CMake 简单介绍

Posted by
子鸣
on 2013 年 4 月 23 日

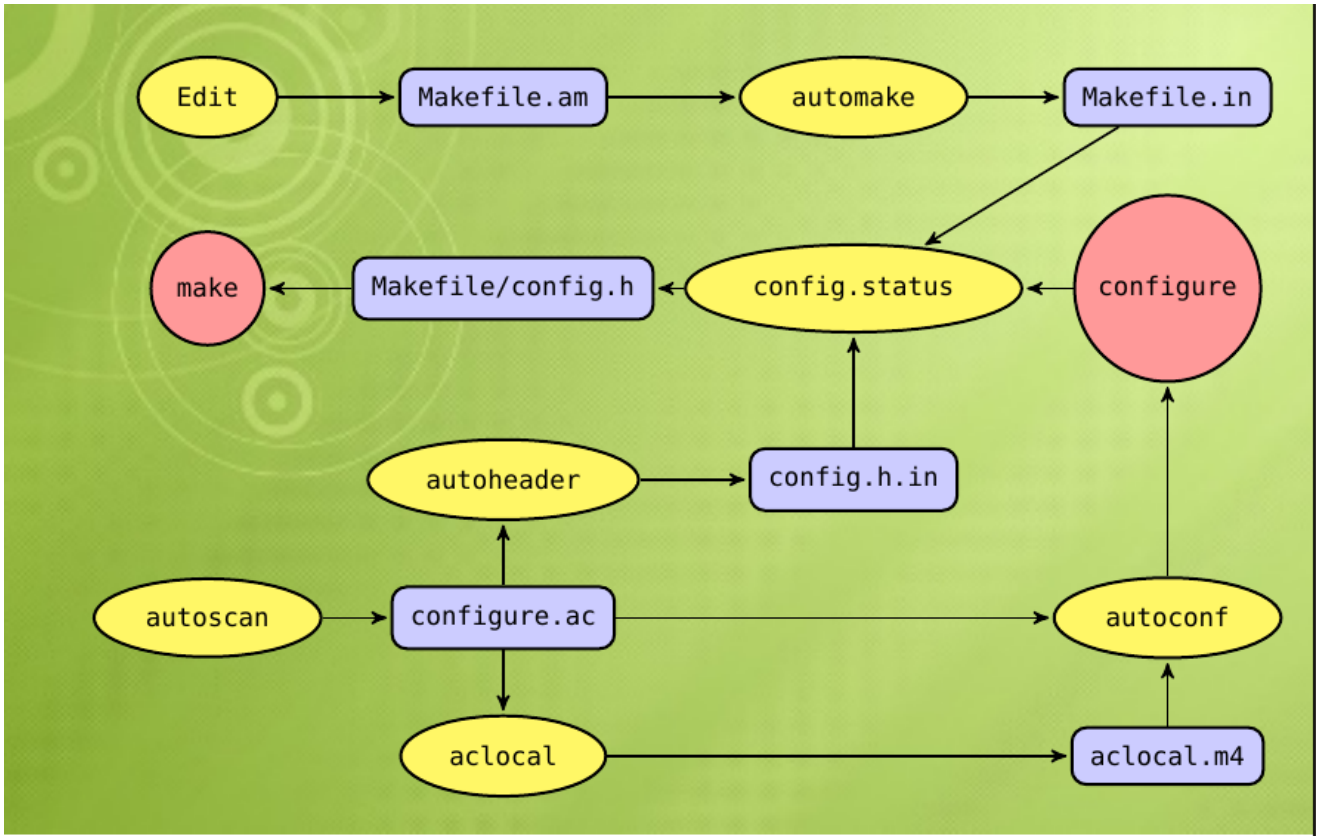
CMake特点

- CMake需要用户用CMake规范的语法编写CMake脚本，该语法简单易用，入门极其顺手
- 原生支持 C/C++/Fortran/Java 的相依性的自动分析功能，免除了程序员对代码依赖的调整，对整个开发工作帮助很大
- 支持 SWIG、Qt、FLTK 开发框架
- 支持跨平台编译，这是CMake名字的来源
- 能够转换特殊平台的 IDE 项目文档,如xcode
- 与Dart、CTest 和 CPack 集成，可以组成自动化的构建系统

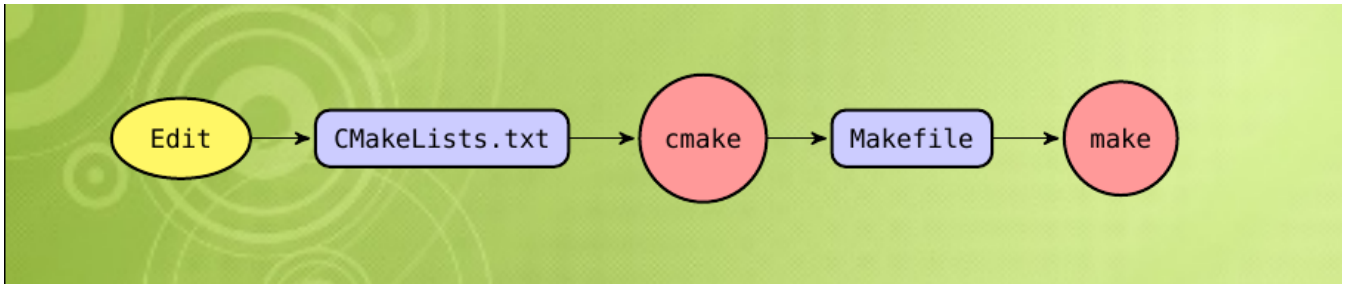
CMake vs GNU AutoTools

使用CMake，程序员必需编写好CMake脚本CMAkeLists.txt，对于一些复杂的项目，可能需要编写CMake模块，但对于构建过程而言，则是极其简单的。
相形之下，GNU AutoTools是很复杂的，无论是程序员在撰写Makefile.am，还是构建时的步骤。致命的是，GNU AutoTools间很难兼容。

GNU AutoTools构建流程

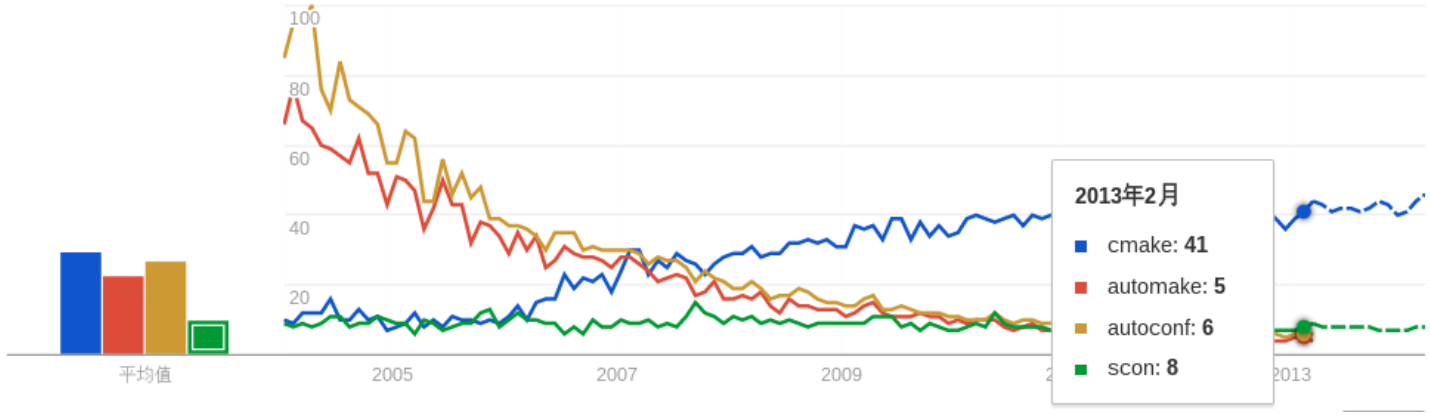


CMake 构建流程



另外，CMake的构建过程会比GNU AutTools在速度上有优势，在输出上更是很友好。

下面的是来自Google的趋势图，可以看出CMake的应用情况。



简单的CMake项目

我们先来看个简单的CMake项目。这里有4个文件：CMakeLists.txt, main.cpp, pr.cpp, pr.h。

CMakeLists.txt的内容如下：

```
project(MyProject)
cmake_minimum_required(VERSION2.8)
add_library(pr SHARED pr.cpp)

target_link_libraries(pr m)
add_library(prs STATIC pr.cpp)
target_link_libraries(prs m)
set_target_properties(prs PROPERTIES OUTPUT_NAME pr)
add_executable(mypr main.cpp)
target_link_libraries(mypr pr)

set(PR_HEADER pr.h)
install(TARGETS mypr pr prs
RUNTIME DESTINATION bin
LIBRARY DESTINATION lib
ARCHIVE DESTINATION lib/static)
install(FILES ${PR_HEADER}
DESTINATION include)
```

这个CMake脚本声明了此项目名为MyProject，并要求cmake的最少版本为2.8。
通常情况下2.6以上的特性不会用到，除非在一些复杂的项目中要自己处理一些问题而用到例如function这样的宏。

- add_executable的作用是指示CMake生成一个可执行文件。
- add_library的作用是指示CMake生成一个库，根据参数SHARED还是STATIC来决定生成动态库还是静态库。
- target_link_libraries指示传递给ld的外部库，相当于传给gcc的-lxx。
- set_target_properties则是用于修改构建目标的一些属性，如上的语句则是让目标prs的输出libprs.a改名为libpr.a。
- set是CMake中常用的命令，用于设置变量，也能修改内部变量
- install用于生成make install使用的语句

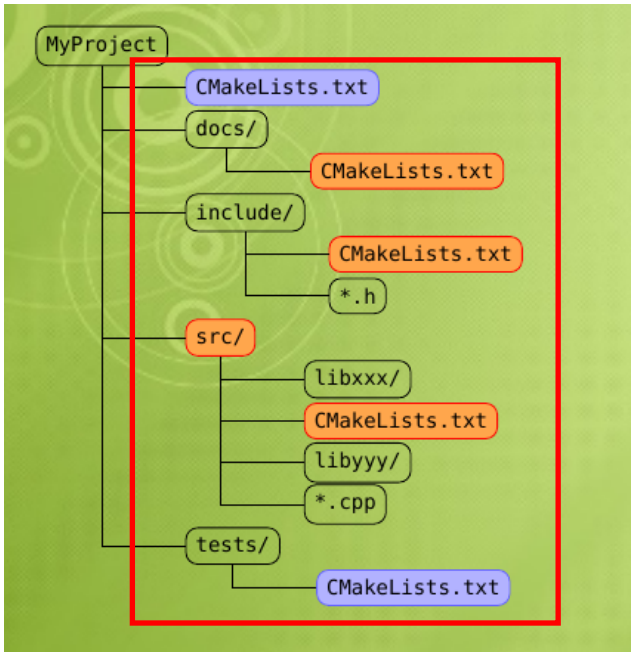
要构建这个项目，可在所在目录建立一个目录，例如名为build，并切换到该目录，然后执行`cmake .. && make`

路径

一般的项目会涉及到多个模块，文档以及测试用例等。下图为一般CMake项目的文件布局。每一层目录如果包含有构建目标，一般都会有一个CMakeLists.txt。

包含CMakeLists.txt的子目录通常由上层的CMakeLists.txt用语句add_subdirectory来包含。对于这个项目而言，顶层的CMakeLists.txt包含如下的语句：

```
add_subdirectory(include)
add_subdirectory(src)
add_subdirectory(tests)
add_subdirectory(docs)
```



在多个模块的情况下，可能一个模块的链接依赖於其它模块，例如一个可执行二进制需要链接某些模块，此时link_directories将有发挥作用。
如在CMakeLists.txt增加：

```
link_directories(${MyProject_BINARY_DIR}/src/libxxx
${MyProject_BINARY_DIR}/src/libyyy)
```

将指示CMake在LDFLAGS附加-Lsrc/libxxx -Lsrc/libyyy。

放置在src下的代码要引用include的头文件，可以使用相对路径引用，也可以让include放置在头文件搜索路径中，即CPPFLAGS，-Iinclude。如何指示CMake呢？只需要加上include_directories(BEFORE include)，或者include_directories(BEFORE ${MyProject_SOURCE_DIR}/include)，前者引用CMakeLists.txt的相对路径，而后者则是依项目的完整路径。

其它

在上面提及到set可以修改一些内部变量，例如可以修改编译器的参数，如

```
set(CMAKE_C_FLAGS "-std=c99 -O2 -pipe -Wall -Wextra")
set(CMAKE_C_FLAGS_DEBUG "${CMAKE_C_FLAGS} -g -ggdb -pg")
set(CMAKE_C_FLAGS_RELEASE "${CMAKE_C_FLAGS} -s -DNDEBUG")
```

CMake还提供了一些find函数，如find_package、find_library，用于检查项目构建的依赖或者控制功能的开启。
这些功能已经超出今天的简单介绍，有兴趣的朋友请参考官方资讯。

参考

http://www.cmake.org/Wiki/CMake:Articles
http://www.cmake.org/Wiki/CMake_FAQ
http://techbase.kde.org/Development/CMake
http://techbase.kde.org/Policies/CMake_Coding_Style

您可能感兴趣的文章

- 开源PHP监控扩展：witness简介
- 使用libevent的http框架的一点小技巧
- Python程序的执行原理

Tags: CMake

发表评论

电子邮件地址不会被公开。 必填项已用*标注