2025-9-22(launch)

1.创建执行文件

c++:

```
在pkg03_topic_cpp下创建文件夹launch,launch中创建文件topic_cpp.launch.py
```

python:

```
在pkg03_topic_py下创建文件夹launch,launch中创建文件topic_py.launch.py
```

2.配置编译文件

c++: CMakeLists.txt文件中,在'ament_package()'之前加入以下内容

```
install(DIRECTORY
  launch
  DESTINATION share/${PROJECT_NAME}
)
```

python: 修改setup.bash文件,添加以下部分内容

```
from glob import glob

data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),

        ('share/' + package_name+'/launch', glob('launch/*.launch.py')),
    ],
```

3.编写代码

c++: topic_cpp.launch.py

```
import launch
import launch_ros

def generate_launch_description():
    talker = launch_ros.actions.Node(
        package='pkg03_topic_cpp',
        executable='demo02_talker_cpp',
        output='screen'
    )

    listener = launch_ros.actions.Node(
        package='pkg03_topic_cpp',
        executable='demo02_listener_cpp',
        output='screen'
    )

    launch_description = launch.LaunchDescription([
        talker,
        listener,
    ])

    return launch_description
```

python: topic_py.launch.py

```
import launch
import launch_ros

def generate_launch_description():
    talker = launch_ros.actions.Node(
        package='pkg03_topic_py',
        executable='demo02_talker_py',
        output='screen'
    )

    listener = launch_ros.actions.Node(
        package='pkg03_topic_py',
        executable='demo02_listener_py',
        output='screen'
    )

    launch_description = launch.LaunchDescription([
        talker,
        listener,
    ])

    return launch_description
```

4.编译运行

编译:

```
cd ros2_ws
colcon build
```

5.1 运行:

c++:

```
source install/setup.bash
ros2 launch pkg03_topic_cpp topic_cpp.launch.py
```

python:

```
source install/setup.bash
ros2 launch pkg03_topic_py topic_py.launch.py
```

2025-9-18(service_custom)

1.创建功能包

```
cd ros2_ws/src
```

c++:

```
ros2 pkg create pkg04_service_cpp --build-type ament_cmake --dependencies rclcpp
pkg00_interfaces
```

python:

```
ros2 pkg create pkg04_service_py --build-type ament_python --dependencies rclpy
pkg00_interfaces
```

2.创建执行文件

接口包配置:

在pkg00_interfaces下创建文件夹srv，srv中创建文件AddInts.srv，并在AddInts.srv中填入以下内容:

```
int32 num1
int32 num2
---
int32 sum
```

c++:

```
在pkg04_service_cpp/src下新建demo01_server_cpp.cpp
在pkg04_service_cpp/src下新建demo01_client_cpp.cpp
```

python:

```
在pkg04_service_py/pkg04_service_py下新建demo01_server_py.py
在pkg04_service_py/pkg04_service_py下新建demo01_client_py.py
```

3.配置编译文件

消息接口: CMakeLists.txt:

```
rosidl_generate_interfaces(${PROJECT_NAME}
  "msg/Student.msg"
  "srv/AddInts.srv"
)
```

c++: CMakeLists.txt

```
add_executable(demo01_server_cpp src/demo01_server_cpp.cpp)
add_executable(demo01_client_cpp src/demo01_client_cpp.cpp)

ament_target_dependencies(
  demo01_server_cpp
  "rclcpp"
  "pkg00_interfaces"
)
ament_target_dependencies(
  demo01_client_cpp
  "rclcpp"
  "pkg00_interfaces"
)

install(TARGETS
  demo01_server_cpp
  demo01_client_cpp
  DESTINATION lib/${PROJECT_NAME})
```

python: setup.py

```
entry_points={
        'console_scripts': [
            'demo01_server_py = pkg04_service_py.demo01_server_py:main',
            'demo01_client_py = pkg04_service_py.demo01_client_py:main',
        ],
    },
```

4.编写代码

4.1 c++

demo01_server_cpp.cpp:

```
#include <rclcpp/rclcpp.hpp>
#include <pkg00_interfaces/srv/add_ints.hpp>
using pkg00_interfaces::srv::AddInts;

class Ser:public rclcpp::Node{
public:
    Ser(const std::string &node_name):Node(node_name){
        RCLCPP_INFO(this->get_logger(),"服务端已启动,等待请求...");
        server_ = this->create_service<AddInts>("service_add_nums",

std::bind(&Ser::add_callback,this,std::placeholders::_1,std::placeholders::_2));
    }

    void add_callback(const AddInts::Request::SharedPtr req,const
AddInts::Response::SharedPtr res){
        res->sum = req->num1 + req->num2;
        RCLCPP_INFO(this->get_logger(),"计算过程: %d + %d = %d",req->num1,req-
>num2,res->sum);
    }
private:
    rclcpp::Service<AddInts>::SharedPtr server_;

};

int main(int argc, char* argv[]){
    rclcpp::init(argc,argv);
    auto server = std::make_shared<Ser>("Ser");
    rclcpp::spin(server);
    rclcpp::shutdown();
}
```

demo01_client_cpp.cpp:

```
#include <rclcpp/rclcpp.hpp>
#include <pkg00_interfaces/srv/add_ints.hpp>
using pkg00_interfaces::srv::AddInts;
using namespace std::chrono_literals;

class Cli:public rclcpp::Node{
public:
    Cli(const std::string &node_name):Node(node_name){
        RCLCPP_INFO(this->get_logger(),"客户端已启动");
        client_ = this->create_client<AddInts>("service_add_nums");
    }

    bool isConnect(){
        while (!client_->wait_for_service(1s))
        {
            if(!rclcpp::ok()){
                RCLCPP_INFO(rclcpp::get_logger("rclcpp"),"强制退出");
                return false;
            }
            RCLCPP_INFO(this->get_logger(),"服务连接中，请稍侯...");
        }
        return true;
    }

    rclcpp::Client<AddInts>::FutureAndRequestId send_request(int32_t num1, int32_t
num2){
        auto request = std::make_shared<AddInts::Request>();
        request->num1 = num1;
        request->num2 = num2;
        return client_->async_send_request(request);
    }

private:
    rclcpp::Client<AddInts>::SharedPtr client_;

};

int main(int argc, char* argv[]){
    if(argc !=3 ){
        RCLCPP_INFO(rclcpp::get_logger("rclcpp"),"请提交两个整数");
        return 1;
    }

    rclcpp::init(argc,argv);
    auto client = std::make_shared<Cli>("Cli");

    bool flag = client->isConnect();
    if(!flag){
        RCLCPP_INFO(rclcpp::get_logger("rclcpp"),"服务连接失败");
        return 0;
    }
    auto response = client->send_request(atoi(argv[1]),atoi(argv[2]));
    if(rclcpp::spin_until_future_complete(client,response) ==
rclcpp::FutureReturnCode::SUCCESS){
        RCLCPP_INFO(client->get_logger(),"请求正常处理");
        RCLCPP_INFO(client->get_logger(),"响应结果: %d",response.get()->sum);
    }else{
        RCLCPP_INFO(client->get_logger(),"请求异常");
    }

    rclcpp::shutdown();
    return 0;
}
```

4.2 python

demo01_server_py.py:

```
import rclpy
from rclpy.node import Node
from pkg00_interfaces.srv import AddInts
import sys

class Ser(Node):
    def __init__(self, node_name):
        super().__init__(node_name)
        self.get_logger().info("服务端已启动,等待请求...")
        self.server_ = self.create_service(AddInts, "service_add_nums",
self.add_callback)

    def add_callback(self, req, res):
        res.sum = req.num1 + req.num2
        self.get_logger().info(f"计算过程: {req.num1} + {req.num2} = {res.sum}")
        return res

def main(args=None):
    rclpy.init(args=args)
    server = Ser("Ser")
    rclpy.spin(server)
    rclpy.shutdown()
```

demo01_client_py.py:

```
import rclpy
from rclpy.node import Node
from pkg00_interfaces.srv import AddInts
import sys

class Cli(Node):
    def __init__(self, node_name):
        super().__init__(node_name)
        self.get_logger().info("客户端已启动")
        self.client_ = self.create_client(AddInts, "service_add_nums")

    def is_connect(self):
        while not self.client_.wait_for_service(timeout_sec=1.0):
            if not rclpy.ok():
                self.get_logger().info("强制退出")
                return False
            self.get_logger().info("服务连接中，请稍侯...")
        return True

    def send_request(self, num1, num2):
        req = AddInts.Request()
        req.num1 = num1
        req.num2 = num2
        return self.client_.call_async(req)

def main(args=None):
    rclpy.init(args=args)

    if len(sys.argv) != 3:
        print("请提交两个整数")
        return 1

    client = Cli("Cli")

    if not client.is_connect():
        client.get_logger().info("服务连接失败")
        return 0

    future = client.send_request(int(sys.argv[1]), int(sys.argv[2]))

    while rclpy.ok():
        rclpy.spin_once(client)
        if future.done():
            try:
                response = future.result()
                client.get_logger().info(f"请求正常处理")
                client.get_logger().info(f"响应结果: {response.sum}")
            except Exception as e:
                client.get_logger().error(f"请求异常: {e}")
            break

    rclpy.shutdown()
```

5.编译运行

编译:

```
cd ros2_ws
colcon build
```

5.1 运行:

c++:

```
source install/setup.bash
ros2 run pkg04_service_cpp demo01_server_cpp
```

```
source install/setup.bash
ros2 run pkg04_service_cpp demo01_client_cpp 1 10
```

python:

```
source install/setup.bash
ros2 run pkg04_service_py demo01_server_py 1 10
```

```
source install/setup.bash
ros2 run pkg04_service_py demo01_client_py 1 10
```

2025-9-15(topic_custom)

1.创建功能包

```
cd ros2_ws/src
```

消息接口:

```
ros2 pkg create pkg00_interfaces --dependencies rosidl_default_generators
```

c++:

```
ros2 pkg create pkg03_topic_cpp --build-type ament_cmake --dependencies rclcpp
pkg00_interfaces
```

python:

```
ros2 pkg create pkg03_topic_py --build-type ament_python --dependencies rclpy
pkg00_interfaces
```

2.创建执行文件

接口包配置:

在pkg00_interfaces下创建文件夹msg，msg中创建文件Student.msg，并在Student.msg中填入以下内容:

```
string name
int32 age
float64 height
```

c++:

```
在pkg03_topic_py/src下新建demo02_talker_cpp.cpp
在pkg03_topic_py/src下新建demo02_listener_cpp.cpp
```

python:

```
在pkg03_topic_py/pkg03_topic_py下新建demo02_talker_py.py
在pkg03_topic_py/pkg03_topic_py下新建demo02_listener_py.py
```

3.配置编译文件

消息接口：

CMakeLists.txt：添加到find_package(rosidl_default_generators REQUIRED)后面

```
rosidl_generate_interfaces(${PROJECT_NAME}
  "msg/Student.msg"
)
```

package.xml：添加到rosidl_default_generators后面

```
<exec_depend>rosidl_default_runtime</exec_depend>
<member_of_group>rosidl_interface_packages</member_of_group>
```

c++: CMakeLists.txt

```
add_executable(demo02_talker_cpp src/demo02_talker_cpp.cpp)
add_executable(demo02_listener_cpp src/demo02_listener_cpp.cpp)

ament_target_dependencies(
  demo02_talker_cpp
  "rclcpp"
  "pkg00_interfaces"
)
ament_target_dependencies(
  demo02_listener_cpp
  "rclcpp"
  "pkg00_interfaces"
)

install(TARGETS
  demo02_talker_cpp
  demo02_listener_cpp
  DESTINATION lib/${PROJECT_NAME})
```

python: setup.py

```
entry_points={
        'console_scripts': [
            'demo02_talker_py = pkg03_topic_py.demo02_talker_py:main',
            'demo02_listener_py = pkg03_topic_py.demo02_listener_py:main',
        ],
    },
```

4.编写代码

4.1 c++

demo02_talker_cpp.cpp:

#include <pkg00_interfaces/msg/student.hpp>此行报红解决方法:

```
打开VsCode左边的文件列表，右键点击pkg00_interfaces，选择复制路径
```

```
法一：打开VsCode左边的文件列表，打开.vscode中c_cpp_properties.json文件，
在"includePath":下面加入"复制的内容/include/**",再把src改为install
```

```
法二：在输入#include <pkg00_interfaces/msg/student.hpp>后，鼠标放到此行，点击fix，再点击第一个灯泡，出现界面后下拉到workspace相关信息框，把 复制的内容/include/**填入此框内，再把src改为install
```

```
#include <rclcpp/rclcpp.hpp>
#include <pkg00_interfaces/msg/student.hpp>
using namespace std::chrono_literals;
using pkg00_interfaces::msg::Student;

class Pub: public rclcpp::Node{
public:
    Pub(const std::string &node_name):Node(node_name){
        RCLCPP_INFO(this->get_logger(),"发送端已经启动");
        talker_ = this->create_publisher<Student>("topic_stu",10);
        timer_ = this-
>create_wall_timer(500ms,std::bind(&Pub::timer_callback,this));
    }

private:
    void timer_callback(){
        auto stu = Student();
        stu.name = "asd";
        stu.age = 100;
        stu.height = 1.70;
        RCLCPP_INFO(this->get_logger(),"发送学生信息: name = %s, age = %d, height =
%.2f", stu.name.c_str(), stu.age, stu.height);
        talker_->publish(stu);
    }
    rclcpp::Publisher<Student>::SharedPtr talker_;
    rclcpp::TimerBase::SharedPtr timer_;
};

int main(int argc, char const *argv[])
{
    rclcpp::init(argc, argv);
    auto talker = std::make_shared<Pub>("pub");
    rclcpp::spin(talker);
    rclcpp::shutdown();
    return 0;
}
```

demo02_listener_cpp.cpp:

```
#include <rclcpp/rclcpp.hpp>
#include <pkg00_interfaces/msg/student.hpp>
using namespace std::chrono_literals;
using pkg00_interfaces::msg::Student;
```

```
class Sub: public rclcpp::Node{
public:
    Sub(const std::string &node_name):Node(node_name){
        RCLCPP_INFO(this->get_logger(),"接收端已经启动");
        listener_ = this->create_subscription<Student>("topic_stu",10,
            std::bind(&Sub::sub_callback,this,std::placeholders::_1));
    }

private:
    void sub_callback(const Student &stu){
        RCLCPP_INFO(this->get_logger(),"收到学生信息: name = %s, age = %d, height =
%.2f", stu.name.c_str(), stu.age, stu.height);
    }
    rclcpp::Subscription<Student>::SharedPtr listener_;
};

int main(int argc, char const *argv[])
{
    rclcpp::init(argc, argv);
    auto listener = std::make_shared<Sub>("sub");
    rclcpp::spin(listener);
    rclcpp::shutdown();
    return 0;
}
```

4.2 python

demo02_talker_py.py:

```
import rclpy
from rclpy.node import Node
from pkg00_interfaces.msg import Student

class Pub(Node):
    def __init__(self,node_name):
        super().__init__(node_name)
        self.get_logger().info("发送端已经启动")
        self.talker_ = self.create_publisher(Student,"topic_stu",10)
        self.timer_ = self.create_timer(0.5,self.timer_callback)
        self.count_ = 0

    def timer_callback(self):
        stu = Student()
        stu.name = "asd"
        stu.age = 1
        stu.height = 1.7
        stu.age = self.count_
        self.count_ += 1
        self.get_logger().info(f'发送学生信息:  name = {stu.name}, age = {stu.age},
height = {stu.height}')
        self.talker_.publish(stu)
```

```
def main():
    rclpy.init()
    talker = Pub("pub")
    rclpy.spin(talker)
    rclpy.shutdown()
```

demo02_listener_py.py:

```
import rclpy
from rclpy.node import Node
from pkg00_interfaces.msg import Student

class Sub(Node):
    def __init__(self,node_name):
        super().__init__(node_name)
        self.get_logger().error("接收端已经启动")
        self.listener_ =
self.create_subscription(Student,"topic_stu",self.sub_callback,10)

    def sub_callback(self,stu):
        self.get_logger().error(f'收到学生信息: name = {stu.name}, age = {stu.age},
height = {stu.height}')

def main():
    rclpy.init()
    listener = Sub("sub")
    rclpy.spin(listener)
    rclpy.shutdown()
```

5.编译运行

编译:

```
cd ros2_ws
colcon build
```

5.1 运行:

查看消息接口:

```
ros2 interface show pkg00_interfaces/msg/Student
```

c++:

```
source install/setup.bash
ros2 run pkg03_topic_cpp demo02_talker_cpp
```

```
source install/setup.bash
ros2 run pkg03_topic_cpp demo02_listener_cpp
```

python:

```
source install/setup.bash
ros2 run pkg03_topic_py demo02_talker_py
```

```
source install/setup.bash
ros2 run pkg03_topic_py demo02_listener_py
```

2025-9-8、2025-9-11(topic)

1.创建功能包

```
cd ros2_ws/src
```

c++:

```
ros2 pkg create pkg02_topic_cpp --build-type  ament_cmake --dependencies rclcpp
std_msgs
```

python:

```
ros2 pkg create pkg02_topic_py --build-type ament_python --dependencies rclpy
std_msgs
```

2.创建执行文件

c++:

```
在pkg02_topic_cpp/src下新建demo01_talker_cpp.cpp
在pkg02_topic_cpp/src下新建demo01_listener_cpp.cpp
```

python:

```
在pkg02_topic_py/pkg02_topic_py下新建demo01_talker_py.py
在pkg02_topic_py/pkg02_topic_py下新建demo01_listener_py.py
```

3.配置编译文件

c++: CMakeLists.txt

```
add_executable(demo01_talker_cpp src/demo01_talker_cpp.cpp)
add_executable(demo01_listener_cpp src/demo01_listener_cpp.cpp)

ament_target_dependencies(
  demo01_talker_cpp
  "rclcpp"
  "std_msgs"
)
ament_target_dependencies(
  demo01_listener_cpp
  "rclcpp"
  "std_msgs"
)

install(TARGETS
  demo01_talker_cpp
  demo01_listener_cpp
  DESTINATION lib/${PROJECT_NAME})
```

python: setup.py

```
entry_points={
        'console_scripts': [
            'demo01_talker_py = pkg02_topic_py.demo01_talker_py:main',
            'demo01_listener_py = pkg02_topic_py.demo01_listener_py:main',
        ],
    },
```

4.编写代码

4.1 c++

demo01_talker_cpp.cpp:

```
#include <rclcpp/rclcpp.hpp>
#include <std_msgs/msg/string.hpp>
using namespace std::chrono_literals;

class Pub: public rclcpp::Node{
public:
    Pub(const std::string &node_name):Node(node_name){
        RCLCPP_INFO(this->get_logger(),"发送端已经启动");
        talker_ = this->create_publisher<std_msgs::msg::String>("topic",10);
        timer_ = this-
>create_wall_timer(500ms,std::bind(&Pub::timer_callback,this));
    }

private:
    void timer_callback(){
        auto msg = std_msgs::msg::String();
        msg.data = "hello";
        RCLCPP_INFO(this->get_logger(),"发送消息: %s", msg.data.c_str());
        talker_->publish(msg);
    }
    rclcpp::Publisher<std_msgs::msg::String>::SharedPtr talker_;
    rclcpp::TimerBase::SharedPtr timer_;
};

int main(int argc, char const *argv[])
{
    rclcpp::init(argc, argv);
    auto talker = std::make_shared<Pub>("pub");
    rclcpp::spin(talker);
    rclcpp::shutdown();
    return 0;
}
```

demo01_listener_cpp.cpp:

```
#include <rclcpp/rclcpp.hpp>
#include <std_msgs/msg/string.hpp>
using namespace std::chrono_literals;

class Sub: public rclcpp::Node{
public:
    Sub(const std::string &node_name):Node(node_name){
        RCLCPP_INFO(this->get_logger(),"接收端已经启动");
        listener_ = this->create_subscription<std_msgs::msg::String>("topic",10,
            std::bind(&Sub::sub_callback,this,std::placeholders::_1));
    }

private:
    void sub_callback(const std_msgs::msg::String &msg){
        RCLCPP_INFO(this->get_logger(),"收到消息: %s",msg.data.c_str());
    }
    rclcpp::Subscription<std_msgs::msg::String>::SharedPtr listener_;
};

int main(int argc, char const *argv[])
{
    rclcpp::init(argc, argv);
    auto listener = std::make_shared<Sub>("sub");
    rclcpp::spin(listener);
    rclcpp::shutdown();
    return 0;
}
```

4.2 python

demo01_talker_py.py:

```
import rclpy
from rclpy.node import Node
from std_msgs.msg import String

class Pub(Node):
    def __init__(self,node_name):
        super().__init__(node_name)
        self.get_logger().info("发送端已经启动")
        self.talker_ = self.create_publisher(String,"topic",10)
        self.timer_ = self.create_timer(0.5,self.timer_callback)
        self.count_ = 0

    def timer_callback(self):
        msg = String()
        msg.data = "hi"+f':{self.count_}'
        self.get_logger().info(f'发送消息：{msg.data}')
        self.count_ += 1
        self.talker_.publish(msg)

def main():
    rclpy.init()
    talker = Pub("pub")
    rclpy.spin(talker)
    rclpy.shutdown()
```

demo01_listener_py.py:

```
import rclpy
from rclpy.node import Node
from std_msgs.msg import String

class Sub(Node):
    def __init__(self,node_name):
        super().__init__(node_name)
        self.get_logger().error("接收端已经启动")
        self.listener_ =
self.create_subscription(String,"topic",self.sub_callback,10)

    def sub_callback(self,msg):
        self.get_logger().error(f'收到消息:{msg.data}')
        # self.get_logger().warning(f'收到消息:{msg.data}')

def main():
    rclpy.init()
    listener = Sub("sub")
    rclpy.spin(listener)
    rclpy.shutdown()
```

5.编译运行

编译:

```
cd ros2_ws
colcon build
```

5.1 运行:

c++:

```
source install/setup.bash
ros2 run pkg02_topic_cpp demo01_talker_cpp
```

```
source install/setup.bash
ros2 run pkg02_topic_cpp demo01_listener_cpp
```

python:

```
source install/setup.bash
ros2 run pkg02_topic_py demo01_talker_py
```

```
source install/setup.bash
ros2 run pkg02_topic_py demo01_listener_py
```

6.ros2命令

6.1 查看消息接口类型

保持代码运行，打开一新终端，执行以下命令

```
ros2 interface show std_msgs/msg/String
```

6.2 查看所有在执行的话题

保持代码运行，打开一新终端，执行以下命令

```
ros2 topic list
```

6.3 rqt

保持代码运行，打开一新终端，执行以下命令，查看节点、话题信息

Plugins->Introspection->Node Graph

```
rqt
```

2025-9-1、2025-9-4(node)

打开终端（快捷键：ctril+alt+t)

vscode界面，左面有一个扩展包，搜索c，下载c/c++和c/c++ Extension；再搜索python，下载python

1.创建工程文件

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

2.创建功能包

c++:

```
ros2 pkg create pkg01_helloworld_cpp --build-type ament_cmake --dependencies
rclcpp
```

python:

```
ros2 pkg create pkg01_helloworld_py --build-type ament_python --dependencies rclpy
```

3.创建执行文件

c++:

```
在pkg01_helloworld_cpp/src下新建helloworld.cpp
```

python:

```
在pkg01_helloworld_py/pkg01_helloworld_py下新建helloworld.py
```

4.配置编译文件

c++: CMakeLists.txt

以下内容粘贴到find_package(rclcpp REQUIRED)下一行

```
add_executable(helloworld src/helloworld.cpp)
ament_target_dependencies(
  helloworld
  "rclcpp"
)

install(TARGETS
  helloworld
  DESTINATION lib/${PROJECT_NAME})
```

python: setup.py

```
entry_points={
        'console_scripts': [
            'helloworld = pkg01_helloworld_py.helloworld:main',
        ],
    },
```

5.编写代码

c++:

代码提示: 在输入#include <rclcpp/rclcpp.hpp>后，鼠标放到此行，点击fix，再点击第一个灯泡，出现界面后下拉到workshace相关信息框，把/opt/ros/humble/include/**填入此框内

```
#include <rclcpp/rclcpp.hpp>

class MyNode:public rclcpp::Node{
public:
    MyNode(const std::string &node_name):Node(node_name){
            RCLCPP_INFO(this->get_logger(), "helloworld");
        }
};

int main(int argc, char const *argv[])
{
    rclcpp::init(argc, argv);
    auto node = std::make_shared<MyNode>("cpp_node");
    rclcpp::spin(node);
    rclcpp::shutdown();
    return 0;
}
```

python:

```
import rclpy
from rclpy.node import Node

class MyNode(Node):
    def __init__(self,node_name):
        super().__init__(node_name)
        self.get_logger().info("helloworld")
        self.get_logger().error("helloworld")

def main():
    rclpy.init()
    node = MyNode("py_node")
    rclpy.spin(node)
    rclpy.shutdown()
```

6.编译运行

编译:

```
cd ros2_ws
colcon build
```

运行: 每次打开一个新终端，需要执行以下命令:

```
source install/setup.bash
```

c++:

```
ros2 run pkg01_helloworld_cpp helloworld
```

python:

```
ros2 run pkg01_helloworld_py helloworld
```