# Cmake入门

## 1.1第一个Cmake

1. //first\_cmake.cpp
2. #include <iostream>
3. **using** **namespace** std;
4. **int** main(**int** argc,**char** \*argv[])
5. {
6. cout<<"first CMake TEST"<<endl;
7. **return** 0;
8. }
9. # CMakeLists.txt cmake文件名大小写不敏感
11. #指定cmake最低版本
12. cmake\_minimum\_required(VERSION 3.20)
14. #构建项目的名称
15. project(first\_cmake)
17. #构建执行程序
18. add\_executable(first\_cmake first\_cmake.cpp)

cmake -S . -B build 当前路径

Cmaek --build build

## 1.2 Cmake基本概念

Cmake用于构建、测试和软件打包的开源跨平台工具

1.为什么需要构建工具？

* 避免硬编码路径
* 在多台计算机上构建一个包
* 持续集成
* 支持不同操作系统
* 使用IDE，但不是所有情况
* 想描述程序逻辑结构，而不是标志和命令
* 使用库
* 使用其他工具帮助编写代码
* 想使用单元测试

注：

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，没完成一点更新，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，会导致集成成本变大

让产品可以快速的通过，同时保持关键测试合格

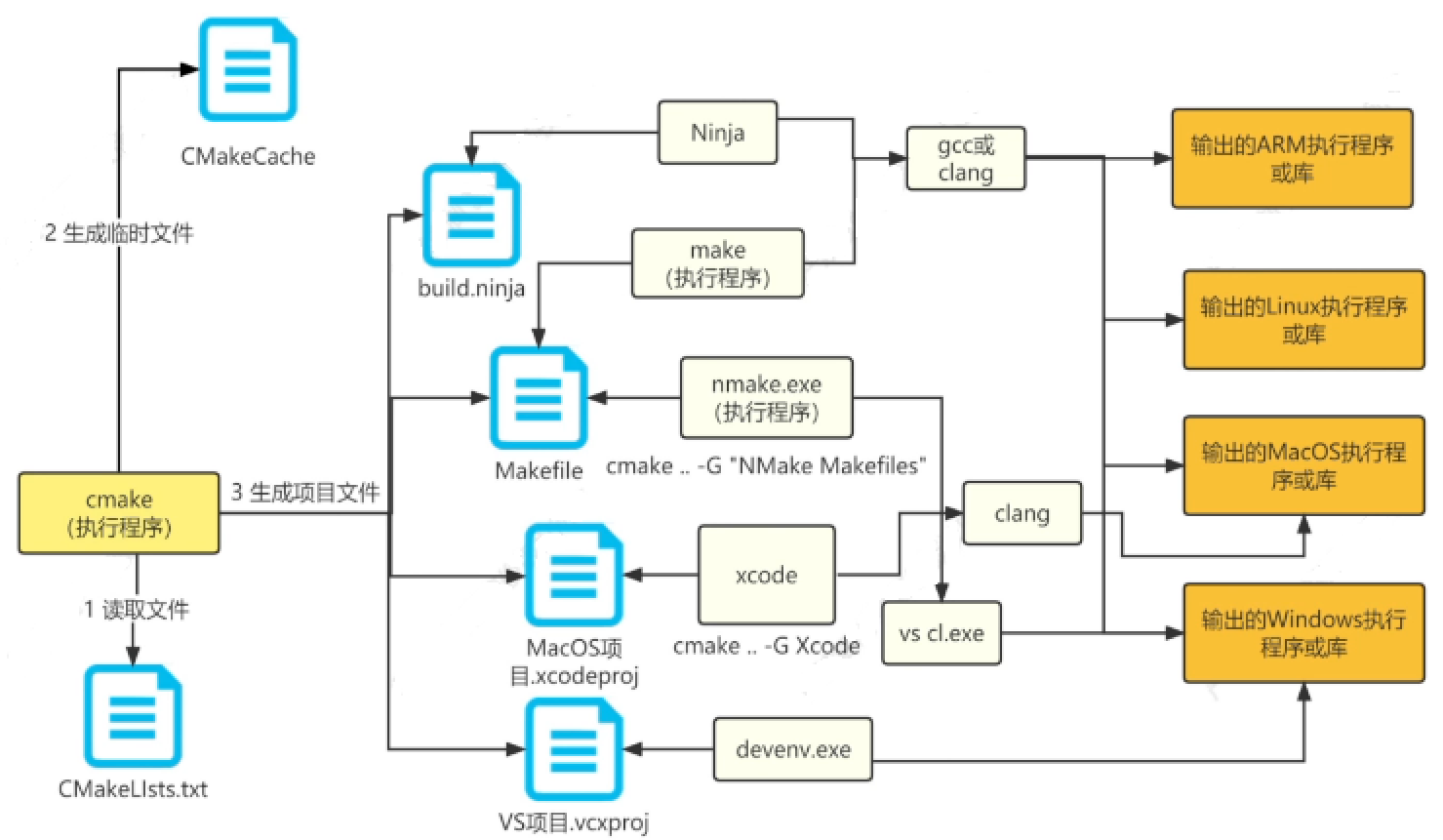
自动化测试，只要有一个测试用例不通过就不集成

集成并不能删除发现的错误，而是让他们很容易改正

2.为什么用Cmake?

* 自动搜素可能需要的程序、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
* 配置时选择可选组件能力
* 从简单的文本文件CMakeLists.txt自动生成工作区和项目能力
* 在静态和共享构建之间轻松切换能力

## 1.3 Cmake工作原理



## 1.4 动态库和静态库

静态库：相当于代码复制进来，运行时不需要加载，链接时间长

Windows xlog.lib xlog\_d.lib

Linux libxlog.a

类似编译后二进制代码 .o文件

开源库 静态链接（商业授权）

动态库：运行时需要加载，链接时间短

Windows xlog.dll xlog\_d.dll

Linux libxlog.so

macOS libxlog.dylib

## 1.5 头文件引用

* 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

## 1.6 静态库编译

1. //xlog.h
2. #ifndef XLOG\_H
3. #define XLOG\_H
5. **class** XLog
6. {
7. **public**:
8. XLog();
9. };
11. #endif
12. ----------------------------------------------------
13. //xlog.cpp
14. #include "xlog.h"
15. #include <iostream>
16. **using** **namespace** std;
17. XLog::XLog()
18. {
19. cout<<"Create XLog"<<endl;
20. }
21. ----------------------------------------------------
22. # CMakeLists.txt cmake文件名大小写不敏感
24. #指定cmake最低版本
25. cmake\_minimum\_required(VERSION 3.2)
27. #构建项目的名称
28. project(xlog)
30. #添加库
31. add\_library(xlog STATIC xlog.cpp xlog.h)



## 1.7编译静态库

1. # CMakeLists.txt cmake文件名大小写不敏感
3. #指定cmake最低版本
4. cmake\_minimum\_required(VERSION 3.2)
6. #构建项目的名称
7. project(first\_cmake)
9. #指定头文件查找路径
10. include\_directories("../XLog")
12. #指定查找路径
13. link\_directories("../XLog/build/Debug")
15. #构建执行程序
16. add\_executable(first\_cmake first\_cmake.cpp)
18. #指定加载库
19. target\_link\_libraries(first\_cmake xlog)

## 1.8动态库编译

add\_library(xlog SHARED xlog.cpp xlog.h)

windows下 class \_\_declspec(dllexport)XLog

Xlog库文件调用\_\_declspec(dllexport)

first\_cmake调用\_\_declspec(dllimport)

windows下

1. #if defined(WIN32) && !defined(CPP\_API\_STATIC)
2. #if defined(CPP\_API\_EXPORTS)
3. #define CPP\_API\_EXPORT \_\_declspec( dllexport )
4. #else
5. #define CPP\_API\_EXPORT \_\_declspec( dllimport )
6. #endif
7. #else
8. #define CPP\_API\_EXPORT
9. #endif

# CMake常用功能

## 2.1CMake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之前版本cmake不支持括号注释

* 行注释

# 一直运行到行尾

## 2.2 message

1)基础使用

message (arg1 arg2 arg3)

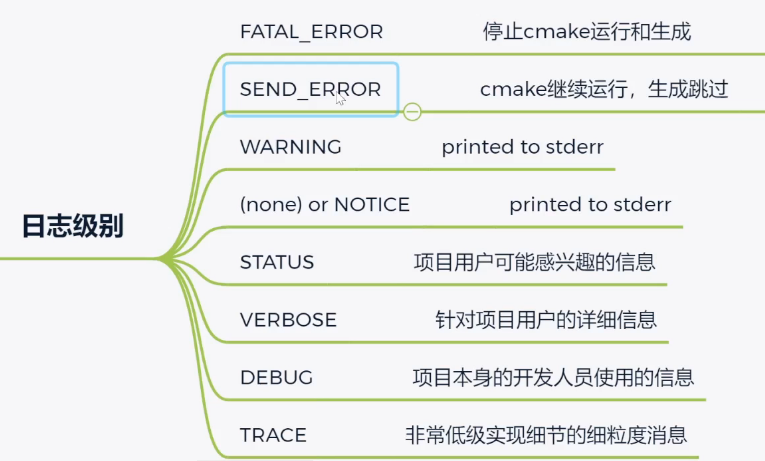
1. 高级使用——置顶日志级别

message([<model>] “message text”)

--log-level=<ERROR|WARNING|NOTICE|STATUS|VERBOSE|DEBUG|RRACE>

标准输出stdout

错误输出 stderr



# FATAL\_ERROR 进程退出，生成退出 打印代码路径和行号 stderr

#message(FATAL\_ERROR "TEST FATAL\_ERROR")

# SEND\_ERROR 进程继续，生成退出 不会生成 add\_executable add\_library stderr

# 打印代码路径和行号

message(SEND\_ERROR "TEST SEND\_ERROR")

add\_executable(test\_message test\_message.cpp)

message("after ERROR")

# WARNING 打印代码路径和行号 stderr

message(WARNING "TEST WARNING")

#NOTICE等同于 none也就是不加 message("TEST NOTICE") stderr

message("TEST none")

message(NOTICE "TEST NOTICE")

#STATUS 加前缀 -- 用户可能感兴趣 stdout

message(STATUS "TEST STATUS")

#VERBOSE 加前缀 -- 默认不显示 用户需要的详细信息 stdout

message(VERBOSE "TEST VERBOSE")

# 设置日志显示级别

# cmake -S . -B build --log-level=VERBOSE

# 标准输出重定向到文件log.txt

# cmake -S . -B build --log-level=VERBOSE > log.txt

# 标准错误输出重定向到标准输出

# cmake -S . -B build --log-level=VERBOSE > log.txt 2>&1

#DEBUG 加前缀 --

message(DEBUG "test DEBUG")

#TRACE 加前缀 --

# cmake -S . -B build --log-level=TRACE

message(TRACE "test TRACE")

1. message Reporting checks查找库日志

#[[

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

]]

message("=======================2.4=============================")

#开始查找

message(CHECK\_START "查找xcpp")

#查找库xcpp的代码

# message消息缩进

set(CMAKE\_MESSAGE\_INDENT "--")

#嵌套查找

message(CHECK\_START "查找xlog")

#查找xlog代码

message(CHECK\_PASS "成功")

message(CHECK\_START "查找xthreadpool")

message(CHECK\_FAIL "失败")

#取消缩进

set(CMAKE\_MESSAGE\_INDENT "")

#结束查找 查找失败

message(CHECK\_FAIL "失败")

1. 关键词 cmake message