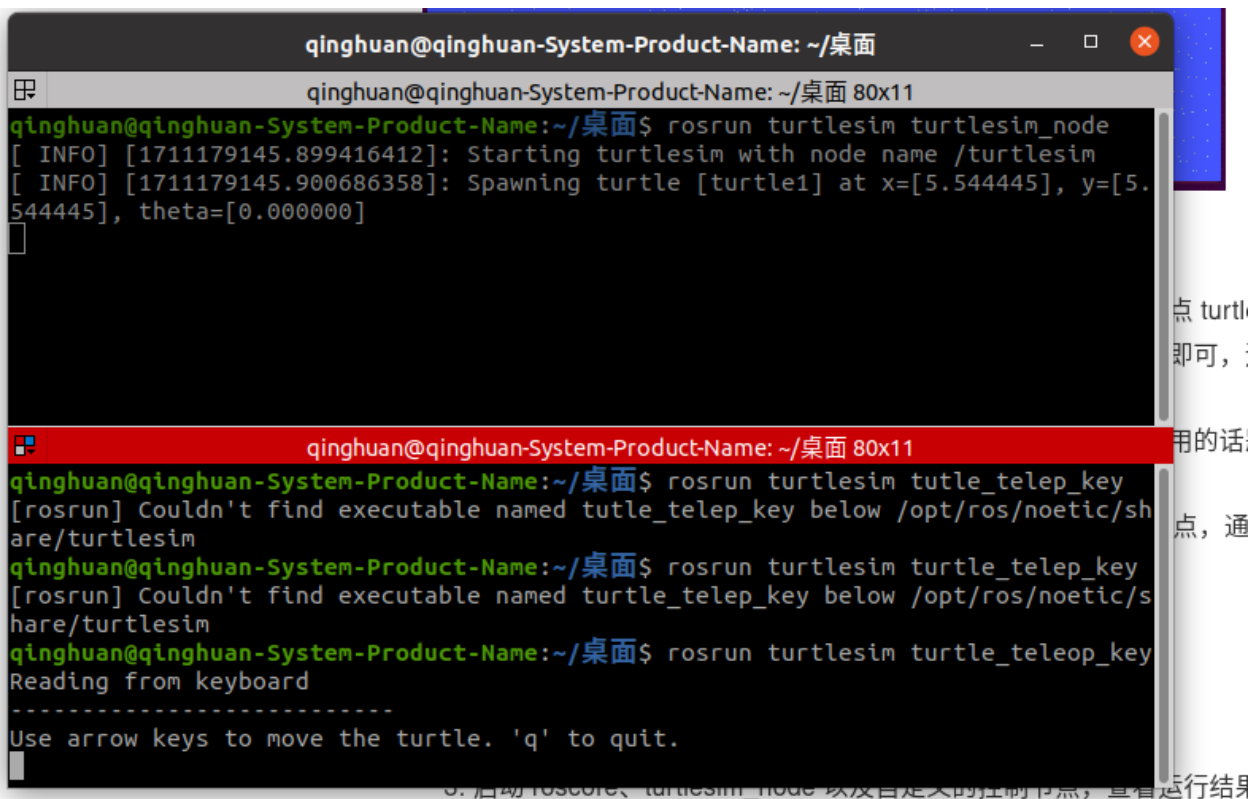


问题分析

1、乌龟运动的实现，控制乌龟运动让他圆周运动

让小乌龟圆周运动

运行乌龟的节点



The image shows a terminal window with the following content:

```
qinghuan@qinghuan-System-Product-Name: ~/桌面
qinghuan@qinghuan-System-Product-Name: ~/桌面 80x11
qinghuan@qinghuan-System-Product-Name:~/桌面$ rosrun turtlesim turtlesim_node
[ INFO] [1711179145.899416412]: Starting turtlesim with node name /turtlesim
[ INFO] [1711179145.900686358]: Spawning turtle [turtle1] at x=[5.544445], y=[5.544445], theta=[0.000000]
qinghuan@qinghuan-System-Product-Name:~/桌面$ rosrun turtlesim turtile_teleop_key
[roslaunch] Couldn't find executable named turtile_teleop_key below /opt/ros/noetic/share/turtlesim
qinghuan@qinghuan-System-Product-Name:~/桌面$ rosrun turtlesim turtle_teleop_key
[roslaunch] Couldn't find executable named turtle_teleop_key below /opt/ros/noetic/share/turtlesim
qinghuan@qinghuan-System-Product-Name:~/桌面$ rosrun turtlesim turtle_teleop_key
Reading from keyboard
-----
Use arrow keys to move the turtle. 'q' to quit.
```

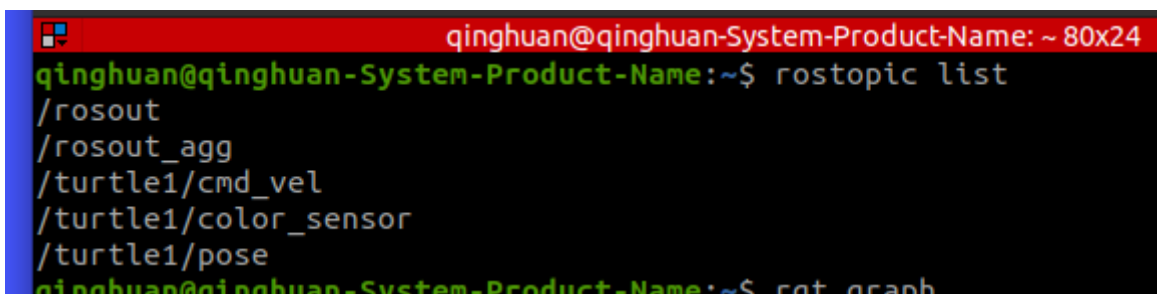
点 turtle
即可，
用的话
点，通
运行结果

在一个终端 roscore 启动 ros

之后 rosrun turtlesim turtlesim_node

roslaunch turtlesim turtle_teleop_key

显示话题



The image shows a terminal window with the following content:

```
qinghuan@qinghuan-System-Product-Name: ~ 80x24
qinghuan@qinghuan-System-Product-Name:~$ rostopic list
/rosout
/rosout_agg
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
qinghuan@qinghuan-System-Product-Name:~$ rqt_graph
```

显示计算图信息

显示 话题的相关使用方法

```
turtle1/pose
qinghuan@qinghuan-System-Product-Name:~$ rostopic
qinghuan@qinghuan-System-Product-Name:~$ rostopic
rostopic is a command-line tool for printing
Commands:
  rostopic bw      display bandwidth use
  rostopic delay   display delay of topic
  rostopic echo    print messages to screen
  rostopic find    find topics by type
  rostopic hz      display publishing rate of topic
  rostopic info    print information about active topic
  rostopic list    list active topics
  rostopic pub     publish data to topic
  rostopic type    print topic or field type

Type rostopic <command> -h for more detailed usage, e.g. 'rostopic echo -h'
```

```
include_directories(
  include
  ${catkin_INCLUDE_DIRS}
)

## Declare a C++ library
add_library(call_f
  include/${PROJECT_NAME}/call_f.h
  src/call_f.cpp
)

## Add cmake target dependencies of the library
```

显示消息类型

```
ERROR: Unknown topic /turtle/cmd_vel
qinghuan@qinghuan-System-Product-Name:~$ rostopic info /turtle1/cmd_vel
Type: geometry_msgs/Twist

Publishers:
  * /teleop_turtle (http://qinghuan-System-Product-Name:44151/)
```

传递的消息类型 是个结构体，我感觉是分别定义的

```
qinghuan@qinghuan-System-Product-Name:~$ rosmmsg show geometry_msgs/Twist
geometry_msgs/Vector3 linear
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 angular
  float64 x
  float64 y
  float64 z
```

2.小海龟位置：

在根目录下，查找路径：`/opt/ros/melodic/share/turtlesim/images`. 就可找到所有的小海龟：

```
/opt/ros/melodic/share/turtlesim/images
```

在这个 ros 的文件夹下赵 `geometry_msg` 文件夹
之后看到 `vector3.msg` 是 3 个数自
之后 `twist` 是用的

```
Twist.msg × Vector3.msg ×  
1 # This represents a vector in free space.  
2 # It is only meant to represent a direction. Therefore, it does not  
3 # make sense to apply a translation to it (e.g., when applying a  
4 # generic rigid transformation to a Vector3, tf2 will only apply the  
5 # rotation). If you want your data to be translatable too, use the  
6 # geometry_msgs/Point message instead.  
7  
8 float64 x  
9 float64 y  
10 float64 z
```

msg 嵌套

```
Twist.msg × Vector3.msg ×  
1 # This expresses velocity in free space broken into its linear and angular parts.  
2 Vector3 linear  
3 Vector3 angular
```

这个是我们发布方的实现

```
1  #include "ros/ros.h"
2  #include "geometry_msgs/Twist.h"
3  /*
4   发布话题消息
5   geometry
6   */
7  int main(int argc, char* argv[])
8  {
9
10
11     ros::init(argc, argv, "mycontroller");
12     ros::NodeHandle nh;
13     ros::Publisher pub = nh.advertise<geometry_msgs::Twist>("/turtle1/cmd_vel", 10);
14     ros::Rate rate(10);
15     geometry_msgs::Twist twist;
16     twist.linear.x = 1.0;
17     twist.linear.y = 0.0;
18     twist.linear.z = 0.0;
19     twist.angular.x = 0.0;
20     twist.angular.y = 0.0;
21
22     twist.angular.z = 0.5;
23     while (ros::ok())
24     {
25         pub.publish(twist);
26         rate.sleep();
27         ros::spinOnce();
28     }
29
30     return 0;
31 }
```

额外添加功能包 在最开始没有添加的话 通过修改 CMakeLists 和 package.xml 文件进行修应该

添加功能包

```
1  use catkin_package to specify packages you need only for buildtool
2
3  <!-- <doc_depend>doxygen</doc_depend> -->
4  <buildtool_depend>catkin</buildtool_depend>
5  <build_depend>geometry_msgs</build_depend>
6  <build_export_depend>roscpp</build_export_depend>
7  <build_export_depend>rospy</build_export_depend>
8  <build_export_depend>std_msgs</build_export_depend>
9  <build_export_depend>turtlesim</build_export_depend>
10 <build_export_depend>geometry_msgs</build_export_depend>
11 <build_export_depend>roscpp</build_export_depend>
12 <build_export_depend>rospy</build_export_depend>
13 <build_export_depend>std_msgs</build_export_depend>
14 <exec_depend>geometry_msgs</exec_depend>
15 <exec_depend>roscpp</exec_depend>
16 <exec_depend>rospy</exec_depend>
17 <exec_depend>std_msgs</exec_depend>
18 <exec_depend>turtlesim</exec_depend>
```

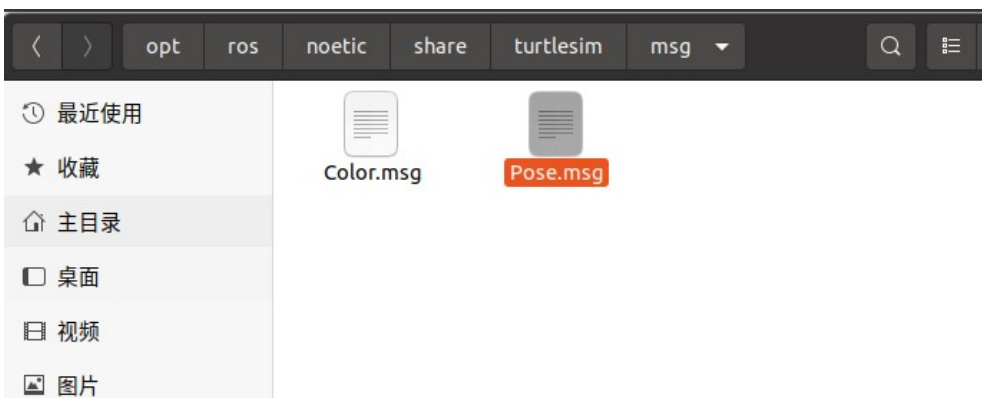
```
src > plumbing_test > M CMakeLists.txt
1 cmake_minimum_required(VERSION 3.0.2)
2 project(plumbing_test)
3
4 ## Compile as C++11, supported in ROS Kinetic and newer
5 # add_compile_options(-std=c++11)
6
7 ## Find catkin macros and libraries
8 ## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
9 ## is used, also find other catkin packages
10 find_package(catkin REQUIRED COMPONENTS
11   geometry_msgs
12   roscpp
13   rospy
14   std_msgs
15   turtlesim
16 )
17
18 ## Custom dependencies are found with CMake's conventions
```

我们向 turtlesim 的节点发送数据

订阅的消息类型 Pose.msg 的消息类型

```
1 float32 x
2 float32 y
3 float32 theta
4
5 float32 linear_velocity
6 float32 angular_velocity
```

位置 Pose.msg 的位置



- 1、我们使用类的成员函数 作为 subscribe 的 callback 函数
- 2、这个类是分文件实现的 在自定义的 namespace 下 的类文件
- 3、如何配置相应的 cmake 文件

具体实现流程

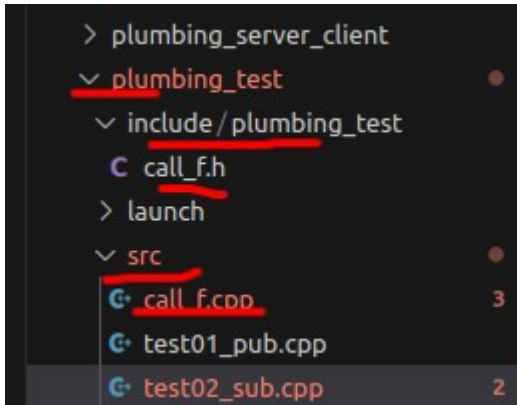
- 1、分文件实现 class

实现.h 头文件和 .cpp 成员函数文件

1.注意头文件是在功能包的 include 文件夹下的功能包下 定义的头文件

2.实现文件是放在 src 里了

上述两条影响在 cmake 的 配置



头文件实现

- 1、namespace 是 callback_f
 - 2、注意头文件 把我们传递的消息类型 引入到这里
 - 3、定义 class cc_f 声明一个 public 的成员函数
 - 4、注意传参 是一个常指针 const 我们不修改
- 格式 功能包::msg 文件::ConstPtr & 名称 传递进来一个实参

```
1  #ifndef call_f
2  #define call_f
3
4  #include "turtlesim/Pose.h"
5  namespace callback_f{
6  class cc_f
7  {
8  public:
9  void run(const turtlesim::Pose::ConstPtr& pose);
10 };
11
12 }
13
14 #endif
```


cpp 文件实现

- 1、首先要把头文件包含
- 2、之后要把 ros 文件包含在内，我感觉这个是可以放在头文件的，
- 3、注意传递的参数类型
- 4、指针调用 数据 不能用.点 要用->的方法 数据的名字在上述的 Pose.msg 截图展示了

```
src > plumbing_test > src > call_f.cpp > {} callback_f
1 #include "plumbing_test/call_f.h"
2 #include "ros/ros.h"
3 namespace callback_f
4 {
5
6 void cc_f::run(const turtlesim::Pose::ConstPtr& pose)
7 {
8     ROS_INFO("位置信息包含 %2.f,%2.f, 朝向 %2.f,速度 %2.f,%2.f",pose->x,pose->y,pose->theta,pose->linear_velocity,pose->angular_velocity);
9 }
10 }
```

在主函数中使用

- 1、包含 class 的头文件
- 2、在这里正常包含头文件，和 msg 文件是 turtlesim 功能包的 Pose.msg 这个功能包生成了这个的头文件 Pose.h 在相应的 devel 文件下
- 3、我们如何使用 首先 是实例化类 namespace::类名 实例化对象的名字 obj
- 4、作为 callback 是传递出类的成员函数的 函数指针，之后传递出 对象的指针
&call_back_f::cc_f::run &这是个指针 call_back_f namespace cc_f 是类名 run 成员函数的名字
&obj 传递一个实例化对象的指针

```
1 #include "plumbing_test/call_f.h"
2 #include "ros/ros.h"
3 namespace callback_f
4 {
5
6 void cc_f::run(const turtlesim::Pose::ConstPtr& pose)
7 {
8     ROS_INFO("class 位置信息包含 %2.f,%2.f, 朝向 %2.f,速度 %2.f,%2.f",pose->x,pose->y,pose->theta,pose->linear_velocity,pose->angular_velocity);
9 }
10 }
```

配置 cmake 的参数

- 1、配置 class 文件

分文件实现的 class 文件的头是在 include/功能包名【在 ros 是叫做 \${PROJECT_NAME} 这里是 plumbing_test】

定一个别名 给这两个文件 是通过 add_library 实现的 call_f 是个别名 他指向 头文件和实现成员函数的 cpp 文件

```
include_directories(
  include
  ${catkin_INCLUDE_DIRS}
)

## Declare a C++ library
add_library(call_f
  include/${PROJECT_NAME}/call_f.h
  src/call_f.cpp
)
```

在这里的 add_executable 给执行的 cpp 起一个别名

test02_sub 是我们实现订阅的文件

注意 add_dependencies 是添加依赖 是保证我们这个 cpp 文件是在 msg 编译后 再编译的

target_link_libraries call_f 与当前的工作空间

test02_pub 也要调用这个 call_f (这个 call_f 代表那两个类的头文件和成员函数的实现文件)

```
add_executable(test01_pub src/test01_pub.cpp)
add_executable(test02_sub src/test02_sub.cpp)
## 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. "roslaunch someones_pkg node" instead of "roslaunch 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(test01_pub ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})
add_dependencies(test02_sub ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})
## Specify libraries to link a library or executable target against
target_link_libraries(call_f
  ${catkin_LIBRARIES})
target_link_libraries(test01_pub
  ${catkin_LIBRARIES})
target_link_libraries(test02_sub
  call_f
  ${catkin_LIBRARIES})
#####
```

调用的结果

先用 launch 文件启动多个 node 节点 这个是 turtlesim 的节点

```
[ INFO] [1711264253.686992348]: Spawning turtle [turtle1] at x=[5.544445], y=[5.544445], theta=[
[1]+ 已停止                  roslaunch plumbing_test start_launch.launch
qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$
```

之后 roslaunch plumbing_test test02_sub

```
[ INFO] [1711264282.491697278]: class 位置信息包含 8, 9, 朝向 1,速度 0, 0
[ INFO] [1711264282.507795408]: class 位置信息包含 8, 9, 朝向 1,速度 0, 0
[ INFO] [1711264282.523863931]: class 位置信息包含 8, 9, 朝向 1,速度 0, 0
[ INFO] [1711264282.539930632]: class 位置信息包含 8, 9, 朝向 1,速度 0, 0
^Z
[2]+ 已停止                  roslaunch plumbing_test test02_sub
qinghuan@qinghuan-System-Product-Name:~/env_cv/demo04_ws$
```

这里的

start_launch 文件的实现方法

```
src > plumbing_test > launch > start_launch.launch
1  <!-- 启动乌龟gui和键盘控制节点 -->
2  <launch>
3  <!-- 启动乌龟gui -->
4  <!-- 报名      定义的别名就算add_execute第一个参数 之后name最定义的 output输出到哪里-->
5  <node pkg="turtlesim" type="turtlesim_node" name="turtle1" output="screen"/>
6  <!-- 启动键盘控制节点 -->
7  <node pkg="turtlesim" type="turtle_teleop_key" name="key" output="screen"/>
8
9
10 </launch>
11
```