

3、 ROS2 Topic communications

Topic communication is one of the most frequently used communication methods in ROS2, topic communication is based on the publish-subscribe model, there are publishers publishing data on a specified topic, subscribers as long as they subscribe to the data of the topic, they can receive data. Next, let's explain how to use the Python language to achieve topic communication between nodes.

1、 Create a new workspace

terminal input:

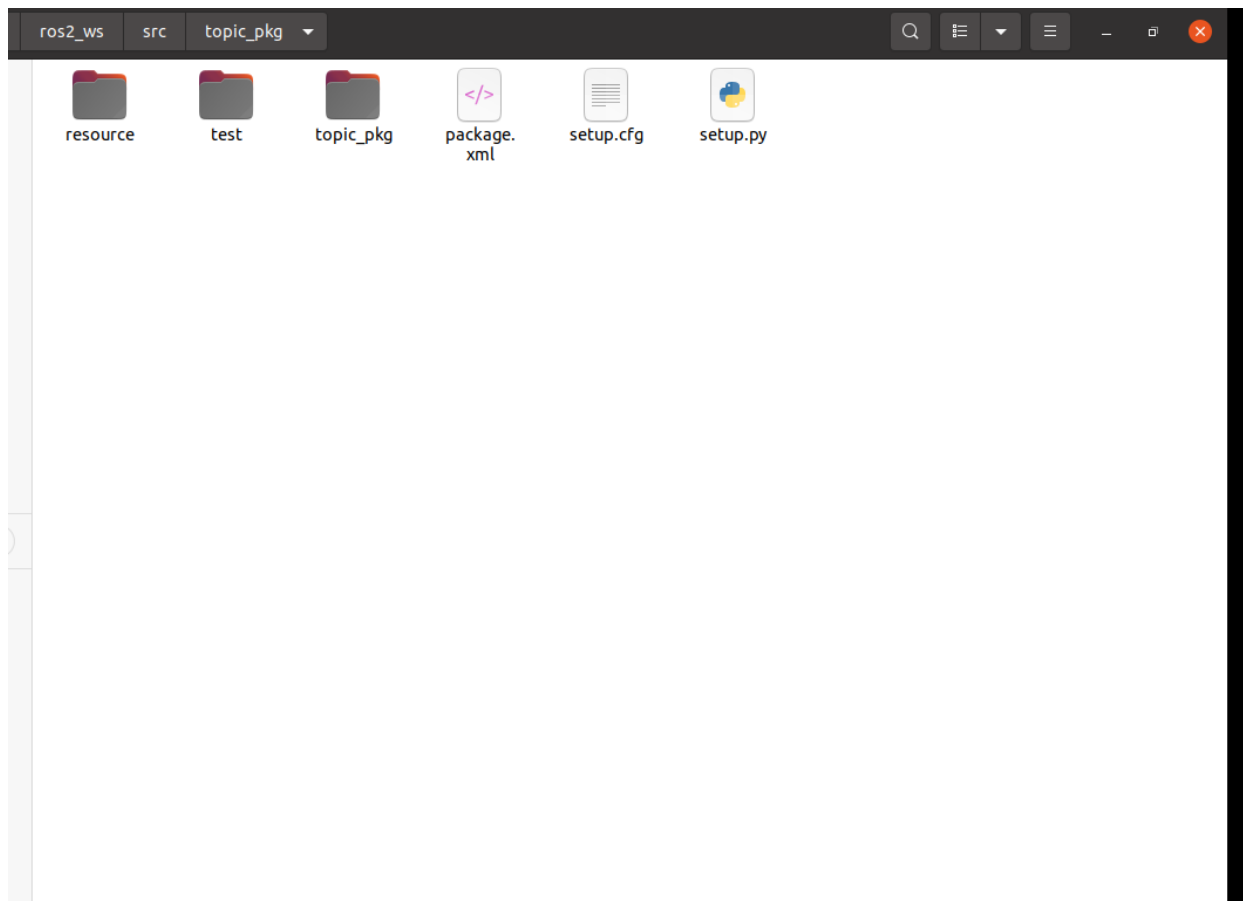
```
mkdir -p ros2_ws/src
cd ros2_ws/src
```

ros2_ws is the name of the workspace, and src is the directory where the feature packs are stored.

2、 Create a new feature pack

terminal input:

```
cd ~/ros2_ws/src
ros2 pkg create --build-type ament_python topic_pkg
```



Switch to topic_pkg folder, there are the following files and folders, the python program we wrote is placed under the topic_pkg folder under this directory, that is:

```
~/ros2_ws/src/topic_pkg/topic_pkg
```

3、 Write the publisher python file

3.1、 Program source code

Create a new file, named publisher_demo.py:

```
cd ~/ros2_ws/src/topic_pkg/topic_pkg
gedit publisher_demo.py
```

Copy the following sections into the file:

```
import rclpy
from rclpy.node import Node
from std_msgs.msg import String
class Topic_Pub(Node):
    def __init__(self,name):
        super().__init__(name)
        self.pub = self.create_publisher(String,"/topic_demo",1)
        self.timer = self.create_timer(1,self.pub_msg)

    def pub_msg(self):
        msg = String()
        msg.data = "Hi,I send a message."
        self.pub.publish(msg)

def main():
    rclpy.init()
    pub_demo = Topic_Pub("publisher_node")
    rclpy.spin(pub_demo)
```

This is achieved with the idea of modularity, which is conducive to porting and modifying the code. First define a class, the class name is Topic_Pub, there are two parts in it, one is init and the other is pub_msg. init is the initialization of the class itself, pub_msg the methods of the class, that is, as long as we create the objects of this class, we can use the variables and methods inside.

```
#Import the RCpy library
import rclpy
from rclpy.node import Node
#Import a String message
from std_msgs.msg import String
#Create a Topic_Pub node subclass that inherits from the Node base class and pass in
a parameter name
def __init__(self,name):
    super().__init__(name)
    #Create a publisher with create_publisher function and pass in the following
parameters:
    #Topic data type, thread name, queue length to save messages
```

```

        self.pub = self.create_publisher(String, "/topic_demo", 1)
        #Create a timer and enter the interrupt handler at intervals of 1s, passing
in the following parameters:
        #The interval between execution of the interrupt function, interrupt the
handler
        self.timer = self.create_timer(1, self.pub_msg)
#Defines interrupt handlers
def pub_msg(self):
    msg = String() #Create a variable of type String msg
    msg.data = "Hi,I send a message." #Assign a value to the data in MSG
    self.pub.publish(msg) #Publish topic data

#Main function
def main():
    rclpy.init() #initialize
    pub_demo = Topic_Pub("publisher_node")
    #To create a Topic_Pub class object, the parameter passed in is the name of the
node
    rclpy.spin(pub_demo)
    #Execute the rclpy.spin function, pass in a parameter, the parameter is the
Topic_Pub class object just created

```

3.2. Modify the setup.py file

terminal input:

```

cd ~/ros2_ws/src/topic_pkg
gedit setup.py

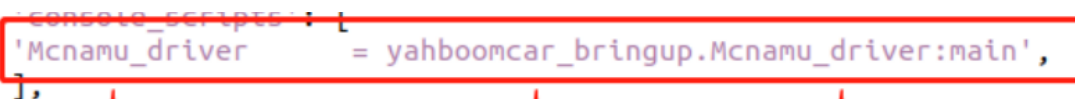
```

Locate the location as shown in the image below:



```
1 from setuptools import setup
2
3 package_name = 'topic_pkg'
4
5 setup(
6     name=package_name,
7     version='0.0.0',
8     packages=[package_name],
9     data_files=[
10         ('share/ament_index/resource_index/packages',
11          ['resource/' + package_name]),
12         ('share/' + package_name, ['package.xml']),
13     ],
14     install_requires=['setuptools'],
15     zip_safe=True,
16     maintainer='yahboom',
17     maintainer_email='yahboom@todo.todo',
18     description='TODO: Package description',
19     license='TODO: License declaration',
20     tests_require=['pytest'],
21     entry_points={
22         'console_scripts': [
23             'publisher_demo = topic_pkg.publisher_demo:main'
24         ],
25     },
26 )
```

Add the following to 'console_scripts': [] in the format,



```
'console_scripts': [
    'Mcnamu_driver = yahboomcar_bringup.Mcnamu_driver:main',
],
```

The executable file name

Feature pack name

The name of the Python file written

The name of the executable file is our custom, that is, which program we need to execute when ros2 run, the function package name is which function package your python file is located in, followed by the name of the python file, and the last main is the program execution entry. As long as it is a program written in python format, compile and generate executable files, you need to add it here in the above format.

3.3、Compile the workspace

```
cd ~/ros2_ws
colcon build
```

Compile successfully, enter the following command to add the workspace to the system environment variable,

```
echo "source ~/ros2_ws/install/setup.bash" >> ~/.bashrc
```

Then, reopen the terminal or refresh the environment variable to take effect.

```
source ~/.bashrc
```

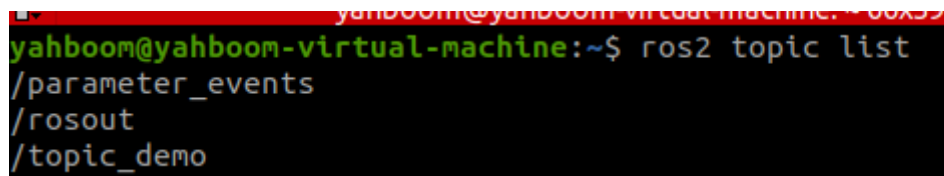
3.4、Run the program

terminal input,

```
ros2 run topic_pkg publisher_demo
```

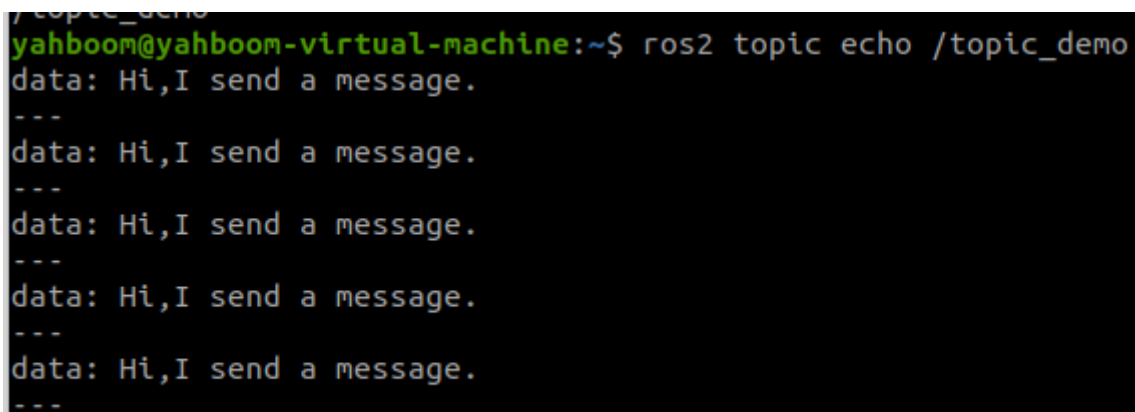
After the program runs successfully, there is no printing anything, we can view the data through the ros2 topic tool, first check whether this topic is published, terminal input,

```
ros2 topic list
```

A terminal window screenshot showing the command 'ros2 topic list' being executed. The output lists three topics: '/parameter_events', '/rosout', and '/topic_demo'. The prompt is 'yahboom@yahboom-virtual-machine:~\$'.

This topic_demo is the topic data defined in the program, next, we use ros2 topic echo to print this data, terminal input,

```
ros2 topic echo /topic_demo
```

A terminal window screenshot showing the command 'ros2 topic echo /topic_demo' being executed. The output shows a repeating pattern of 'data: Hi,I send a message.' followed by three dashes '---' on each line. The prompt is 'yahboom@yahboom-virtual-machine:~\$'.

As you can see, the terminal prints "Hi, I send a message." msg.data = "Hi,I send a message" in our code." unanimous.

4、 Write subscriber python files

4.1、 Program source code

Create a new file, named subscriber_demo.py,

```
cd ~/ros2_ws/src/topic_pkg/topic_pkg
gedit subscriber_demo.py
```

Copy the following sections into the file,

```
#Import the relevant libraries
import rclpy
from rclpy.node import Node
from std_msgs.msg import String

class Topic_Sub(Node):
    def __init__(self,name):
        super().__init__(name)
        #The create_subscription used to create the subscriber is: topic data type,
        #topic name, callback function name, queue length
        self.sub =
self.create_subscription(String,"/topic_demo",self.sub_callback,1)
        #Callback function executor: prints the received information
    def sub_callback(self,msg):
        print(msg.data)
def main():
    rclpy.init() #ROS2 Python interface initialization
    sub_demo = Topic_Sub("subscriber_node") #Create an object and initialize it
    rclpy.spin(sub_demo)
```

4.2、 Modify the setup.py file

terminal input,

```
cd ~/ros2_ws/src/topic_pkg
gedit setup.py
```

Locate the location as shown in the image below,

```

1 from setuptools import setup
2
3 package_name = 'topic_pkg'
4
5 setup(
6     name=package_name,
7     version='0.0.0',
8     packages=[package_name],
9     data_files=[
10         ('share/ament_index/resource_index/packages',
11          ['resource/' + package_name]),
12         ('share/' + package_name, ['package.xml']),
13     ],
14     install_requires=['setuptools'],
15     zip_safe=True,
16     maintainer='yahboom',
17     maintainer_email='yahboom@todo.todo',
18     description='TODO: Package description',
19     license='TODO: License declaration',
20     tests_require=['pytest'],
21     entry_points={
22         'console_scripts': [
23             'publisher_demo = topic_pkg.publisher_demo:main',
24             'subscriber_demo = topic_pkg.subscriber_demo:main'
25         ],
26     },
27 )

```

Add the following to 'console_scripts': [] in the format,

```
'subscriber_demo = topic_pkg.subscriber_demo:main'
```

```
'subscriber_demo = topic_pkg.subscriber_demo:main'
```

4.3. Compile the workspace

```
cd ~/ros2_ws
colcon build
```

After the compilation is completed, refresh the environment variables of the workspace.

```
source ~/ros2_ws/install/setup.bash
```

4.4. Run the program

terminal input,

```
#Start the publisher node
ros2 run topic_pkg publisher_demo
#Start the subscriber node
ros2 run topic_pkg subscriber_demo
```



```

import rclpy
from rclpy.node import Node

class ClassName(Node):
    def __init__(self, name):
        super().__init__(name)
        #create subscriber
        self.sub =
self.create_subscription(Msg_Type, Topic_Name, self.CallbackFunction, Msg_Line_Size
)

        #create publisher
        self.pub = self.create_publisher(Msg_Type, Topic_Name, Msg_Line_Size)

        #create timer
        self.timer = self.create_timer(Timer, self.TimerExcuteFunction)

    def CallbackFunction(self):
        ...
    def TimerExcuteFunction(self):
        ...

def main():
    rclpy.init()
    class_object = ClassName("node_name")
    rclpy.spin(class_object)

```

5.2. Modify the contents of setup.py file

To write a node program using Python, you need to modify the setup.py file, refer to the above content, and add and generate your own node program.