

# DynamicX控制组、视觉组、rmua组联合培训 **CMake与Catkin**

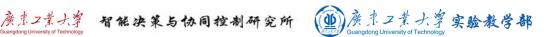
广东工业大学DynamicX 机器人队





## CMake简介

CMake是一种过程式语言, 是一个跨平台的安装(编译)工具, 可以用简单的语句来描 述所有平台的安装(编译过程)。他能够产生标准的建构档makefile或者project文件 , CMake 的组态档(即configuration file)取名为 CMakeLists.txt。也就是在 CMakeLists.txt这个文件中写CMake代码。

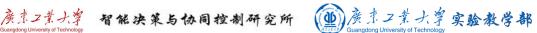


```
cmake_minimum_required(VERSION 3.10)
指定最小 CMake 版本要求:
```

project(example0) 设置项目名称:

add\_executable(executableFile main.cpp answer.cpp) 用来添加可执行目标target。





## CMake例子1

set——将普通、缓存或环境变量设置为给定值。

set(CMAKE\_CXX\_STANDARD 14)

通过为CMAKE CXX STANDAND赋值, 指定要使用 C/C++ 的什么版本;

set(CMAKE\_CXX\_ STANDARD REQUIRED ON)

通过为CMAKE CXX STANDARD REQUIRED赋值,设置指定的C++编译器版本是 必须的, 如果不设置, 或者为OFF, 则指定版本不可用时, 会使用上一版本;

add definitions(-Wall -Werror)

为当前目录及以下目录中的源代码向编译器命令行添加标志(flag),此命令可用于添 加任何标志(flag),但旨在添加预处理器定义。<sub>详细链接</sub>





## CMake例子2

add library(libanswer STATIC OtherFile.cpp)

添加 libanswer 库目标, STATIC 表示libanswer是个静态库。

STATIC, 代表静态链接库, 编译的时候link到工程中静态库是 .a(或在 Windows 中的 .lib)文件 ,使用静态库的程序从静态库中获取它使用的代码的拷贝, 并使其成为程序的一部分;

SHARED,代表动态链接库,运行时候加载,使用共享库的程序只引用它在共享库中使用的代 码, 共享库是 .so(或在 Windows .dll 中, 或在 OS X .dylib 中)文件;

详细链接&详细链接





### Catkin

CMakeLists.txt

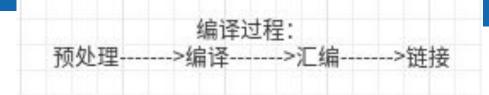
add dependencies(\${PROJECT NAME} libanswer)

一个顶层target是由命令ADD EXECUTABLE, ADD LIBRARY, 或者 ADD CUSTOM TARGET产生的目标。定义目标target依赖于其他目标target,确保 其他target已被build, 让一个顶层target依赖于其他的顶层target。用到的情况就是两 个targets有依赖关系(通过target link libraries解决)并且依赖库也是通过编译源码产 生的。这时候一句add dependencies可以在直接编译上层target时,自动检查下层依 赖库是否已经生成。没有的话先编译下层依赖库,然后再编译上层target,最后link depend target.

本例中libanswer要在可执行目标之前编译生成。







target link libraries(\${PROJECT NAME}(关键字)libanswer) 为可执行目标\${PROJECT NAME}(即example2)链接libanswer库

本例将可执行文件target命名为\${PROJECT NAME}, \${PROJECT NAME}是 CMake的一个宏, 其值为项目的名字, 本例中项目名为example2。

中间其实也有个关键字,默认情况下是PUBLIC

PUBLIC:表示被链接的库会被附加到targetA的公开接口,这个被链接的库能被用于targetA ,链接此targetA的targetB也可以使用这个被链接的库;

PRIVATE:表示链接是 targetA 的私有内容, 不应对使用targetA的 其他targetB 产生影响;

INTERFACE:表示被链接的库会被附加到targetA的公开接口, 但不能被用于targetA。

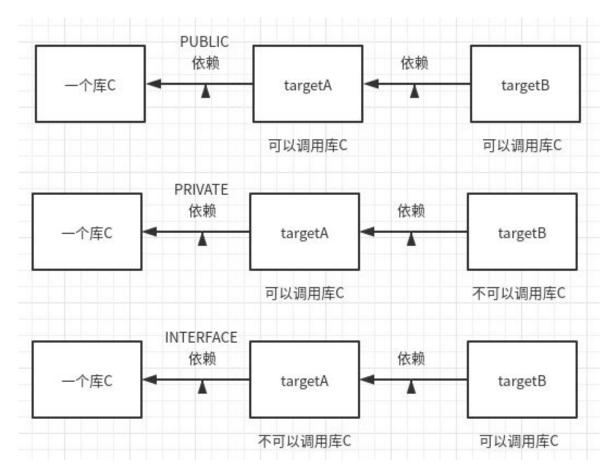




**PUBLIC:** 

PRIVATE:

INTERFACE:







```
add subdirectory(Folder)
将子目录Folder添加到build中:
target compile features(libanswer PRIVATE cxx std 20)
向target添加预期的编译器功能,可以指定细粒度更高的 C++ 特性,例如
cxx auto type、cxx lambda 等;
```

message(STATUS "libanswer INCLUDE DIRS: \$ENV{libanswer INCLUDE DIRS}") 可用于打印调试信息或错误信息,除了 STATUS外还有 AUTHOR WARNING、 

## CMake例子3

include directories(\$ENV{libanswer INCLUDE DIRS}) 将给定的目录添加到编译器用于搜索头文件的目录中,添加的目录范围在本CMakeList.txt及其子 目录中有效:

target\_include\_directories(libanswer PUBLIC \${CMAKE\_CURRENT\_SOURCE\_DIR}/include) 作用和include directories差不多,也可以由PUBLIC、PRIVATE、INTERFACE控制传递;





## CMake例子3

```
set (ENV{libanswer INCLUDE DIRS} "${CMAKE CURRENT SOURCE DIR}/include")
设置名为libanswer INCLUDE DIRS的环境变量, 值为Folder/include的绝对路径
   set(<variable> <value>... [PARENT SCOPE]):设置普通变量
   set(<variable> <value>... CACHE <type> <docstring> [FORCE]):设置缓存变量
   set(ENV{<variable>} [<value>]):设置环境变量
```

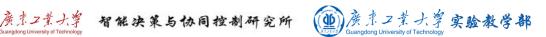
详细链接1&详细链接2





### CMake例子4

```
find package(Boost REQUIRED COMPONENTS <...>)
指定依赖的其他pacakge, 并生成了一些环境变量, 如<NAME>_FOUND,
<NAME>_INCLUDE_DIRS, <NAME> LIBRARYIS。REQUIRED强调必须找到该模
块,COMPONENTS后面跟着必须有的组件。
```

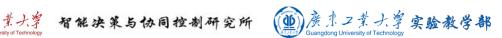


```
set(HELLO "" CACHE STRING "HELLO")
Cache 变量的值可在命令行调用 cmake 时通过 -D 传入. 比如:
cmake -B build -DHELLO=xxx
```

target compile definitions(\${PROJECT NAME} PRIVATE HELLO="\${HELLO}") 将 HELLO 添加到编译 .cpp 文件时的 definition 列表, 从而可在 C++ 代码中使用。

if(HELLO STREQUAL "") endif() CMake的逻辑判断语句, 这里是判断HELLO是否为空 详情链接&详情链接





标记要安装的target

```
install(
    TARGETS ${PROJECT_NAME}
${PROJECT_NAME}_core
    ARCHIVE DESTINATION lib
    LIBRARY DESTINATION lib
    RUNTIME DESTINATION bin
```

TARGETS:要安装的目标

ARCHIVE DESTINATION:静态库和 动态链接库DLL(Windows).lib存根

LIBRARY DESTINATION: 非DLL共享 库和模块

RUNTIME DESTINATION: 可执行目标和DLL(Windows)模式共享库





```
标记要安装的头文件
```

```
install(
   DIRECTORY ${CMAKE_CURRENT_SOURCE_DIR}/include/otherfile
   DESTINATION include
   FILES_MATCHING PATTERN "*.h"
标记要安装的其他文件
install(
   DIRECTORY ${CMAKE_CURRENT_SOURCE_DIR}/doc
   DESTINATION share
```





### Catkin

Catkin是基于CMake的编译构建系统, 具有以下特点:

Catkin沿用了包管理的传统像 find\_package()基础结构,pkg-config

扩展了CMake, 例如 软件包编译后无需安装就可使用 自动生成find\_package()代码, pkg-config文件 解决了多个软件包构建顺序问题 一个Catkin的软件包(package)必须要包括两个文件: package.xml:

包括了package的描述信息 name, description, version, maintainer(s), license, opt. authors, url's, dependencies, plugins, etc...

CMakeLists.txt:

构建package所需的CMake文件 调用Catkin的函数/宏 解析package.xml 找到其他依赖的catkin软件包 将本软件包添加到环境变量

#### 引用链接





### Catkin

CMakeLists.txt

```
catkin package() 是一个由catkin提供的CMake宏。需要指定特定的catkin信息到编译
系统, 而这些信息又会被用于生成pkg-config和CMake文件。该函数必须在使用
add library()或add executable()声明任何targets之前调用。其5个可选参数:
## The catkin package macro generates cmake config files for your package
## Declare things to be passed to dependent projects
## INCLUDE DIRS: uncomment this if your package contains header files
## LIBRARIES: libraries you create in this project that dependent projects also need
## CATKIN DEPENDS: catkin packages dependent projects also need
## DEPENDS: system dependencies of this project that dependent projects also need
```





### Catkin

CMakeLists.txt

catkin add gtest 构建一个可执行文件, 可调用此可执行文件进行测试; Gtest 是用于运行 C++ 单元测试的 Google 框架。

testing::InitGoogleTest(&argc, argv) 它初始化框架并且必须在 RUN ALL TESTS 之前调用;

RUN ALL TESTS()

会自动检测并运行使用 TEST 宏定义的所有测试。默认情况下,结果打印到标准输出。

详情链接&详情链接&详情链接 Z業大學 智能决策与协同控制研究所 (D) 廣東工業大學实验教学部

### Catkin

package.xml

- <name>-包的名字
- <version>-包的版本号(格式:xxx.xxx.xxx)
- <description>-包的内容描述
- <maintainer>-维护包的人员的名字
- se>-软件许可证(例如GPL, BSD, ASL, TODO)
- <author> 原作者名
- <url> 介绍本package的网站链接



### Catkin

package.xml

<package format="2"> 格式2

<bul><buildtool depend>

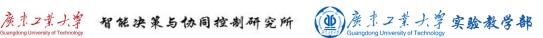
编译构建工具,构建工具依赖关系指定此软件包需要构建自身的构建系统工具。通常只有catkin;

<build depend>

编译依赖项,构建依赖关系指定构建此包所需的包,如果你只使用一些特定的依赖来构建你的包,而 不是在执行时,你可以使用 <build\_depend>标签;

<bul><build export depend> 指定包构需要哪些包用来build;

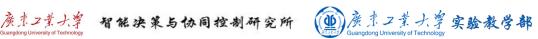




### Catkin

```
package.xml
<package format="2"> 格式2
<exec_depend>
运行依赖项, 指定运行此包中的代码需要哪些包, 或针对此包构建库;
<doc depend>
文档依赖项:
<depend>
指定依赖项为编译、导出、运行需要的依赖, 最常用,可以涵盖上面的
<build depend>, <build export depend>和<exec depend>;
```





### 主要参考链接:

https://sychaichangkun.gitbooks.io/ros-tutorial-icourse163/content/chapter2/

https://zhuanlan.zhihu.com/p/62344573

https://gist.github.com/mbinna/c61dbb39bca0e4fb7d1f73b0d66a4fd1#get-your-handsoff-cmake cxx flags

https://github.com/richardchien/modern-cmake-by-example

https://github.com/ttroy50/cmake-examples



