

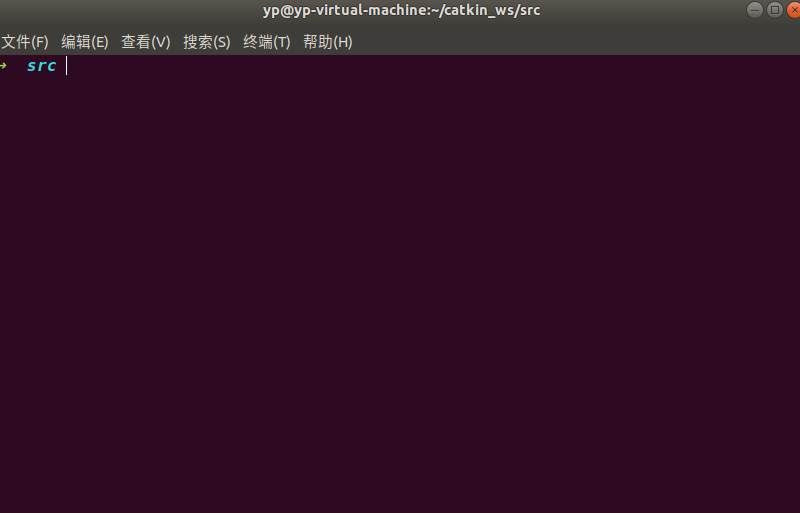
创建工作空间：

1.在主目录下，新建一个文件夹，并命名为catkin\_ws

2.在catkin\_ws目录下新建一个src文件夹

3.在src区域内，右击，选择在终端中打开

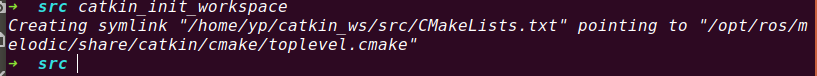
4.进入如图所示界面：



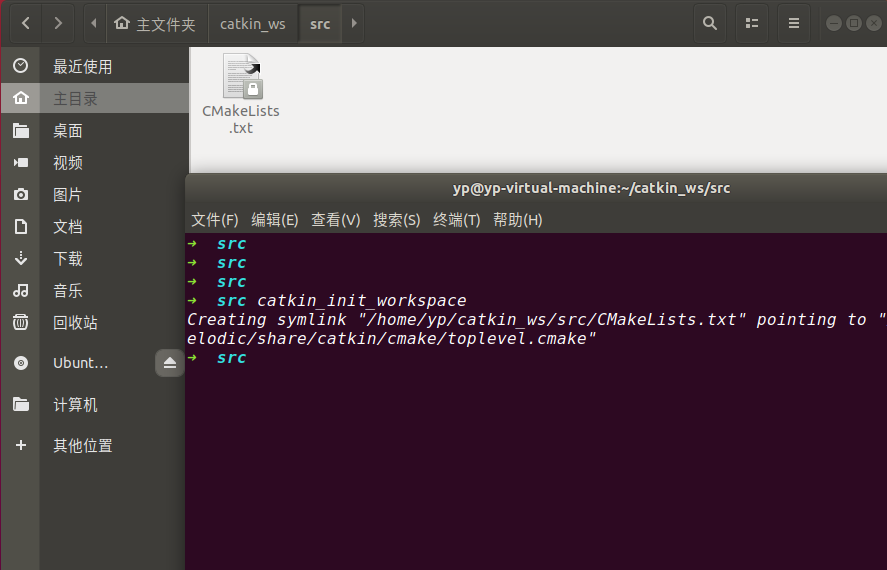
5.在终端中输入

catkin\_init\_workspace

得到下图所示：



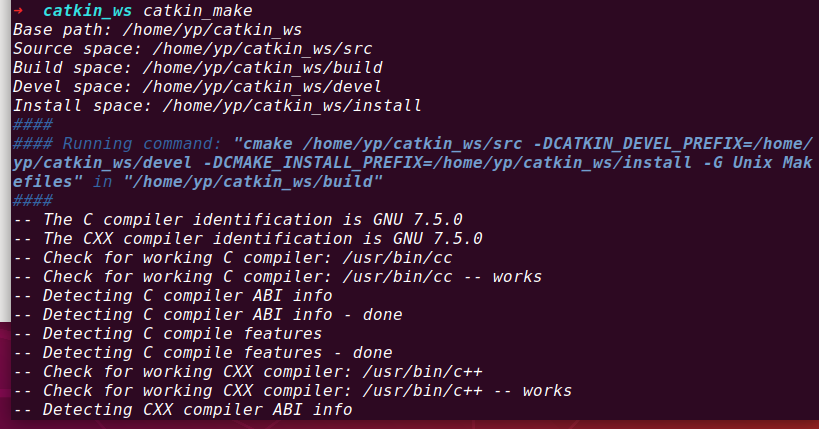
即在src目录下创建了一个CMake文件



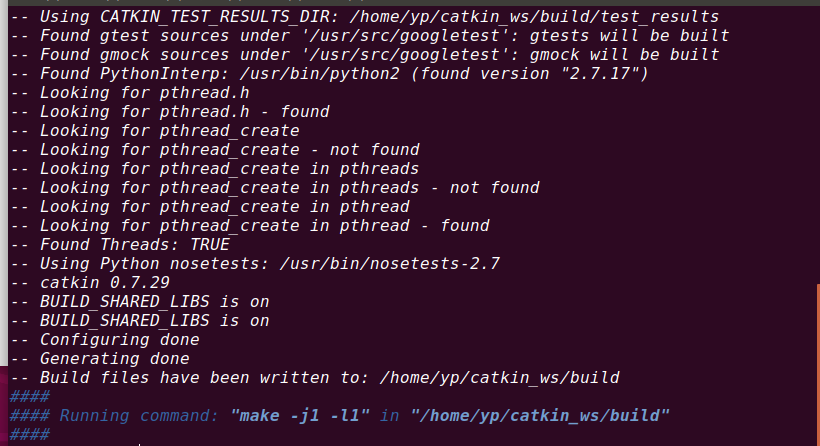
编译工作空间：

1.进入到catkin\_ws目录下;

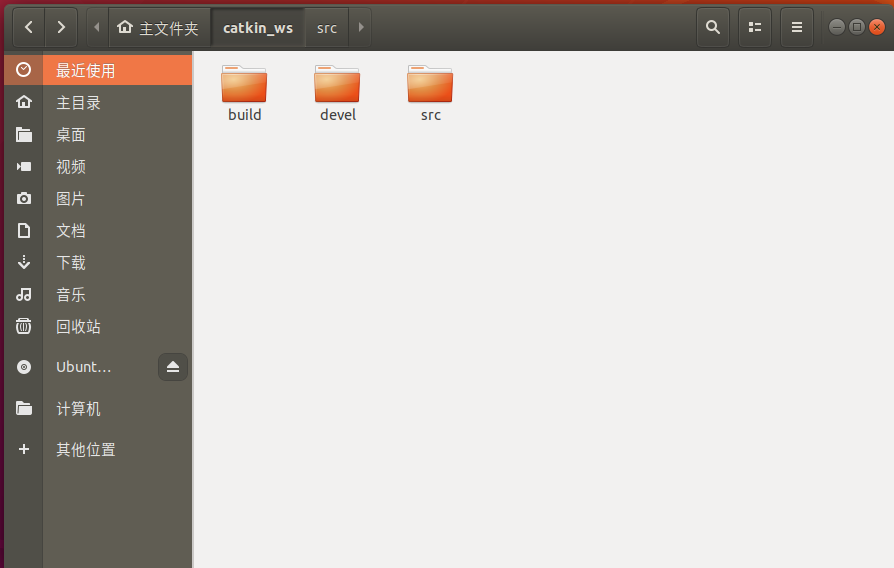
2.使用catkin\_make进行编译



显示如下界面，表示编译成成功



并且在src根目录下新出现了两个文件夹：



设置环境变量：作用是：让系统知道环境变量放在devel目录下

source devel/setup.zsh

//如果终端是zsh的话，后跟setup.zsh

//如果终端是bash的话，后跟setup.bash

//缺点就是这个环境变量的设置，只能在当前终端下生效，新开一个终端无效

通过如下设置：可以让新打开的环境变量也能实现相同的功能

vi ~/.zshrc

//翻到最后一行，在之前修改了终端bash为zsh后

//添加环境变量如下

source ~/catkin\_ws/devel/setup.zsh

//:wq保存退出

打印当前环境变量：

echo $ROS\_PACKAGE\_PATH

得出下图所示：

clipboard.png

冒号前的是刚才设置的环境变量，冒号后的是ROS安装时系统默认的环境变量地址



功能包是实现机器人功能所存放的具体代码

package\_name是功能包的名字

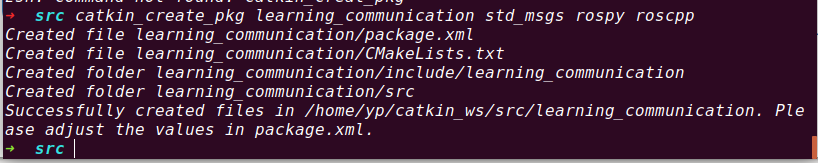
depend是功能包实现功能所依赖的依赖文件

创建功能包的路径要在刚才新建的文件夹src中进行

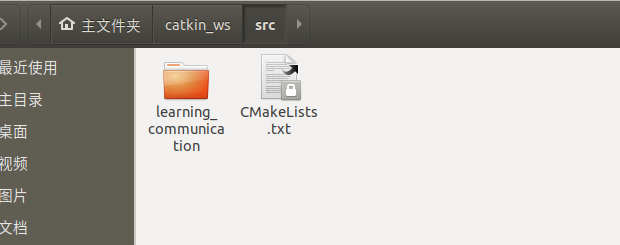
如图所示：创建了一个名为learning\_communication的功能包，所依赖的文件是类似于c语言的stdio.h文件的std\_msgs，python语言rospy，用C++编程的roscpp

catkin\_create\_pkg learning\_communication std\_msgs rospy roscpp

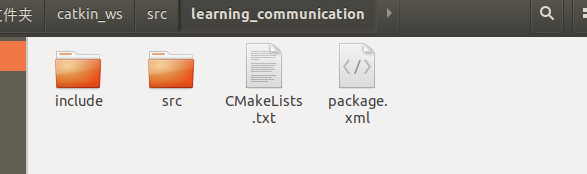
当出现下图所示的一些日志，就代表着创建成功：



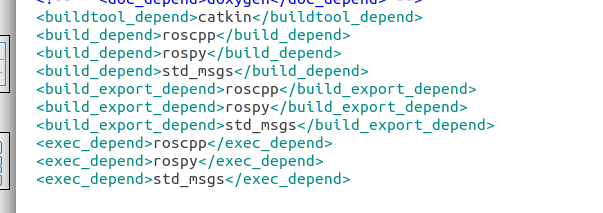
打开src文件：出现了新建的功能包文件



功能包文件里面出现了一些默认文件：



在package.xml文件里如下图：

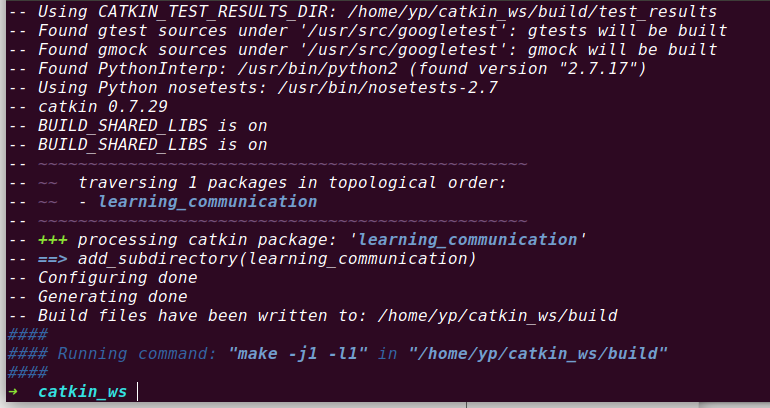


如果后面需要添加新的依赖文件，需要在上述文件中加上依赖文件

回到上级目录catkin\_ws下，运行指令：catkin\_make

因为src里面已经有一个功能包：learning\_package

运行如下：



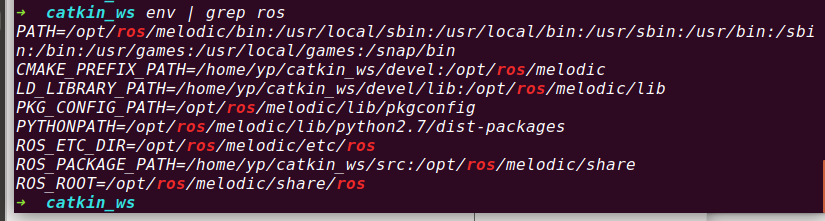
另外，在src中创建功能包时，不能出现同名功能包

那么问题来了，在ROS不同工作空间中出现了多个同名功能包，ROS是如何识别是哪一个功能包的，这依赖于ROS工作空间的Overlaying机制

使用指令：

env | grep ros

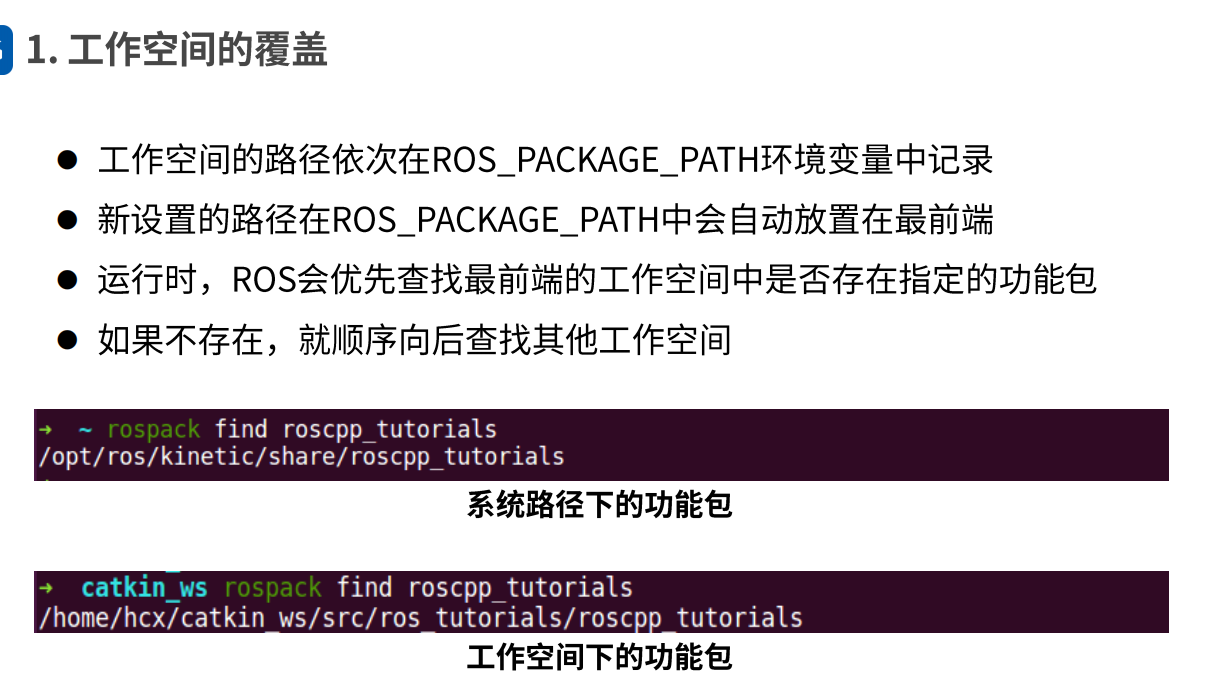
查找和ROS相关的指令，如下图：



倒数第二行如下：

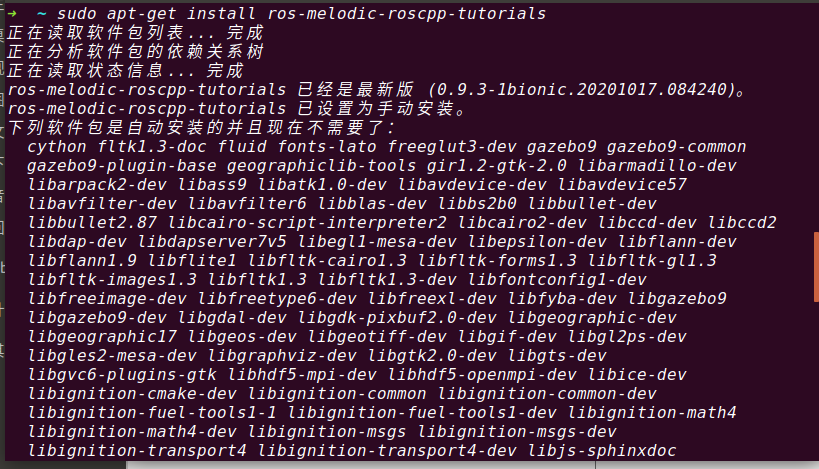
clipboard.png

可以查看ros功能包的所有路径，ros从前往后依次寻找



最后设置的环境变量往往最先被查找

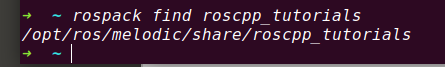
安装c++功能包：



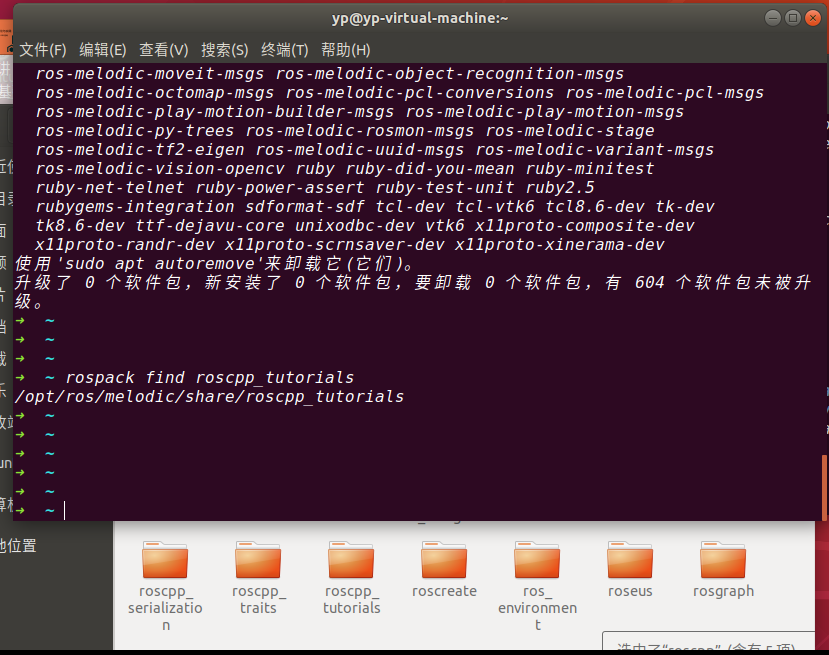
如果想要找到功能包的位置：

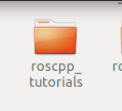
rospack find roscpp\_tutorials

如下图所示：

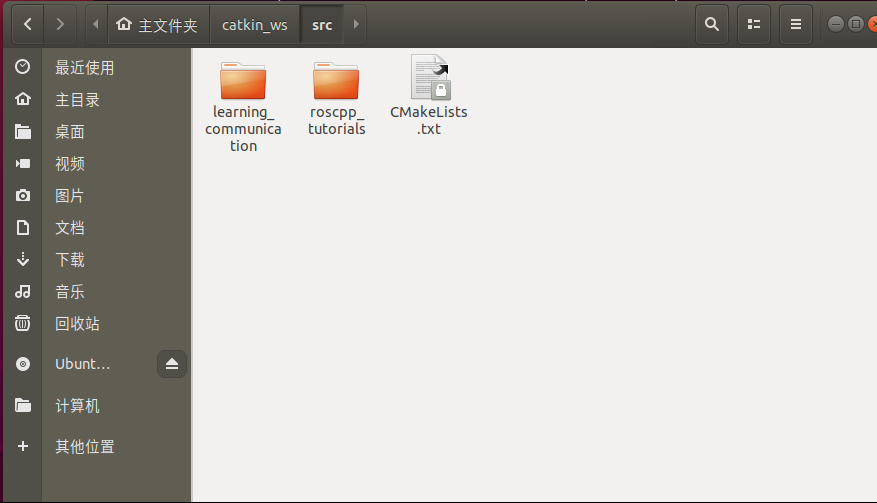


打开文件管理器查看：

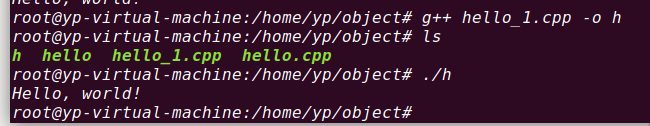




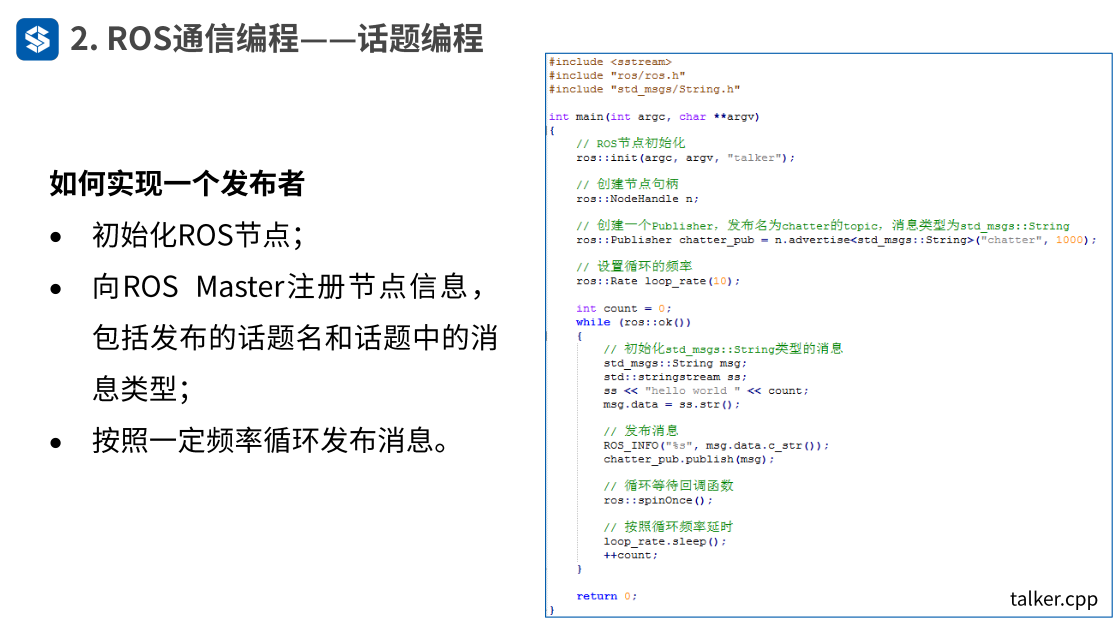
将roscpp\_tutorials复制到创建的工作目录src下：



g++编译C++：



在learning\_communication目录文件里面，进入src文件夹，创建talker.cpp文件



/\*\*

\* 该例程将发布chatter话题，消息类型String

\*/

#include <sstream>

#include "ros/ros.h"

#include "std\_msgs/String.h"

int main(int argc, char \*\*argv)

{

// ROS节点初始化

ros::init(argc, argv, "talker");

//talker是定义整个节点运行起来的名称

// 创建节点句柄

ros::NodeHandle n;

// 创建一个Publisher，发布名为chatter的topic，消息类型为std\_msgs::String

ros::Publisher chatter\_pub = n.advertise<std\_msgs::String>("chatter", 1000);

//1000是指发布者的队列长度，当发布的速度过快时，会把数据存放在这个队列中

//如果发布过快，会把一开始存入的数据覆盖，造成断帧的现象

// 设置循环的频率

ros::Rate loop\_rate(10);

int count = 0;

while (ros::ok())

{

// 初始化std\_msgs::String类型的消息

std\_msgs::String msg;

std::stringstream ss;

ss << "hello world " << count;

msg.data = ss.str();

// 发布消息

ROS\_INFO("%s", msg.data.c\_str());

chatter\_pub.publish(msg);

// 循环等待回调函数

ros::spinOnce();

//无论实现发布者还是订阅者，都要把这个函数放进来

// 按照循环频率延时

loop\_rate.sleep();

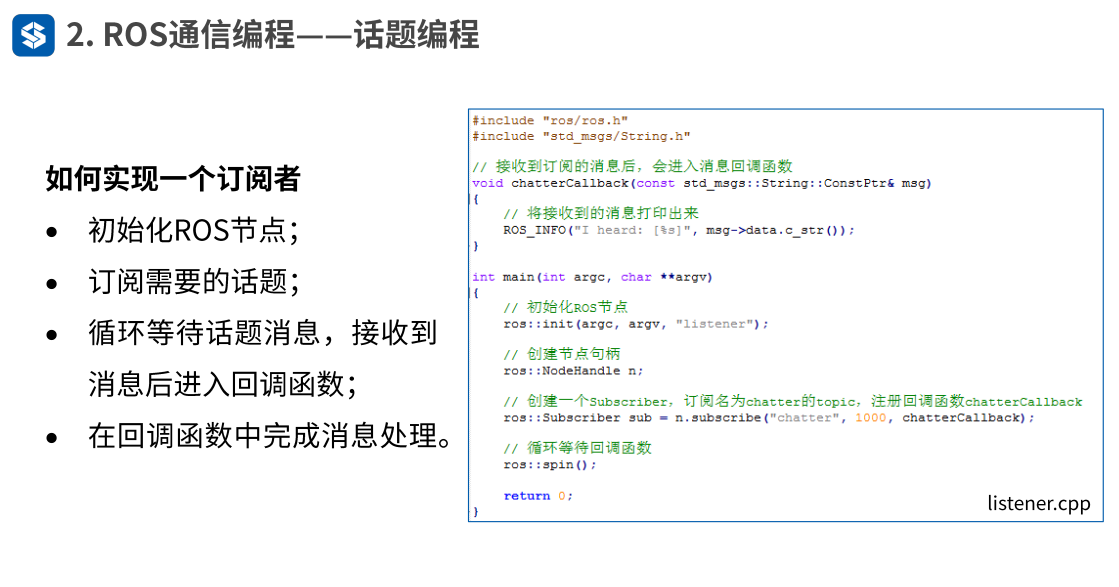
++count;

}

return 0;

}

同样，在src目录下创建listener.cpp文件



/\*\*

\* 该例程将订阅chatter话题，消息类型String

\*/

#include "ros/ros.h"

#include "std\_msgs/String.h"

// 接收到订阅的消息后，会进入消息回调函数

void chatterCallback(const std\_msgs::String::ConstPtr& msg)

{

// 将接收到的消息打印出来

ROS\_INFO("I heard: [%s]", msg->data.c\_str());

}

int main(int argc, char \*\*argv)

{

// 初始化ROS节点

ros::init(argc, argv, "listener");

// 创建节点句柄

ros::NodeHandle n;

// 创建一个Subscriber，订阅名为chatter的topic，注册回调函数chatterCallback

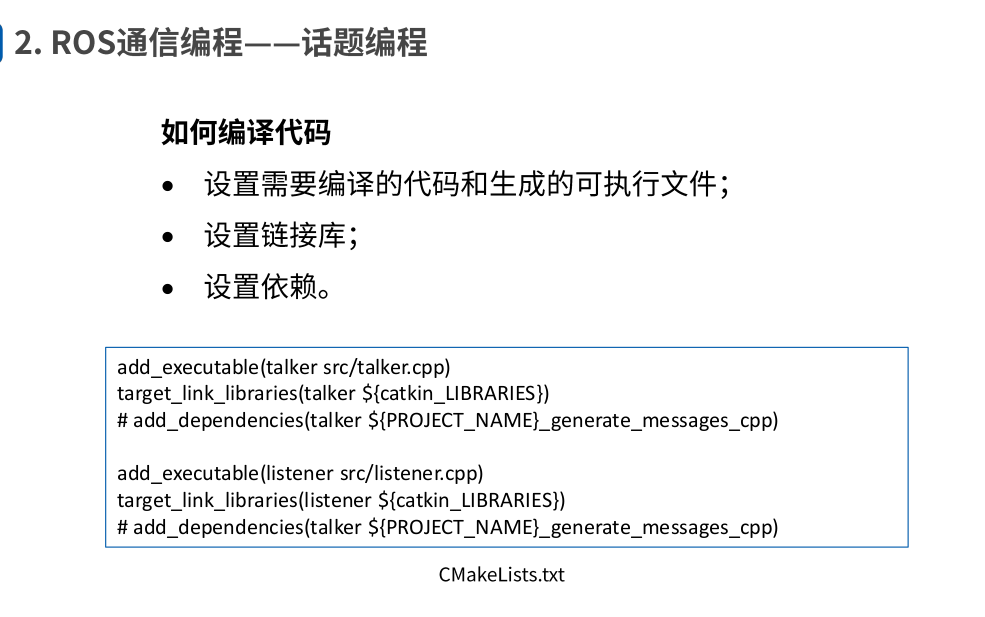
ros::Subscriber sub = n.subscribe("chatter", 1000, chatterCallback);

// 循环等待回调函数

ros::spin();

return 0;

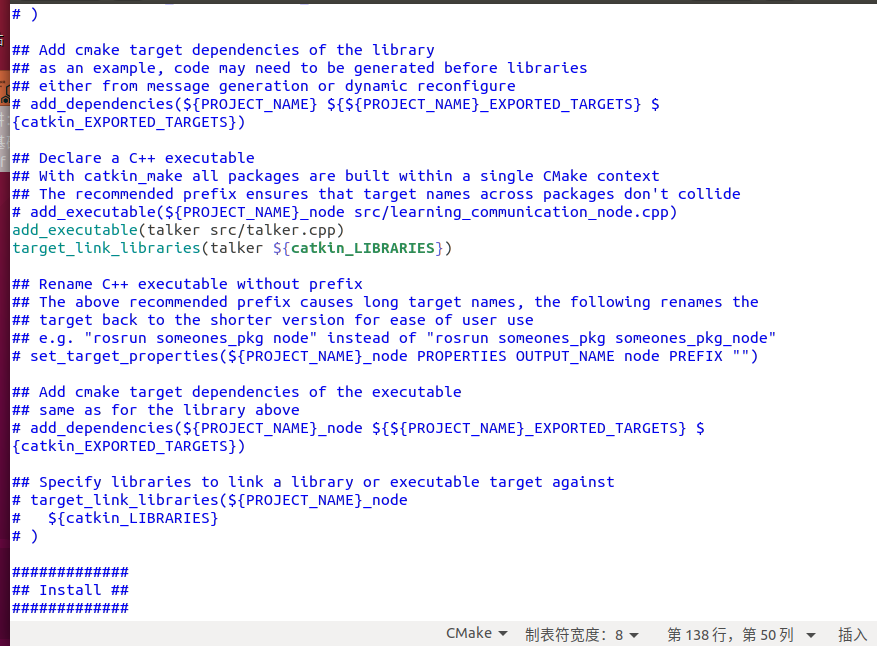
}



在CMakeList.txt文件里面执行下述操作：

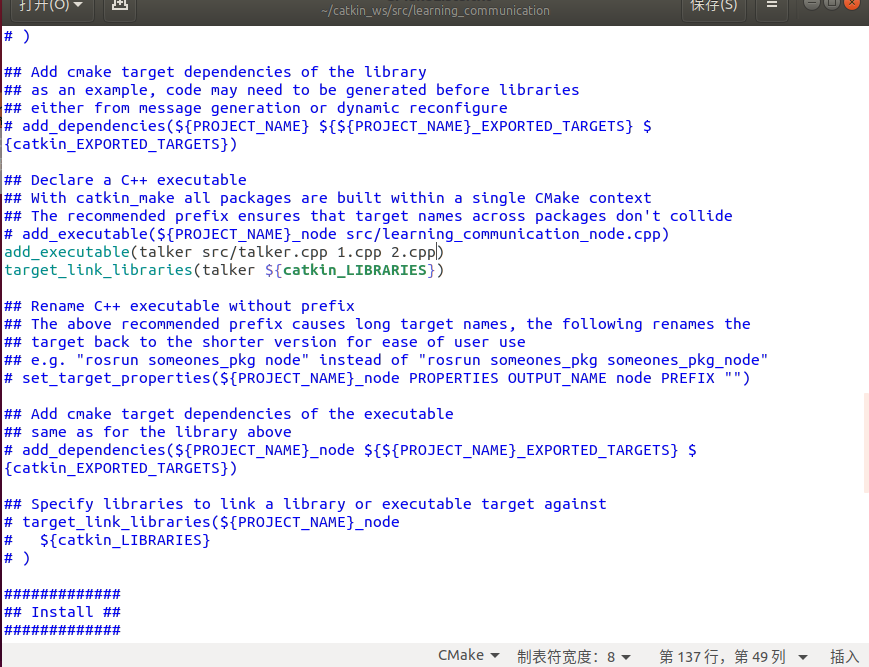
add\_executable(talker src/talker.cpp)

target\_link\_libraries(talker ${catkin\_LIBRARIES})

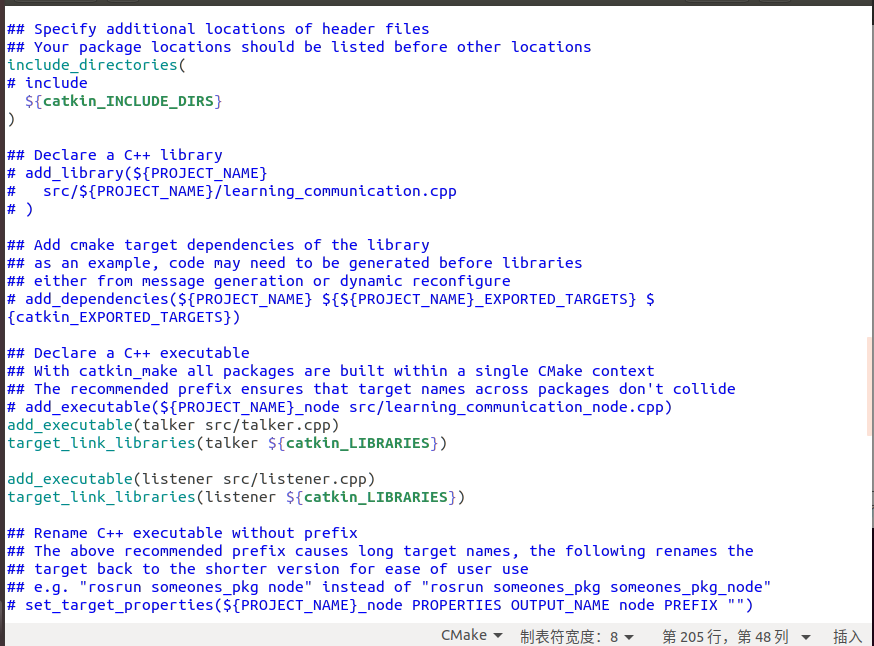


如有更多的可执行文件需要加入：

例如1.cpp，2.cpp



添加完talker.cpp，在添加listener.cpp



全部代码如下：

cmake\_minimum\_required(VERSION 3.0.2)

project(learning\_communication)

## Compile as C++11, supported in ROS Kinetic and newer

# add\_compile\_options(-std=c++11)

## Find catkin macros and libraries

## if COMPONENTS list like find\_package(catkin REQUIRED COMPONENTS xyz)

## is used, also find other catkin packages

find\_package(catkin REQUIRED COMPONENTS

roscpp

rospy

std\_msgs

)

## System dependencies are found with CMake's conventions

# find\_package(Boost REQUIRED COMPONENTS system)

## Uncomment this if the package has a setup.py. This macro ensures

## modules and global scripts declared therein get installed

## See http://ros.org/doc/api/catkin/html/user\_guide/setup\_dot\_py.html

# catkin\_python\_setup()

################################################

## Declare ROS messages, services and actions ##

################################################

## To declare and build messages, services or actions from within this

## package, follow these steps:

## \* Let MSG\_DEP\_SET be the set of packages whose message types you use in

## your messages/services/actions (e.g. std\_msgs, actionlib\_msgs, ...).

## \* In the file package.xml:

## \* add a build\_depend tag for "message\_generation"

## \* add a build\_depend and a exec\_depend tag for each package in MSG\_DEP\_SET

## \* If MSG\_DEP\_SET isn't empty the following dependency has been pulled in

## but can be declared for certainty nonetheless:

## \* add a exec\_depend tag for "message\_runtime"

## \* In this file (CMakeLists.txt):

## \* add "message\_generation" and every package in MSG\_DEP\_SET to

## find\_package(catkin REQUIRED COMPONENTS ...)

## \* add "message\_runtime" and every package in MSG\_DEP\_SET to

## catkin\_package(CATKIN\_DEPENDS ...)

## \* uncomment the add\_\*\_files sections below as needed

## and list every .msg/.srv/.action file to be processed

## \* uncomment the generate\_messages entry below

## \* add every package in MSG\_DEP\_SET to generate\_messages(DEPENDENCIES ...)

## Generate messages in the 'msg' folder

# add\_message\_files(

# FILES

# Message1.msg

# Message2.msg

# )

## Generate services in the 'srv' folder

# add\_service\_files(

# FILES

# Service1.srv

# Service2.srv

# )

## Generate actions in the 'action' folder

# add\_action\_files(

# FILES

# Action1.action

# Action2.action

# )

## Generate added messages and services with any dependencies listed here

# generate\_messages(

# DEPENDENCIES

# std\_msgs

# )

################################################

## Declare ROS dynamic reconfigure parameters ##

################################################

## To declare and build dynamic reconfigure parameters within this

## package, follow these steps:

## \* In the file package.xml:

## \* add a build\_depend and a exec\_depend tag for "dynamic\_reconfigure"

## \* In this file (CMakeLists.txt):

## \* add "dynamic\_reconfigure" to

## find\_package(catkin REQUIRED COMPONENTS ...)

## \* uncomment the "generate\_dynamic\_reconfigure\_options" section below

## and list every .cfg file to be processed

## Generate dynamic reconfigure parameters in the 'cfg' folder

# generate\_dynamic\_reconfigure\_options(

# cfg/DynReconf1.cfg

# cfg/DynReconf2.cfg

# )

###################################

## catkin specific configuration ##

###################################

## 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\_package(

# INCLUDE\_DIRS include

# LIBRARIES learning\_communication

# CATKIN\_DEPENDS roscpp rospy std\_msgs

# DEPENDS system\_lib

)

###########

## Build ##

###########

## Specify additional locations of header files

## Your package locations should be listed before other locations

include\_directories(

# include

${catkin\_INCLUDE\_DIRS}

)

## Declare a C++ library

# add\_library(${PROJECT\_NAME}

# src/${PROJECT\_NAME}/learning\_communication.cpp

# )

## Add cmake target dependencies of the library

## as an example, code may need to be generated before libraries

## either from message generation or dynamic reconfigure

# add\_dependencies(${PROJECT\_NAME} ${${PROJECT\_NAME}\_EXPORTED\_TARGETS} ${catkin\_EXPORTED\_TARGETS})

## Declare a C++ executable

## With catkin\_make all packages are built within a single CMake context

## The recommended prefix ensures that target names across packages don't collide

# add\_executable(${PROJECT\_NAME}\_node src/learning\_communication\_node.cpp)

add\_executable(talker src/talker.cpp)

target\_link\_libraries(talker ${catkin\_LIBRARIES})

add\_executable(listener src/listener.cpp)

target\_link\_libraries(listener ${catkin\_LIBRARIES})

## Rename C++ executable without prefix

## The above recommended prefix causes long target names, the following renames the

## target back to the shorter version for ease of user use

## e.g. "rosrun someones\_pkg node" instead of "rosrun someones\_pkg someones\_pkg\_node"

# set\_target\_properties(${PROJECT\_NAME}\_node PROPERTIES OUTPUT\_NAME node PREFIX "")

## Add cmake target dependencies of the executable

## same as for the library above

# add\_dependencies(${PROJECT\_NAME}\_node ${${PROJECT\_NAME}\_EXPORTED\_TARGETS} ${catkin\_EXPORTED\_TARGETS})

## Specify libraries to link a library or executable target against

# target\_link\_libraries(${PROJECT\_NAME}\_node

# ${catkin\_LIBRARIES}

# )

#############

## Install ##

#############

# all install targets should use catkin DESTINATION variables

# See http://ros.org/doc/api/catkin/html/adv\_user\_guide/variables.html

## Mark executable scripts (Python etc.) for installation

## in contrast to setup.py, you can choose the destination

# catkin\_install\_python(PROGRAMS

# scripts/my\_python\_script

# DESTINATION ${CATKIN\_PACKAGE\_BIN\_DESTINATION}

# )

## Mark executables for installation

## See http://docs.ros.org/melodic/api/catkin/html/howto/format1/building\_executables.html

# install(TARGETS ${PROJECT\_NAME}\_node

# RUNTIME DESTINATION ${CATKIN\_PACKAGE\_BIN\_DESTINATION}

# )

## Mark libraries for installation

## See http://docs.ros.org/melodic/api/catkin/html/howto/format1/building\_libraries.html

# install(TARGETS ${PROJECT\_NAME}

# ARCHIVE DESTINATION ${CATKIN\_PACKAGE\_LIB\_DESTINATION}

# LIBRARY DESTINATION ${CATKIN\_PACKAGE\_LIB\_DESTINATION}

# RUNTIME DESTINATION ${CATKIN\_GLOBAL\_BIN\_DESTINATION}

# )

## Mark cpp header files for installation

# install(DIRECTORY include/${PROJECT\_NAME}/

# DESTINATION ${CATKIN\_PACKAGE\_INCLUDE\_DESTINATION}

# FILES\_MATCHING PATTERN "\*.h"

# PATTERN ".svn" EXCLUDE

# )

## Mark other files for installation (e.g. launch and bag files, etc.)

# install(FILES

# # myfile1

# # myfile2

# DESTINATION ${CATKIN\_PACKAGE\_SHARE\_DESTINATION}

# )

#############

## Testing ##

#############

## Add gtest based cpp test target and link libraries

# catkin\_add\_gtest(${PROJECT\_NAME}-test test/test\_learning\_communication.cpp)

# if(TARGET ${PROJECT\_NAME}-test)

# target\_link\_libraries(${PROJECT\_NAME}-test ${PROJECT\_NAME})

# endif()

## Add folders to be run by python nosetests

# catkin\_add\_nosetests(test)

回到catkin\_ws目录下，执行编译，运行catkin\_make