## 一、CMakeLists

### 1. CMAKE\_<LANG>\_FLAGS

Flags for all build types.

CMAKE\_C\_FLAGS: Initialized by the CFLAGS environment variable.

CMAKE\_CXX\_FLAGS: Initialized by the CXXFLAGS environment variable.

add\_compile\_options 命令添加的编译选项是针对所有的编译器的，包括C和C++编译器。CMAKE\_C\_FLAGS和CMAKE\_CXX\_FLAGS可以分别只针对C和C++编译器。

### 2. CheckCXXSourceCompiles

Check if given C++ source compiles and links into an executable.

check\_cxx\_source\_compiles(<code> <resultVar>

[FAIL\_REGEX <regex1> [<regex2>...]])

Check that the source supplied in <code> can be compiled as a C++ source file and linked as an executable (so it must contain at least a main() function). The result will be stored in the internal cache variable specified by <resultVar>, with a boolean true value for success and boolean false for failure. If FAIL\_REGEX is provided, then failure is determined by checking if anything in the output matches any of the specified regular expressions.

### 3. configure\_file

Copy a file to another location and modify its contents.

### 4. add\_compile\_options

Add options to the compilation of source files.

Adds options to the COMPILE\_OPTIONS directory property. These options are used when compiling targets from the current directory and below.

### 5. GNUInstallDirs

Define GNU standard installation directories

比如变量：CMAKE\_INSTALL\_INCLUDEDIR

### 6. 生成器表达式

生成器表达式简单来说就是在CMake生成构建系统的时候根据不同配置动态生成特定的内容。比如：

* 条件链接，如针对同一个编译目标，debug版本和release版本链接不同的库文件
* 条件定义，如针对不同编译器，定义不同的宏

生成器表达式的格式形如$<...>，可以嵌套，可以用在很多构建目标的属性设置和特定的CMake命令中。值得强调的是，生成表达式被展开是在生成构建系统的时候

#### 布尔生成器表达式

* 逻辑运算符

CMake生成器表达式中有这些：

$<BOOL:string>：如果字符串为空、0；不区分大小写的FALSE、OFF、N、NO、IGNORE、NOTFOUND；或者区分大小写以-NOTFOUND结尾的字符串，则为0，否则为1

$<AND:conditions>：逻辑与，conditons是以逗号分割的条件列表

$<OR:conditions>：逻辑或，conditons是以逗号分割的条件列表

$<NOT:condition>：逻辑非

* 字符串比较

$<STREQUAL:string1,string2>：判断字符串是否相等

$<EQUAL:value1,value2>：判断数值是否相等

$<IN\_LIST:string,list>：判断string是否包含在list中，list使用分号分割

* 变量查询

根据不同CMake内置变量生成不同配置，核心就在于“判断”：

$<TARGET\_EXISTS:target>：判断目标是否存在

$<CONFIG:cfgs>：判断编译类型配置是否包含在cfgs列表（比如"release,debug"）中；不区分大小写

$<PLATFORM\_ID:platform\_ids>：判断CMake定义的平台ID是否包含在platform\_ids列表中

$<COMPILE\_LANGUAGE:languages>：判断编译语言是否包含在languages列表中

#### 字符串值生成器表达式

使用生成器表达式的主要目的：**生成特定的字符串**。

主要有两个格式：

$<condition:true\_string>：如果条件为真，则结果为true\_string，否则为空

$<IF:condition,str1,str2>：如果条件为真，则结果为str1，否则为str2

举例

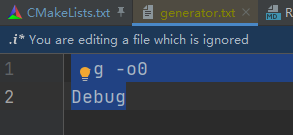
add\_compile\_options("$<$<CONFIG:Debug>:-g;-o0>")

#### 调试生成器表达式

值得强调的是，生成表达式被展开是在生成构建系统的时候，所以不能通过解析配置CMakeLists.txt阶段的message命令打印，其调试方法:

调试可以通过输出到文件的方式，在cmake执行完之后去检查是否符合预期，比如：

file(GENERATE OUTPUT "./generator.txt" CONTENT "$<$<CONFIG:Debug>: -g -o0> \n$<CONFIG>")



#### 更多生成器例子

target\_include\_directories(leveldb

PUBLIC

$<BUILD\_INTERFACE:${PROJECT\_SOURCE\_DIR}/include>

$<INSTALL\_INTERFACE:${CMAKE\_INSTALL\_INCLUDEDIR}>

)

$<INSTALL\_INTERFACE:...>

Content of ... when the property is exported using install(EXPORT), and empty otherwise.

$<BUILD\_INTERFACE:...>

Content of ... when the property is exported using export(), or when the target is used by another target in the same buildsystem. Expands to the empty string otherwise.

target\_sources(leveldb

PRIVATE

"${PROJECT\_BINARY\_DIR}/${LEVELDB\_PORT\_CONFIG\_DIR}/port\_config.h"

"port/port\_stdcxx.h"

"port/port.h"

"port/thread\_annotations.h"

"util/coding.cc"

"util/coding.h"

"util/random.h"

"util/mutexlock.h"

# Only CMake 3.3+ supports PUBLIC sources in targets exported by "install".

$<$<VERSION\_GREATER:CMAKE\_VERSION,3.2>:PUBLIC>

"${LEVELDB\_PUBLIC\_INCLUDE\_DIR}/comparator.h"

"${LEVELDB\_PUBLIC\_INCLUDE\_DIR}/iterator.h"

"${LEVELDB\_PUBLIC\_INCLUDE\_DIR}/slice.h"

"${LEVELDB\_PUBLIC\_INCLUDE\_DIR}/status.h"

"${LEVELDB\_PUBLIC\_INCLUDE\_DIR}/export.h"

"${LEVELDB\_PUBLIC\_INCLUDE\_DIR}/env.h"

)

$<VERSION\_GREATER:v1,v2>

1 if v1 is a version greater than v2, else 0.

### 7. 添加生成库的target

**add\_library**(tar\_name\_name "")

**target\_sources**(tar\_name\_name

PRIVATE

"util/mutexlock.h"

$<$<VERSION\_GREATER:CMAKE\_VERSION,3.2>:PUBLIC>

"${LEVELDB\_PUBLIC\_INCLUDE\_DIR}/comparator.h"

)

**target\_include\_directories**(leveldb

PUBLIC

$<BUILD\_INTERFACE:${PROJECT\_SOURCE\_DIR}/include>

$<INSTALL\_INTERFACE:${CMAKE\_INSTALL\_INCLUDEDIR}>

)

# -D LEVELDB\_COMPILE\_LIBRARY -D LEVELDB\_PLATFORM\_NAME=1

**target\_compile\_definitions**(leveldb

PRIVATE

# Used by include/export.h when building shared libraries.

LEVELDB\_COMPILE\_LIBRARY

# Used by port/port.h.

${LEVELDB\_PLATFORM\_NAME}=1

)

**target\_link\_libraries**(leveldb crc32c)

target\_compile\_definitions

Add compile definitions to a target.

### 8. enable\_testing

Enable testing for current directory and below.

This command is automatically invoked when the CTest module is included, except if the BUILD\_TESTING option is turned off.

### 9. add\_test

add\_test(NAME <name> COMMAND <command> [<arg>...]

[CONFIGURATIONS <config>...]

[WORKING\_DIRECTORY <dir>]

[COMMAND\_EXPAND\_LISTS])

Add a test called <name> to the project to be run by ctest.

COMMAND

Specify the test command-line. If <command> specifies an executable target (created by add\_executable()) it will automatically be replaced by the location of the executable created at build time.

### 10. install

Specify rules to run at install time.