMELINK_ONLY|NAMELINK_SKIP] ] [...] [INCLUDES DESTINATION [<dir> ...]] )

-

DESTINATION 安装路径

- 指定安装的目录,可以是相对路径或绝对路径
- 相对路径则这相对于 CMAKE_INSTALL_PREFIX

PERMISSIONS 权限

- 指 定 文 件 权 限 OWNER_READ, OWNER_WRITE, OWNER_EXECUTE, GROUP_READ, GROUP_WRITE, GROUP_EXECUTE, WORLD_READ, WORLD_WRITE, WORLD_EXECUTE, SETUID, 和 SETGID. 在某些平台上无意义的权限会被忽略。

CONFIGURATIONS （Debug Release）

- 指定安装规则适用的构建配置列表（Debug，Release）
- install(TARGETS target CONFIGURATIONS Debug RUNTIME DESTINATION Debug/bin)install(TARGETS target CONFIGURATIONS Release RUNTIME DESTINATION Release/bin)
- 需要设置在 RUNTIME DESTINATION 之前

OPTIONAL

- 可选的，如果目标不存在，不失败

目标分类

- RUNTIME
    - 执行程序
        - 由 add_executable 创建
    - windows 动态链接库 dll 文件
    - 设置 bin
- ARCHIVE
    - windows 动态库库导出符号
        - .lib on most Windows, .dll.a on Cygwin and MinGW
    - 静态库
        - add_library 添加 STATIC 参数
        - windows 是 .lib, Unix、Linux 和 MinGW 是.a
- LIBRARY
    - 动态库
        - add_library 使用 SHARED 参数

        - linux、unix
            - .so
        - mac
            - dylib

- PUBLIC_HEADER、PRIVATE_HEADER
    - set_target_properties(slib PROPERTIES PUBLIC_HEADER include/slib.h)set_target_properties(slib PROPERTIES PRIVATE_HEADER include/slib_pri.h)

- 代码演示
    - install(TARGETS mylib RUNTIME DESTINATION bin LIBRARY DESTINATION lib ARCHIVE DESTINATION lib/myproject)

## cmake install 安装文件

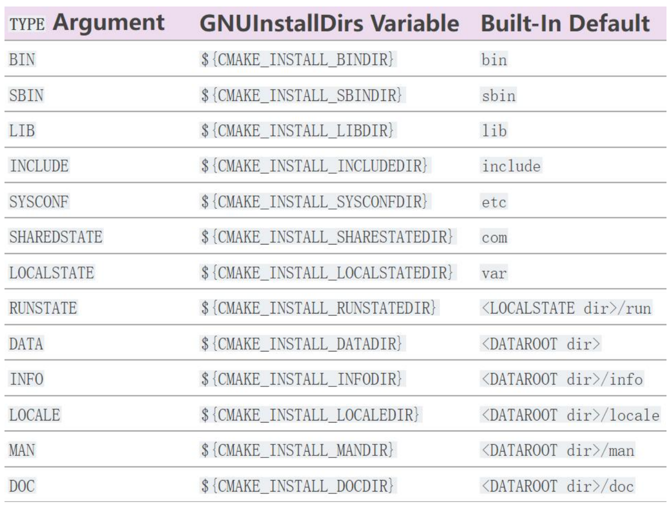
### 语法

- install(<FILES|PROGRAMS> files... TYPE <type> | DESTINATION <dir> [PERMISSIONS permissions...] [CONFIGURATIONS [Debug|Release|...]] [COMPONENT <component>] [RENAME <name>] [OPTIONAL] [EXCLUDE_FROM_ALL])

### 文件权限

- 安 装 的 文 件 默 认 权 限 OWNER_WRITE, OWNER_READ, GROUP_READ, WORLD_READ

TYPE

- 

| TYPE Argument | GNUInstallDirs Variable | Built-In Default |
|---|---|---|
| BIN | ${CMAKE_INSTALL_BINDIR} | bin |
| SBIN | ${CMAKE_INSTALL_SBINDIR} | sbin |
| LIB | ${CMAKE_INSTALL_LIBDIR} | lib |
| INCLUDE | ${CMAKE_INSTALL_INCLUDEDIR} | include |
| SYSCONF | ${CMAKE_INSTALL_SYSCONFDIR} | etc |
| SHAREDSTATE | ${CMAKE_INSTALL_SHARESTATEDIR} | com |
| LOCALSTATE | ${CMAKE_INSTALL_LOCALSTATEDIR} | var |
| RUNSTATE | ${CMAKE_INSTALL_RUNSTATEDIR} | <LOCALSTATE dir>/run |
| DATA | ${CMAKE_INSTALL_DATADIR} | <DATAROOT dir> |
| INFO | ${CMAKE_INSTALL_INFODIR} | <DATAROOT dir>/info |
| LOCALE | ${CMAKE_INSTALL_LOCALEDIR} | <DATAROOT dir>/locale |
| MAN | ${CMAKE_INSTALL_MANDIR} | <DATAROOT dir>/man |
| DOC | ${CMAKE_INSTALL_DOCDIR} | <DATAROOT dir>/doc |

- include(GNUInstallDirs)install(FILES t.h TYPE doc)
- DATAROOT
  - CMAKE_INSTALL_DATAROOTDIR： share

## cmake install 目录

### 语法

- install(DIRECTORY dirs... TYPE <type> | DESTINATION <dir> [FILE_PERMISSIONS permissions...] [DIRECTORY_PERMISSIONS permissions...] [USE_SOURCE_PERMISSIONS] [OPTIONAL] [MESSAGE_NEVER] [CONFIGURATIONS [Debug|Release|...]] [COMPONENT <component>] [EXCLUDE_FROM_ALL] [FILES_MATCHING] [[PATTERN <pattern> | REGEX <regex>] [EXCLUDE] [PERMISSIONS

permissions...]] [...])

测试内容准备

- file(WRITE doc/index.html " ")file(WRITE doc/index.cc " ")file(WRITE doc/index.c " ")file(WRITE doc/.svn/tmp.cc " ")file(WRITE doc/.svn/tmp.html " ")file(WRITE doc/.git/tmp.cc " ")file(WRITE doc/d1/tmp.cc " ")

只匹配指定类型文件，所有目录都复制

- install(DIRECTORY doc DESTINATION doc1FILES_MATCHINGPATTERN "*.html")

去除所有 EXCLUDE 指定的目录，并匹配指定条件的文件

- install(DIRECTORY doc DESTINATION doc2FILES_MATCHING PATTERN "*.cc"PATTERN ".git" EXCLUDE #PATTERN "d1" EXCLUDE )

仅排除指定目录 加上 FILES_MATCHING 如果没有指定匹配文件内容, 则不匹配任何文件

- install(DIRECTORY doc DESTINATION doc3PATTERN ".git" EXCLUDE PATTERN ".svn" EXCLUDE #PATTERN "d1" EXCLUDE )

## 安装时执行程序

# %Y-%m-%dT%H:%M:%S install(CODE "MESSAGE(\"Sample install message.\")") install(CODE [=[ string(TIMESTAMP now "%Y-%m-%d %H:%M:%S")message(${now})FILE(APPEND install_log.txt "${now}\n")]=])

## 安装指定的模块

```
cmake .. -DCMAKE_INSTALL_PREFIX=./
```

```
cmake -DCOMPONENT=Runtime -P cmake_install.cmake
```

```
install(TARGETS ${PROJECT_NAME} dlib slib                RUNTIME DESTINATION bin2
COMPONENT Runtime #test_install                          LIBRARY DESTINATION lib2
COMPONENT Runtime # libdlib.so                 ARCHIVE DESTINATION lib2/myproject
COMPONENT Development                    PUBLIC_HEADER DESTINATION pub_include
COMPONENT Development                              PRIVATE_HEADER DESTINATION
pri_include            ) #libslib.a
```

```
cmake -DBUILD_TYPE=Debug -P cmake_install.cmake
```

## 自定义 find_package 可导入库

find_package

- find_package(<PackageName> [version] [EXACT] [QUIET] [MODULE] [REQUIRED] [[COMPONENTS] [components...]] [OPTIONAL_COMPONENTS components...] [NO_POLICY_SCOPE])
- <PackageName>_FOUND
- Module mode
  - Find<PackageName>.cmake
  - CMAKE_MODULE_PATH
- Config mode
  - 查找路径

  - CMAKE_PREFIX_PATH

- 读取文件
  - config
    - <lowercasePackageName>-config.cmake
    - <PackageName>Config.cmake
  - version
    - <lowercasePackageName>-config-version.cmake
    - <PackageName>ConfigVersion.cmake
- 生成 Config mode 文件
  - config
    - install(TARGETS slib EXPORT slibRUNTIME DESTINATION binLIBRARY DESTINATION ${CMAKE_SOURCE_DIR} libPUBLIC_HEADER DESTINATION include)
    - install (EXPORT slib　NAMESPACE xcpp::　FILE slibConfig.cmake DESTINATION mod/slib/)
  - version
    - include(CMakePackageConfigHelpers)write_basic_package_version_file(${CMAKE_SOURCE_DIR}/out/mod/slib-${version}/slibConfigVersion.c

make VERSION ${version}
COMPATIBILITY SameMajorVersion)

- 使用示例

  - find_package(slib)add_executable(main main.cpp)target_link_libraries(main slib)

install export

- install(TARGETS slib EXPORT slibRUNTIME DESTINATION binLIBRARY DESTINATION libPUBLIC_HEADER DESTINATION include)
- install (EXPORT slib NAMESPACE xcpp:: FILE slibConfig.cmake DESTINATION slib)

code

- code1

  - cmake_minimum_required(VERSION 3.22)project(slib2)if(NOT version)set(version 1.1)endif()file(WRITE include/slib.h [=[void SLib();]=])file(WRITE include/slib_pri.h [=[void SLib2();]=])file(WRITE slib.cpp.in [=[#include <iostream>void SLib(){ std::cout<<"test slib ${version} \n";}]=])configure_file("slib.cpp.in" "${CMAKE_SOURCE_DIR}/slib.cpp" )file(WRITE slib2.cpp [=[#include <iostream>void SLib(){ std::cout<<"test slib 1.1 \n";}]=])add_library(slib SHARED slib.cpp)set_target_properties(slib PROPERTIES VERSION ${version})target_include_directories(slib PUBLIC /home/xcj/test_mode/out/include )set_target_properties(slib PROPERTIES

```
PUBLIC_HEADER include/slib.h)set_target_properties(slib PROPERTIES
PRIVATE_HEADER include/slib_pri.h)install(TARGETS slib EXPORT slibRUNTIME
DESTINATION binLIBRARY DESTINATION
${CMAKE_SOURCE_DIR}/out/mod/slib-${version}/libPUBLIC_HEADER
DESTINATION includePRIVATE_HEADER DESTINATION
include/in)include(CMakePackageConfigHelpers)write_basic_package_version_
file(${CMAKE_SOURCE_DIR}/out/mod/slib-${version}/slibConfigVersion.cmake
VERSION ${version}                                              COMPATIBILITY
SameMajorVersion)#write_basic_package_version_file(${CMAKE_SOURCE_DIR}
/out/mod1/slibConfigVersion.cmake#
VERSION 1.1#                                                    COMPATIBILITY
SameMajorVersion)      install (EXPORT slib     NAMESPACE xcpp::        FILE
slibConfig.cmake DESTINATION mod/slib-${version}/)
#install(EXPORT slib_mod NAMESPACE mp_#install(EXPORT slib_mod#
DESTINATION mod)#export(PACKAGE slib_mod)message("path is
${CMAKE_SOURCE_DIR}/out/mod/")#set(CMAKE_MODULE_PATH
"${CMAKE_SOURCE_DIR}/out/mod/")set(CMAKE_PREFIX_PATH
"${CMAKE_SOURCE_DIR}/out/mod/")#find_package(slib
${version})message("slib_DIR = ${slib_DIR}")message("slib_FOUND =
${slib_FOUND}")message("slib_INCLUDES =
${slib_INCLUDES}")message("slib_INCLUDE_DIR =
${slib_INCLUDE_DIR}")message("slib_LIBRARY =
${slib_LIBRARY}")message("slib_LIBRARIES =
${slib_LIBRARIES}")message("slib_CONSIDERED_CONFIGS =
${slib_CONSIDERED_CONFIGS}")message("slib_CONSIDERED_VERSIONS =
${slib_CONSIDERED_VERSIONS}")message("slib_CONFIG =
${slib_CONFIG}")#FIND_PACKAGE(curl)#message("CURL_DIR = ${curl_DIR}")
```

- code2

  - cmake_minimum_required(VERSION 3.22)project(findpkg)file(WRITE main.cpp [=[#include <iostream>int main(){　　std::cout<<"test main\n";　void SLib();　　SLib();　　return 0;}]=])set(CMAKE_PREFIX_PATH "/home/xcj/test_mode/out/mod/;/home/xcj/test_mode/out/mod1/")find_package(slib 1.2)add_executable(main main.cpp)target_link_libraries(main xcpp::slib)get_target_property(pa xcpp::slib INCLUDE_DIRECTORIES)include(CMakePrintHelpers)cmake_print_properties(TARGETS xcpp::slib PROPERTIESINCLUDE_DIRECTORIESINTERFACE_INCLUDE_DIRECTORIES)message("xcpp::slib INCLUDE_DIRECTORIES = ${pa}")message("slib_DIR = ${slib_DIR}")message("slib_FOUND = ${slib_FOUND}")message("slib_INCLUDES = ${slib_INCLUDES}")message("slib_INCLUDE_DIR = ${slib_INCLUDE_DIR}")message("slib_LIBRARY = ${slib_LIBRARY}")message("slib_LIBRARIES = ${slib_LIBRARIES}")message("slib_CONSIDERED_CONFIGS = ${slib_CONSIDERED_CONFIGS}")message("slib_CONSIDERED_VERSIONS = ${slib_CONSIDERED_VERSIONS}")message("slib_CONFIG = ${slib_CONFIG}")

# 第七章　编译安卓、嵌入式 Linux 和鸿蒙的程序-CMake 交叉编译

## toolchain

### CMAKE_SYSTEM_NAME

- (必填)系统名称

  - Linux

  - Windows
  - Generic
    - 嵌入式无系统

CMAKE_SYSTEM_PROCESSOR

- （可选）目标系统的处理器或硬件名称
  - 用于加载 ${CMAKE_SYSTEM_NAME}-COMPILER_ID-${CMAKE_SYSTEM_PROCESSOR}.cmake
  - 修改目标的编译器标志

CMAKE_C_COMPILER

- c 编译器全路径

CMAKE_CXX_COMPILER

- c++编译器全路径
  - GNU 工具链, 则只需设置 CMAKE_C_COMPILER; CMake 应该会自动找到相应的 C++ 编译器, 实测-D 才能自动找

CMAKE_SYSROOT

- （可选）

  - 系统库头文件的路径

查看程序架构

- file main

CMAKE_TOOLCHAIN_FILE

- 指定文件路径
- ohos.toolchain.cmake

## linux arm

GCC 编译器命名格式

- arch 目标芯片架构 os 操作系统 gnu C 标准库类型 eabi 应用二进制接口 hf 浮点模式
- aarch64-linux-gnu-g++

测试环境

- 编译使用的系统
  - ubuntu20.04 x64
- 目标系统
  - ubuntu arm 版本

- 编译工具
    - gcc-linaro-7.3.1-2018.05-x86_64_aarch64-linux-gnu
- 开发板
    - rockpi4
        - rk3399

准备工具

- tar -xvf gcc-linaro-7.3.1-2018.05-x86_64_aarch64-linux-gnu.tar.xz

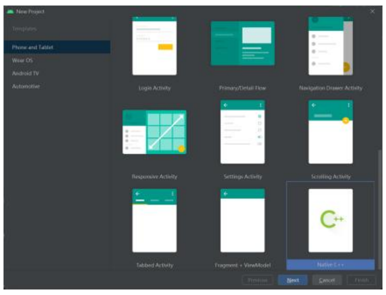
编译指令

- cmake -S . -B build -DCMAKE_TOOLCHAIN_FILE=linux_arm_toolchain.cmake

## cmake 交叉编译安卓 NDK 库

环境

- Android Studio Bumblebee
- 创建 native c++项目



编译配置

- ANDROID_ABI
  - x86
  - x86_64
  - armeabi-v7a
  - arm64-v8a
- CMAKE_TOOLCHAIN_FILE
  - C:/Users/xiaca/AppData/Local/Android/Sdk/ndk/21.4.7075529/build/cmake/android.toolchain.cmake
- ANDROID_NDK
  - C:/Users/xiaca/AppData/Local/Android/Sdk/ndk/21.4.7075529/
- ANDROID_PLATFORM
  - android-30

编译指令

- cmake -DANDROID_ABI=x86 -DCMAKE_TOOLCHAIN_FILE=C:/Users/xiaca/AppData/Local/Android/Sdk/ndk/21.4.7075529/build/cmake/android.toolchain.cmake -DANDROID_NDK=C:/Users/xiaca/AppData/Local/Android/Sdk/ndk/21.4.707552

9/ -DANDROID_PLATFORM=android-30 -S . -B build -G "NMake Makefiles"

- 测试虚拟机的 ANDROID_ABI 和 ANDROID_PLATFORM 要和编译环境一致

代码说明

- 编译静态库
  - CMakeLists.txt
    - cmake_minimum_required(VERSION 3.18)project(mylib)file(WRITE mylib.h [=[const char *MyLib();]=])file(WRITE mylib.cpp [=[#include "mylib.h"const char *MyLib(){ return "mylib return";};]=])# 给安卓使用的静态库 add_library(mylib STATIC mylib.cpp)target_compile_options(mylib PRIVATE -fPIC)
  - 编译四种不同的 ABI
    - cmake -S . -B build -G "NMake Makefiles" -DANDROID_ABI=x86 -DANDROID_PLATFORM=android-30 -DCMAKE_TOOLCHAIN_FILE=C:/Users/xiaca/AppData/Local/Android/Sdk/ndk/21.4.7075529/build/cmake/android.toolchain.cmake -DANDROID_NDK=C:/Users/xiaca/AppData/Local/Android/Sdk/ndk/21.4.7075529
    - cmake -S . -B build -G "NMake Makefiles" -DANDROID_ABI=x86_64 -DANDROID_PLATFORM=android-30 -DCMAKE_TOOLCHAIN_FILE=C:/Users/xiaca/AppData/Local/Android/Sdk/ndk/21.4.7075529/build/cmake/android.toolchain.cmake

-DANDROID_NDK=C:/Users/xiaca/AppData/Local/Android/Sdk/ndk/21.4.7075529

    - cmake -S . -B build -G "NMake Makefiles" -DANDROID_ABI=armeabi-v7a -DANDROID_PLATFORM=android-30 -DCMAKE_TOOLCHAIN_FILE=C:/Users/xiaca/AppData/Local/Android/Sdk/ndk/21.4.7075529/build/cmake/android.toolchain.cmake -DANDROID_NDK=C:/Users/xiaca/AppData/Local/Android/Sdk/ndk/21.4.7075529

    - cmake -S . -B build -G "NMake Makefiles" -DANDROID_ABI=arm64-v8a -DANDROID_PLATFORM=android-30 -DCMAKE_TOOLCHAIN_FILE=C:/Users/xiaca/AppData/Local/Android/Sdk/ndk/21.4.7075529/build/cmake/android.toolchain.cmake -DANDROID_NDK=C:/Users/xiaca/AppData/Local/Android/Sdk/ndk/21.4.7075529

- 导入静态库

    - target_link_directories(a602cmake_android_ndk PUBLIC ${CMAKE_SOURCE_DIR}/mylib/build/)target_link_libraries( # Specifies the target library.　　a602cmake_android_ndk　　# Links the target library to the log library　　# included in the NDK.　　${log-lib} mylib　　)

code

- $　cmake　../src　\　-DCMAKE_SYSTEM_NAME=Android　\ -DCMAKE_SYSTEM_VERSION=21 \　-DCMAKE_ANDROID_ARCH_ABI=arm64-v8a \　-DCMAKE_ANDROID_NDK=/path/to/android-ndk　\

-DCMAKE_ANDROID_STL_TYPE=gnustl_static

- target_link_directories(myapplication PUBLIC ${CMAKE_SOURCE_DIR}/mylib/${ANDROID_ABI}/)target_link_libraries(myapplication libmylib.a)

- set(CMAKE_SYSTEM_NAME Android)set(CMAKE_SYSTEM_VERSION 21) # API levelset(CMAKE_ANDROID_ARCH_ABI arm64-v8a)set(CMAKE_ANDROID_NDK /path/to/android-ndk)set(CMAKE_ANDROID_STL_TYPE gnustl_static)

- cmake -DCMAKE_TOOLCHAIN_FILE=C:\Users\xiaca\AppData\Local\Android\Sdk\ndk\24.0.8215888\build\cmake\android.toolchain.cmake -S . -B b7 -G "NMake Makefiles"

- cmake -DANDROID_ABI=x86_64 -DCMAKE_TOOLCHAIN_FILE=C:/Users/xiaca/AppData/Local/Android/Sdk/ndk/21.4.7075529/build/cmake/android.toolchain.cmake -DANDROID_NDK=C:/Users/xiaca/AppData/Local/Android/Sdk/ndk/21.4.7075529/ -DANDROID_PLATFORM=android-30 -S . -B build -G "NMake Makefiles" cmake -DANDROID_ABI=x86 -DCMAKE_TOOLCHAIN_FILE=C:/Users/xiaca/AppData/Local/Android/Sdk/ndk/21.4.7075529/build/cmake/android.toolchain.cmake -DANDROID_NDK=C:/Users/xiaca/AppData/Local/Android/Sdk/ndk/21.4.7075529/ -DANDROID_PLATFORM=android-30 -S . -B build -G "NMake Makefiles" cmake -DANDROID_ABI=armeabi-v7a -DCMAKE_TOOLCHAIN_FILE=C:/Users/xiaca/AppData/Local/Android/Sdk/ndk/21.4.7075529/build/cmake/android.toolchain.cmake -DANDROID_NDK=C:/Users/xiaca/AppData/Local/Android/Sdk/ndk/21.4.707552

9/ -DANDROID_PLATFORM=android-30 -S . -B build -G "NMake Makefiles" cmake -DANDROID_ABI=arm64-v8a -DCMAKE_TOOLCHAIN_FILE=C:/Users/xiaca/AppData/Local/Android/Sdk/ndk/21.4.7075529/build/cmake/android.toolchain.cmake -DANDROID_NDK=C:/Users/xiaca/AppData/Local/Android/Sdk/ndk/21.4.7075529/ -DANDROID_PLATFORM=android-30 -S . -B build -G "NMake Makefiles"

- cmake_minimum_required (VERSION 3.10)project (mylib)file(WRITE mylib.h [=[const char * Mylib();]=])file(WRITE mylib.cpp [=[#include <iostream>using namespace std;const char * Mylib(){ cout<<"call Mylib"<<endl; return "mylib";}]=])#[[cmake -DCMAKE_TOOLCHAIN_FILE=D:/harmony_sdk/native/2.2.0.3/build/cmake/ohos.toolchain.cmake -S . -B b2 -G Ninjacmake -DCMAKE_TOOLCHAIN_FILE=C:\Users\xiaca\AppData\Local\Android\Sdk\ndk\24.0.8215888\build\cmake\android.toolchain.cmake -S . -B build -G "NMake Makefiles"cmake -DCMAKE_TOOLCHAIN_FILE=C:\Users\xiaca\AppData\Local\Android\Sdk\ndk\21.4.7075529\build\cmake\android.toolchain.cmake -S . -B build -G "NMake Makefiles"cmake -DCMAKE_TOOLCHAIN_FILE=C:/Users/xiaca/AppData/Local/Android/Sdk/ndk/21.4.7075529/build/cmake/android.toolchain.cmake -DANDROID_ABI=armeabi-v7a -DANDROID_NDK=C:/Users/xiaca/AppData/Local/Android/Sdk/ndk/21.4.7075529/ -DANDROID_PLATFORM=android-30 -S . -B build -G "NMake Makefiles" cmake -DCMAKE_TOOLCHAIN_FILE=C:/Users/xiaca/AppData/Local/Android/Sdk/ndk/21.4.7075529/build/cmake/android.toolchain.cmake -DANDROID_ABI=x86_64 -DANDROID_NDK=C:/Users/xiaca/AppData/Local/Android/Sdk/ndk/21.4.707552

9/ -DANDROID_PLATFORM=android-30 -S . -B build -G "NMake Makefiles" cmake -DANDROID_ABI=x86_64 -DCMAKE_TOOLCHAIN_FILE=C:/Users/xiaca/AppData/Local/Android/Sdk/ndk/21.4.7075529/build/cmake/android.toolchain.cmake -DANDROID_NDK=C:/Users/xiaca/AppData/Local/Android/Sdk/ndk/21.4.7075529/ -DANDROID_PLATFORM=android-30 -S . -B build -G "NMake Makefiles" cmake -DANDROID_ABI=x86 -DCMAKE_TOOLCHAIN_FILE=C:/Users/xiaca/AppData/Local/Android/Sdk/ndk/21.4.7075529/build/cmake/android.toolchain.cmake -DANDROID_NDK=C:/Users/xiaca/AppData/Local/Android/Sdk/ndk/21.4.7075529/ -DANDROID_PLATFORM=android-30 -S . -B build -G "NMake Makefiles" cmake -DANDROID_ABI=armeabi-v7a -DCMAKE_TOOLCHAIN_FILE=C:/Users/xiaca/AppData/Local/Android/Sdk/ndk/21.4.7075529/build/cmake/android.toolchain.cmake -DANDROID_NDK=C:/Users/xiaca/AppData/Local/Android/Sdk/ndk/21.4.7075529/ -DANDROID_PLATFORM=android-30 -S . -B build -G "NMake Makefiles" cmake -DANDROID_ABI=arm64-v8a -DCMAKE_TOOLCHAIN_FILE=C:/Users/xiaca/AppData/Local/Android/Sdk/ndk/21.4.7075529/build/cmake/android.toolchain.cmake -DANDROID_NDK=C:/Users/xiaca/AppData/Local/Android/Sdk/ndk/21.4.7075529/ -DANDROID_PLATFORM=android-30 -S . -B build -G "NMake Makefiles" cmake -DANDROID_ABI=arm64-v8a -DCMAKE_TOOLCHAIN_FILE=C:/Users/xiaca/AppData/Local/Android/Sdk/ndk/21.4.7075529/build/cmake/android.toolchain.cmake -DANDROID_PLATFORM=android-30 -S . -B build -G "NMake Makefiles" ]]message("ANDROID_ABI = ${ANDROID_ABI}")#if(ANDROID_ABI equal "x86_64")#set(ARCHIVE_OUTPUT_DIRECTORY ${CMAKE_SOURCE_DIR}/${ANDROID_ABI})#endif()add_library(mylib STATIC mylib.cpp)set_target_properties(mylib PROPERTIES ARCHIVE_OUTPUT_DIRECTORY ${CMAKE_SOURCE_DIR}/${ANDROID_ABI} )#add_library(mylib SHARED mylib.cpp)target_compile_options(mylib PRIVATE -fPIC)

## 鸿蒙 HarmonyOS

### 测试环境

- 编译使用的系统
  - windows11
- 目标系统
  - HarmonyOS 2.0
- 编译工具
  - llvm
    - clang
  - Ninja
    - D:\harmony_sdk\native\2.2.0.3\build-tools\cmake\bin\ninja.exe
- 手机
  - 华为 P40

    - arm64-v8a

- 开发工具

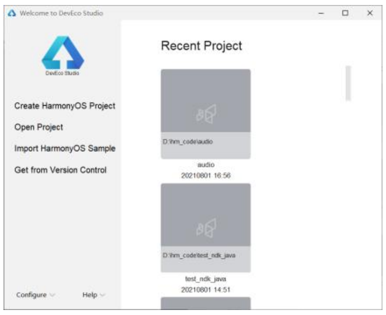
    - DevEco Studio 3.0.0.600

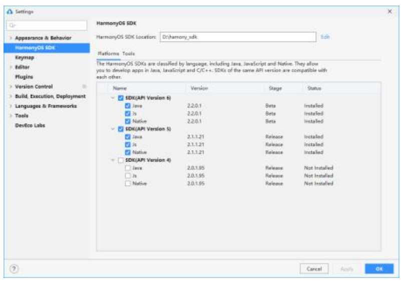
hap 应用开发测试

- 确认安装好开发工具 DevEco studio，版本需要支持 Native SDK 的

    - DevEco Studio 3.0.0.600

- 设置安装 Native SDK（NDK）

    - 

    - 

Settings

Appearance & Behavior
HarmonyOS SDK
Keymap
Editor
Plugins
Version Control
Build, Execution, Deployment
Languages & Frameworks
Tools
DevEco Labs

HarmonyOS SDK

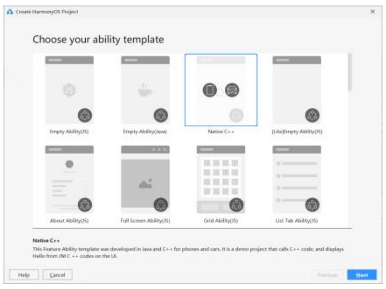
HarmonyOS SDK Location: D:\harmony_sdk　Edit

Platforms　Tools

The HarmonyOS SDKs are classified by language, including Java, JavaScript and Native. They allow you to develop apps in Java, JavaScript and C/C++. SDKs of the same API version are compatible with each other.

| Name | Version | Stage | Status |
|---|---|---|---|
| ✓ SDK(API Version 6) | | | |
| ✓ Java | 2.2.0.1 | Beta | Installed |
| ✓ Js | 2.2.0.1 | Beta | Installed |
| ✓ Native | 2.2.0.1 | Beta | Installed |
| ✓ SDK(API Version 5) | | | |
| ✓ Java | 2.1.1.21 | Release | Installed |
| ✓ Js | 2.1.1.21 | Release | Installed |
| ✓ Native | 2.1.1.21 | Release | Installed |
| ☐ SDK(API Version 4) | | | |
| ☐ Java | 2.0.1.95 | Release | Not Installed |
| ☐ Js | 2.0.1.95 | Release | Not Installed |
| ☐ Native | 2.0.1.95 | Release | Not Installed |

- 创建 NDK 项目

- 

  **Native C++**
  This Feature Ability template was developed in Java and C++ for phones and cars. It is a demo project that calls C++ code, and displays Hello from JNI C ++ codes on the UI.

- #导入 mylib 静态库 add_library(mylib STATIC IMPORTED)#指定导入库的路径 set_target_properties(mylib PROPERTIES IMPORTED_LOCATION ${CMAKE_CURRENT_SOURCE_DIR}/mylib /liblua.a)add_library(test_ndk SHARED test_ndk cpp)target_link_libraries(test_ndk libhilog_ndk.z.so mylib)

cmake -DCMAKE_TOOLCHAIN_FILE=D:/harmony_sdk/native/2.2.0.3/build/cmake/ohos.toolchain.cmake -S . -B build -G Ninja

code

if(CMAKE_SYSTEM MATCHES Windows) message(STATUS "Target system is Windows")endif()if(CMAKE_HOST_SYSTEM MATCHES Linux) message(STATUS "Build host runs Linux")endif()

cmake -DCMAKE_TOOLCHAIN_FILE=~/Toolchains/Toolchain-eldk-mips4K.cmake \ -DCMAKE_INSTALL_PREFIX=~/eldk-mips-extra-install ..

--toolchain path/to/file or -DCMAKE_TOOLCHAIN_FILE=path/to/file

# 第八章 测试驱动开发-cmake 自动单元测试

ctest

add_test

- add_test(NAME <name> COMMAND <command> [<arg>...] [CONFIGURATIONS <config>...] [WORKING_DIRECTORY <dir>] [COMMAND_EXPAND_LISTS])

- add_test(NAME test_uni COMMAND $<TARGET_FILE:${PROJECT_NAME}> 1)

enable_testing()

成功失败判断方法

- main 函数返回值
  - 0 成功
- PASS_REGULAR_EXPRESSION
  - 匹配成功的控制台输出
    - 支持正则
  - set_tests_properties(test PROPERTIES PASS_REGULAR_EXPRESSION "99" )
- FAIL_REGULAR_EXPRESSION
  - 匹配失败的控制台输出
    - 支持正则

- set_tests_properties(test PROPERTIES FAIL_REGULAR_EXPRESSION "fail")

编译步骤

- 1 编写 CMakeLists.txt

  - ## test_ctest/CMakeLists.txtcmake_minimum_required(VERSION 3.22)project(test_ctest)file(WRITE ${PROJECT_NAME}.cpp [=[#include <iostream>using namespace std;int main(int argc,char *argv[]){ cout<<"test_ctest"<<endl; if(argc>1) cout<<argv[1]<<endl; return 0; }]=])add_executable(${PROJECT_NAME} ${PROJECT_NAME}.cpp)enable_testing()#[[ctest --build-and-test . b --build-generator "Visual Studio 17 2022" --build-options Debugadd_test(NAME <name> COMMAND <command> [<arg>...] [CONFIGURATIONS <config>...] [WORKING_DIRECTORY <dir>] [COMMAND_EXPAND_LISTS])]]add_test(NAME test_success COMMAND ${PROJECT_NAME} success #CONFIGURATIONS Debug Release #-C <cfg>, --build-config <cfg> WORKING_DIRECTORY ${CMAKE_SOURCE_DIR} )set_tests_properties(test_success PROPERTIES PASS_REGULAR_EXPRESSION success ) add_test(NAME test_failed COMMAND ${PROJECT_NAME} failed #CONFIGURATIONS Debug Release #-C <cfg>, --build-config <cfg> )set_tests_properties(test_failed PROPERTIES FAIL_REGULAR_EXPRESSION failed ) add_test(NAME test3 COMMAND ${PROJECT_NAME} test3 )set_tests_properties(test3 PROPERTIES PASS_REGULAR_EXPRESSION success )

- 2 生成+编译

  - ctest --build-and-test . build --build-generator "Visual Studio 17 2022" --build-config Debug
  - ctest --build-and-test . build --build-generator "Unix Makefiles" --build-config Debug
- 2 生成
  - cmake -S . -B build
- 3 编译
  - cmake --build build
- 4 运行测试
  - cd build
  - ctest -C Debug

## gtest

### 安装方法

- git 源码下载编译（网络状况不确定）
  - DownloadProject
  - Fetch (CMake 3.11)

  - cmake_minimum_required(VERSION 3.14)project(my_project)# GoogleTest requires at least C++14set(CMAKE_CXX_STANDARD 14)include(FetchContent)FetchContent_Declare(    googletest    URL https://github.com/google/googletest/archive/609281088cfefc76f9d0ce82e1ff6c30cc3591e5.zip)# For Windows: Prevent overriding the parent project's compiler/linker settingsset(gtest_force_shared_crt ON CACHE BOOL "" FORCE)FetchContent_MakeAvailable(googletest)

  - #include <gtest/gtest.h>// Demonstrate some basic assertions.TEST(HelloTest, BasicAssertions) {    // Expect two strings not to be equal.    EXPECT_STRNE("hello", "world");    // Expect equality.    EXPECT_EQ(7 * 6, 42);}

  - enable_testing()add_executable(    hello_test    hello_test.cc)target_link_libraries(    hello_test    gtest_main)include(GoogleTest)gtest_discover_tests(hello_test)

- 直接下载发布库和头文件

- 手动下载源码编译安装

execute_process

- execute_process(COMMAND <cmd1> [<arguments>] [COMMAND <cmd2> [<arguments>]]... [WORKING_DIRECTORY <directory>]

- cmake 解压

  - tar

- gtest-1.11.0.tar.gz

- cmake 配置

- cmake 编译

- cmake 安装

FetchContent_Declare

- FetchContent_Declare( googletest GIT_REPOSITORY https://github.com/google/googletest.git GIT_TAG 703bd9caab50b139428cea1aaff9974ebee5742e # release-1.10.0)FetchContent_Declare( myCompanyIcons URL https://intranet.mycompany.com/assets/iconset_1.12.tar.gz URL_HASH MD5=5588a7b18261c20068beabfb4f530b87)FetchContent_Declare( myCompanyCertificates SVN_REPOSITORY svn+ssh://svn.mycompany.com/srv/svn/trunk/certs SVN_REVISION -r12345)

简单测试

- TEST(TestSuiteName, TestName) {　... test body ...}

- // Tests factorial of 0.TEST(FactorialTest, HandlesZeroInput) { EXPECT_EQ(Factorial(0), 1);}// Tests factorial of positive numbers.TEST(FactorialTest, HandlesPositiveInput) {　EXPECT_EQ(Factorial(1), 1); EXPECT_EQ(Factorial(2), 2); EXPECT_EQ(Factorial(3), 6); EXPECT_EQ(Factorial(8), 40320);}

运行测试

- #include "gtest/gtest.h"int main(int argc, char **argv) {  ::testing::InitGoogleTest(&argc, argv);  return RUN_ALL_TESTS();}

# 第九章 实战综合项目-CMake 开源项目 xcpp

## 项目配置需求

### 输出路径配置和编译器

- 静态库
- 动态库
- 执行程序
- windows
  - pdb 调试输出路径 pdb
  - 库导入 lib

### 每个项目可以独立编译

### Debug Release 配置

- 输出到同一个路径
- windows Debug 加后缀 d

源码需求

- 由 cmake 统一配置命名空间
- 支持 windows 和 linux 上的动态库
  - windows 需要 export import

用户配置

- xlog xthread_pool 设置为静态库或者动态库
- 是否编译 xlog xthread_pool
- 是否编译示例
- 是否编译单元测试

集成测试 samples 需求

- 添加示例只增加文件，不修改已有 cmake 文件

单元测试需求

- 第一次访问自动解压编译 gtest

install

- 安装库文件、头文件、执行文件

- 安装 cmake Package
    - 生成配置文件
        - XLogConfig.cmake
    - 生成版本文件

## 项目代码说明

### 开源库程序

- xlog
- xthread_pool

### 集成测试-示例程序

- test_xlog 集成测试程序
- test_xthread_pool 集成测试程序

### 单元测试程序

- unit_xlog
- unit_xthread_pool

## 目录结构

├── bin ├── cmake│ ├── common.cmake│ └── gtest.cmake ├──

CMakeLists.txt ├── lib ├── src | ├── samples | | ├── CMakeLists.txt | | ├── test_xlog | | | ├── CMakeLists.txt | | | └── test_xlog.cpp | | └── test_xthread_pool | | ├── CMakeLists.txt | | └── test_thread_pool.cpp | ├── xlog | | ├── CMakeLists.txt | | ├── include | | | ├── xconfig.h | | | └── xlog.h | | ├── unit_test | | | ├── CMakeLists.txt | | | └── testmain.cpp | | ├── xconfig.h.in | | ├── xlog.cpp | | ├── xlog_thread.cpp | | └── xlog_thread.h | └── xthread_pool | ├── CMakeLists.txt | ├── include | | ├── xlib.h | | ├── xthread.h | | └── xthread_pool.h | ├── unit_test | | ├── CMakeLists.txt | | └── testmain.cpp | ├── xthread.cpp | └── xthread_pool.cpp └── tools └── gtest-1.11.0.tar.gz

目录层次

- bin
    - 执行程序、pdb、dll 文件输出
- lib
    - 动态库、静态库、lib、so、dylib 文件
- cmake
    - 公共 cmake 库文件
- src
    - samples

    - 示例程序
    - test_xlog
        - CMakeLists.txt
            - 示例程序项目
    - test_xthread_pool
        - CMakeLists.txt
            - 示例程序项目
    - CMakeLists.txt
        - 所有示例程序项目
- xlog
    - unit_test
        - 单元测试源码
        - CMakeLists.txt
            - 单元测试项目
    - 库源码和内部头文件
    - include
        - 对外接口头文件

    - CMakeLists.txt
      - 库项目
  - xthread_pool
    - unit_test
      - 单元测试源码
      - CMakeLists.txt
        - 单元测试项目
    - 库源码和内部头文件
    - include
      - 对外接口头文件
    - CMakeLists.txt
      - 库项目
- tools
  - 三方依赖库
- CMakeLists.txt
  - 完整项目

## 开发步骤

准备目录结构

- src
    - xlog
        - unit_test
    - samples
        - test_xlog
- bin
- lib
- cmake

1 xlog 库配置

2 xlog 配置重构

- cpp_library
    - function(cpp_library name shared)

3 test_xlog 配置

- cpp_execute

4 test_xlog 重构

- cpp_executable

- 重构 cpp_library

5 自动配置所有 samples

6 统一的 CMakeLists.txt 编写

- 同时编译 xlog 和所有 samples

7 xlog 单元测试配置

8 单元测试配置重构

9 添加库 xthread_pool 和它的示例和单元测试