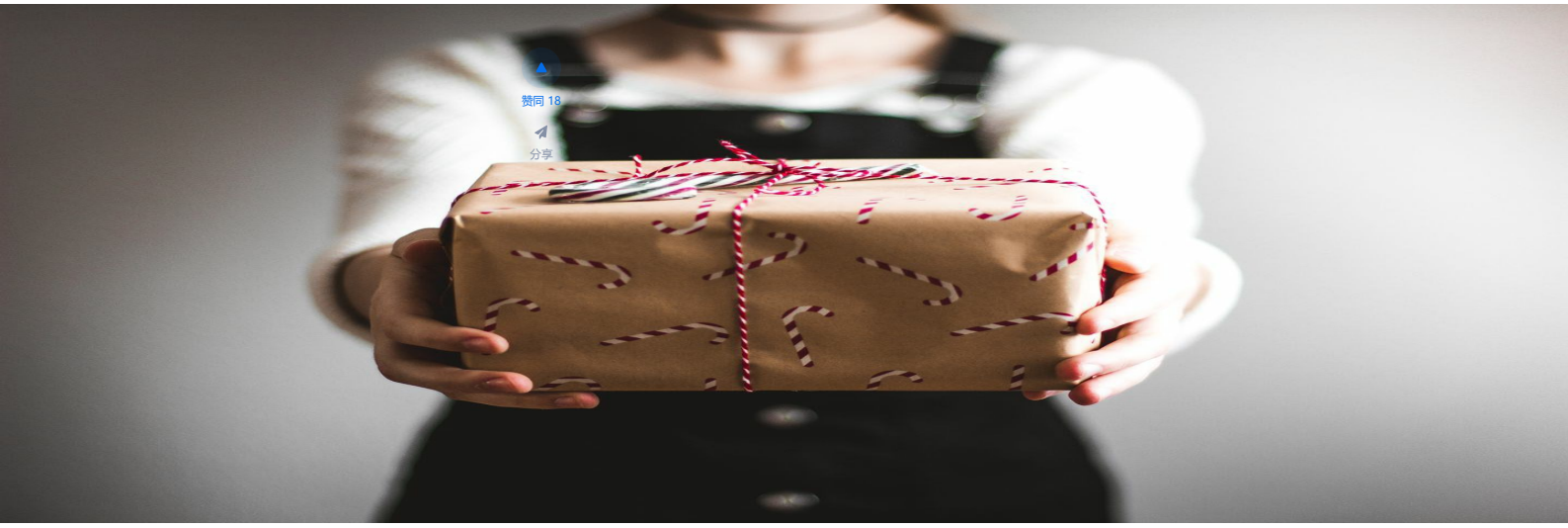


使用 pkg-config 让 C++ 工程编译配置更灵活

淇右

背景与问题

在进行 C++ 项目开发的过程中，多少会依赖一些外部库，这些依赖有些可能是通过 git submodule 源码的形式直接引入到自己的工程中进行静态链接，而有时希望控制编译产物的大小，把一些比较通用的依赖希望以动态链接的形式依赖，比如图片编解码库 libpng、libwebp 或者文本塑形库 harfbuzz 之类的。

这种情况下在进行编译配置的时候，需要依赖的库不一定都在 `/usr/local/include` 目录下，大部分情况下需要对编译器手动指定 include 目录，同理也可能需要对链接器手动指定 lib 所在目录，而由于不同系统下安装的库所在的目录不一定一致，会导致编译配置难以跨平台执行。

我们可以用 pkg-config 来帮我们解决这个问题，当需要依赖某个外部库时，可以让 pkg-config 来告诉我们它在当前环境下要依赖它的话，编译链接命令应该是什么，让我们看下具体是如何应用的吧。

安装 pkg-config

pkg-config 最初是为 Linux 开发的，但目前是跨平台的，支持 Mac、Linux、Windows，如果你的环境下还没有 pkg-config 的话，首先安装它：

```
# macos
brew install pkg-config

# ubuntu
apt-get install pkg-config

# 其他平台相信你能找到办法
```

开始使用 pkg-config

pkg-config 本身是一个命令行工具，我们可以试着输出一个比如 libwebp 在当前环境的编译选项：

```
pkg-config libwebp --cflags
```

在我的环境下的输出为：

```
-I/usr/local/Cellar/webp/1.2.0/include
```

再试着输出一下 libwebp 的链接选项：

```
pkg-config libwebp --libs
```

在我的环境下的输出为：

```
-L/usr/local/Cellar/webp/1.2.0/lib -lwebp
```

可以指定多个选项让 pkg-config 一起输出，比如一起输出 cflags 和 libs 选项：

```
pkg-config libwebp --libs --cflags
```

此时的输出为：

```
-I/usr/local/Cellar/webp/1.2.0/include -L/usr/local/Cellar/webp/1.2.0/lib -lwebp
```

你甚至可以在一条命令中指定多个外部库，比如我同时指定 libwebp 和 libpng：

```
pkg-config libwebp libpng --libs --cflags
```

测试的输出为：

```
-I/usr/local/Cellar/webp/1.2.0/include -I/usr/local/Cellar/libpng/1.6.37/include/libp
```

可以发现它的输出是可以直接应用在编译工具上的，所以在命令行下和编译工具可以直接结合起来：

```
gcc -o test test.c `pkg-config --libs --cflags libwebp`
```

你甚至可以用 pkg-config 查看外部库在当前环境下的版本号：

```
pkg-config libwebp --version
```

可以命令参数可以参考 --help 的输出。

在 CMake 中使用 pkg-config

当然一般正式的项目都会使用元构建工具来进行跨平台的工程编译配置的生成，我这边使用的是 CMake，在 CMake 中也可以非常方便的使用 pkg-config：

```
find_package(PkgConfig REQUIRED)

if (PKG_CONFIG_FOUND)
  pkg_check_modules(my_deps REQUIRED IMPORTED_TARGET libpng libwebp)
endif()
```

首先利用 CMake 的 find_package 机制找到本地的 pkg-config，如果成功找到，则有两种办法查找外部库：

- pkg_check_modules：根据列表中给的外部库，在当前环境下都试着去找到
- pkg_search_module：找到列表中第一个成功找到的外部库

可以根据实际需求使用，大部分情况下使用 pkg_check_modules，第一个参数为匹配前缀，当你需要依赖多个外部库时，通过这个前缀，可以一次性的消费结果。你也可以指定 REQUIRED 来表示依赖对这次构建是必须的，否则直接失败终止构建。接下来就是应用 pkg-config 的结果了，在 CMake 上下文中可以使用 `PkgConfig::${prefix}` 来消费结果，直接作为 target_link_libraries 的参数即可：

```
target_link_libraries(${PROJECT_NAME} PkgConfig::my_deps)
```

如果你的 CMake 版本小于 3.6，也可以使用一下变量：

- <prefix>_LDFLAGS
- <prefix>_CFLAGS

关于在 CMake 中使用 pkg-config 的更多细节，可以参考其官方文档：

> FindPkgConfig - CMake 3.21.3 Documentation
> cmake.org/cmake/help/latest/module/FindPkgConfig....

pkg-config 如何查找依赖

如果我们让 pkg-config 查找一个未安装的外部库会怎么样？

```
pkg-config libxxx --cflags
```

会输出：

```
Package libxxxx was not found in the pkg-config search path.
Perhaps you should add the directory containing `libxxxx.pc'
to the PKG_CONFIG_PATH environment variable
No package 'libxxxx' found
```

在聊如何解决之前，首先需要介绍下 pkg-config 是如何查找依赖的，首先介绍一些 pc 文件，它在上面的错误信息中也出现了，pkg-config 是通过读取目录下的 pc 文件在确定查找结果的，这个目录通常是 libdir/pkgconfig，比如你的 libwebp 安装在 /usr/local/lib 下，那么放 pc 文件就是 /usr/local/lib/pkgconfig 下的 libwebp.pc 文件。pc 文件就是普通的文本文件，我们来看一下 pc 文件的内容：

```
prefix=/usr/local/Cellar/webp/1.2.0
exec_prefix=${prefix}
libdir=${exec_prefix}/lib
includedir=${prefix}/include

Name: libwebp
Description: Library for the WebP graphics format
Version: 1.2.0
Cflags: -I${includedir}
Libs: -L${libdir} -lwebp
Libs.private: -lm -D_THREAD_SAFE -pthread
```

可以看到这个文件中定义了这个依赖的名称、版本号、在当前环境的目录前缀、编译选项等等，看到这里你应该明白 pkg-config 的输出是哪里来的了。

外部依赖之间可能也会有依赖关系，pkg-config 会帮你解决这个依赖问题，其实库的依赖也是会在 pc 文件中定义的，比如我们看下 harfbuzz 的 pc 文件：

```
prefix=/usr/local/Cellar/harfbuzz/2.8.0_1
libdir=${prefix}/lib
includedir=${prefix}/include

Name: harfbuzz
Description: HarfBuzz text shaping library
Version: 2.8.0
Requires.private: freetype2, graphite2, glib-2.0
Libs: -L${libdir} -lharfbuzz
Libs.private: -lm -framework ApplicationServices
Cflags: -I${includedir}/harfbuzz
```

可以发现这个库同时依赖了 freetype2、graphite2、glib-2.0 这三个库，只要在 pc 文件中有通过 Requires 或者 Requires.private 声明过依赖，在 cflags 或者 libs 的输出结果中也会带有依赖的编译选项：

```
-I/usr/local/Cellar/harfbuzz/2.8.0_1/include/harfbuzz -I/usr/local/opt/freetype/inclu
```

通常一个 lib 对应一个 pc 文件，而有时有些项目有多个 lib，那么它也会分别定义多个 pc 文件，比如 ffmpeg 就有这些 pc 文件：

- libavcodec.pc
- libavdevice.pc
- libavfilter.pc
- libavformat.pc
- libavutil.pc

pkg-config 查找路径配置

其实在使用 pkg-config 过程中遇到库找不到的情况，不一定是外部库没有安装，默认情况下 pkg-config 的查找路径为 /usr/lib/pkgconfig 和 /usr/share/pkgconfig，可以通过环境变量 PKG_CONFIG_PATH 在额外指定 pkg-config 的查找路径，比如：

```
export PKG_CONFIG_PATH="/usr/local/opt/icu4c/lib/pkgconfig:${PKG_CONFIG_PATH}"
export PKG_CONFIG_PATH="/usr/local/opt/jpeg-turbo/lib/pkgconfig:${PKG_CONFIG_PATH}"
```

生成 pc 文件

目前大部分知名的库应该都存在 pc 文件，如果你的依赖没有带 pc 文件，你可以自己为它编写 pc 文件，可是不难发现，如果手动编写 pc 文件，那么文件内的路径依然是和当前环境绑定的绝对路径，那么问题还是没有解决，大部分情况下使用工具自动生成 pc 文件才是正确的姿势，这里简单介绍一下如何使用 Autotools 生成 pc 文件，首先需要在你的项目中定义 `pc.in` 文件作为模板，比如 libwebp 项目的 `pc.in` 文件：

```
prefix=@prefix@
exec_prefix=@exec_prefix@
libdir=@libdir@
includedir=@includedir@

Name: libwebp
Description: Library for the WebP graphics format
Version: @PACKAGE_VERSION@
Cflags: -I${includedir}
Libs: -L${libdir} -lwebp
Libs.private: -lm @PTHREAD_CFLAGS@ @PTHREAD_LIBS@
```

其中 @variable@ 是变量，会被 configure 替换。接下来就是在 configure.ac 文件中通过 AC_CONFIG_FILES 添加 pc 文件，接下来在库被安装的时候，通过 configure 正确设置变量后，就可以动态生成 pc 文件和库的其他文件一起安装在合适的位置了。

最后

通过 pkg-config 基本解决了跨平台编译是三方库依赖的编译配置问题，并且使用简单，可以很简单的和命令行或者很多元构建工具结合起来使用。当然 pkg-config 并不是包管理工具，虽然大部分知名的库都有 pc 文件，但有时也需要在特定环境下为 pkg-config 配置正确的查找路径。如果是你自己定义的库需要被别的项目愉快的依赖，那么就要自己通过工具去生成 pc 文件啦。

发布于 2021-10-05 20:42

C / C++ CMake 编程

写下你的评论...

发布

还没有评论，发表第一个评论吧

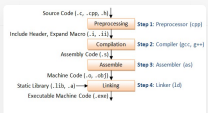
文章被以下专栏收录

欲买桂花同载酒
分享一些技术成长道路上的总结或者畅想。

推荐阅读

如何在Linux中编译和运行C / C ++ 程序，简单示例教懂你
C是一种过程编程语言。它最初是由Dennis Ritchie在1969年至1973年之间开发的。它主要是作为用于编写操作系统的系统编程语言而开发的。C语言的主要功能包括对内存的低级访问，一组简单的关键...
编程重度爱好者


英特尔最新版 C/C++ 编译器采用 LLVM 架构，性能提升...
InfoQ


C/C++程序编译过程为什么要分为四个步骤？
李豪 发表于有深度的学...

C++高性能编程笔记（第4讲编译器优化与SIMD指令集）
该笔记来源于小彭老师的高性能公开课：【公开课】编译器优化与SIMD指令集（#4）_哔哩哔哩_bilibili x64架构下的寄存器模型 寄存器的读写速度比内存快，如果读到寄存器里去计算就比较高效。...
北方有匈奴 发表于C++高性...