Ros2文档

> 当前版本为ros2 humble版本

1.开发环境配置

参考资料:

1. https://blog.csdn.net/weixin_55944949/article/details/140373710

bash Bash

```bash
sudo apt update

wget http://fishros.com/install -O fishros && . fishros
```

2.ros2基础命令

1. 使用命令行查询节点列表: ros2 node list
2. 查看节点列表及信息: ros2 node info /cpp_node
3. 查看某个话题的具体信息: ros2 topic info /turtle1/cmd_vel -v
4. 创建python功能包: ros2 pkg create demo_python_pkg --build-type ament_python --license Apache-2.0

setup.py需要进行配置注册

setup.py Python

```python
# setup.py需要进行配置注册Python节点
from setuptools import find_packages, setup

package_name = 'demo_python_pkg_test'

setup(
    name=package_name,
    version='0.0.0',
    packages=find_packages(exclude=['test']),
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='zme',
    maintainer_email='zranguai@gmail.com',
    description='TODO: Package description',
    license='Apache-2.0',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            "python_node = demo_python_pkg_test.python_node:main",
            "person_node = demo_python_pkg_test.person_node:main",
            "writer_node = demo_python_pkg_test.writer_node:main",
        ],
    },
)

```

package.xml中添加依赖声明

package.xml XML

```xml
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd" schematypens="http://www.w3.org/2001/XMLSchema"?>
<package format="3">
  <name>demo_python_pkg_test</name>
  <version>0.0.0</version>
  <description>TODO: Package description</description>
  <maintainer email="zranguai@gmail.com">zme</maintainer>
  <license>Apache-2.0</license>
  <depend>rclpy</depend>
  <!-- <depend>demo_cpp_pkg_test</depend> -->

  <test_depend>ament_copyright</test_depend>
  <test_depend>ament_flake8</test_depend>
  <test_depend>ament_pep257</test_depend>
  <test_depend>python3-pytest</test_depend>

  <export>
    <build_type>ament_python</build_type>
  </export>
</package>
```

5. 在目录下使用colcon构建功能包: colcon build(构建目录下的所有功能包)
6. 运行节点: ros2 run demo_python_pkg python_node（需要先source install/setup.bash）
7. 创建c++功能包: ros2 pkg create demo_cpp_pkg --build-type ament_cmake --license Apache-2.0

在CMakeLists.txt中注册节以及添加依赖

CMakeLists.txt Plain Text

```
cmake_minimum_required(VERSION 3.8)
project(demo_cpp_pkg_test)

if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
  add_compile_options(-Wall -Wextra -Wpedantic)
endif()

# find dependencies
find_package(ament_cmake REQUIRED)
# uncomment the following section in order to fill in
# further dependencies manually.
# find_package(<dependency> REQUIRED)

# 1.查找 rclcpp 头文件和库
find_package(rclcpp REQUIRED)
# 2. 添加可执行文件 cpp_node
add_executable(cpp_node src/cpp_node.cpp)
# 3. 为 cpp_node 添加依赖
ament_target_dependencies(cpp_node rclcpp)

add_executable(person_node src/person_node.cpp)
ament_target_dependencies(person_node rclcpp)

add_executable(learn_auto src/learn_auto.cpp)
ament_target_dependencies(learn_auto rclcpp)

# 4. 将 cpp_node 拷贝到 install 目录
install(TARGETS
  cpp_node
  person_node
  learn_auto
  DESTINATION lib/${PROJECT_NAME}
)


if(BUILD_TESTING)
  find_package(ament_lint_auto REQUIRED)
  # the following line skips the linter which checks for copyrights
  # comment the line when a copyright and license is added to all source f
iles
  set(ament_cmake_copyright_FOUND TRUE)
  # the following line skips cpplint (only works in a git repo)
  # comment the line when this package is in a git repo and when
  # a copyright and license is added to all source files
  set(ament_cmake_cpplint_FOUND TRUE)
```

```
  ament_lint_auto_find_test_dependencies()
endif()

ament_package()
```

在package.xml中添加依赖声明

package.xml XML

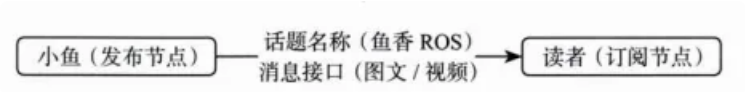
```xml
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd" schematypens="http://www.w3.org/2001/XMLSchema"?>
<package format="3">
  <name>demo_cpp_pkg_test</name>
  <version>0.0.0</version>
  <description>TODO: Package description</description>
  <maintainer email="zranguai@gmail.com">zme</maintainer>
  <license>Apache-2.0</license>
  <depend>rclcpp</depend>

  <buildtool_depend>ament_cmake</buildtool_depend>

  <test_depend>ament_lint_auto</test_depend>
  <test_depend>ament_lint_common</test_depend>

  <export>
    <build_type>ament_cmake</build_type>
  </export>
</package>

```

8. 选择某一个功能包进行构建: colcon build --packages-select demo_cpp_pkg
9. 多功能包的最佳实践: 把包都放置到my_ws/src下(mkdir -p my_ws/src)
10. 查看消息接口的详细定义: ros2 interface show geometry_msgs/msg/Twist
11. 使用命令行发布命令: ros2 topic pub /turtle1/cmd_vel geometry_msgs/msg/Twist "linear: {x: 1.0}"
12. 查询服务列表和对应接口(-t参数表示服务的接口类型,[]表示服务的接口类型): ros2 sevice list -t
13. 通过命令行调用服务(也可以通过rqt选择Plugins->Serives->Service Caller选项): ros2 service call /spawn turtlesim/srv/Spawn "{x:1, y:1}"
14. 查看参数列表: ros2 param list
15. 查看指定节点的参数描述: ros2 param describe /turtlesim background_r

16. 获取/修改节点的参数值: ros2 param get /turtlesim background_r ros2 param set /turtlesim background 255
17. 将参数导出到文件里: ros2 param dump /turtlesim > turtlesim_param.yaml
18. 运行节点时指定参数文件: ros2 run turtlesim turtlesim_node --ros-args --params-file turtlesim_param.yaml
19. 查看参数使用帮助(delete,describe,dump,get,list,load,set): ros2 param --help

3.订阅和发布-话题通信

> 单向的数据传递。ros2的话题机制有四个关键点，分别是发布者，订阅者，话题名称和话题类型。



同一个节点，既有话题又有话题订阅案例

turtle_control.cpp C++

```cpp
#include "geometry_msgs/msg/twist.hpp"
#include "rclcpp/rclcpp.hpp"
#include "turtlesim/msg/pose.hpp"

class TurtleController : public rclcpp::Node
{
public:
    TurtleController() : Node("turtle_controller")
    {
        velocity_publisher_ = this->create_publisher<geometry_msgs::msg::Twist>(
            "/turtle1/cmd_vel", 10);
        pose_subscription_ = this->create_subscription<turtlesim::msg::Pose>(
            "/turtle1/pose", 10,
            std::bind(&TurtleController::on_pose_received_, this, std::placeholders::_1));
    }

private:
    void on_pose_received_(const turtlesim::msg::Pose::SharedPtr pose) {
        auto message = geometry_msgs::msg::Twist();
        // 1.记录当前位置
        double current_x = pose->x;
        double current_y = pose->y;
        RCLCPP_INFO(this->get_logger(), "当前位置:(x=%f,y=%f)", current_x, current_y);

        // 2.计算距离目标的距离，与当前海龟朝向的角度差
        double distance =
            std::sqrt((target_x_ - current_x) * (target_x_ - current_x) +
                      (target_y_ - current_y) * (target_y_ - current_y));
        double angle =
            std::atan2(target_y_ - current_y, target_x_ - current_x) - pose->theta;

        // 3.控制策略：距离大于0.1继续运动，角度差大于0.2则原地旋转，否则直行
        if (distance > 0.1) {
        if(fabs(angle)>0.2)
        {
            message.angular.z = fabs(angle);
        }else{
            // 通过比例控制器计算输出速度
            message.linear.x = k_ * distance;

```

```
        }
        }

        // 4.限制最大值并发布消息
        if (message.linear.x > max_speed_) {
        message.linear.x = max_speed_;
        }
        velocity_publisher_->publish(message);
    }


private:
    rclcpp::Subscription<turtlesim::msg::Pose>::SharedPtr pose_subscriptio
n_;
    rclcpp::Publisher<geometry_msgs::msg::Twist>::SharedPtr velocity_publi
sher_;
    double target_x_{1.0};  // 目标位置X,设置默认值1.0
    double target_y_{1.0};  // 目标位置Y,设置默认值1.0
    double k_{1.0};         // 比例系数，控制输出=误差*比例系数
    double max_speed_{3.0}; // 最大线速度，设置默认值3.0
};

int main(int argc, char **argv)
{
    rclcpp::init(argc, argv);
    auto node = std::make_shared<TurtleController>();
    rclcpp::spin(node);
    rclcpp::shutdown();
    return 0;
}
```

发布，订阅，话题消息定义整个项目案例

消息定义接口功能包:

> 在ros2中，消息定义文件名必须以大写字母开头且只能由大，小写字母及数字组成

首先在ws/src目录下创建一个接口功能包

`ros2 pkg create status_interfaces --build-type ament_cmake --dependencies r
osidl_default_generators builtin_interfaces --license Apache-2.0`

- 消息结构体文件

SystemStatus.msg Plain Text

```
builtin_interfaces/Time stamp # 记录时间戳 builtin_interfaces接口下的Time类型
string host_name    # 系统名称
float32 cpu_percent   # CPU使用率
float32 memory_percent  # 内存使用率
float32 memory_total  # 内存总量
float32 memory_available  # 剩余有效内存
float64 net_sent    # 网络发送数据总量
float64 net_recv    # 网络接收数据总量
```

- ros2消息接口支持的9种数据类型

9种数据类型.txt Plain Text

```
bool
byte
char
float32,float64
int8,uint8
int16,uint16
int32,uint32
int63,uint63
string
```

- 在CMakeLists.txt中进行注册，声明为消息接口文件

CMakeLists.txt Plain Text

```
cmake_minimum_required(VERSION 3.8)
project(status_interfaces)

if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
  add_compile_options(-Wall -Wextra -Wpedantic)
endif()

# find dependencies
find_package(ament_cmake REQUIRED)
find_package(rosidl_default_generators REQUIRED)
find_package(builtin_interfaces REQUIRED)

rosidl_generate_interfaces(${PROJECT_NAME}
  "msg/SystemStatus.msg"
  DEPENDENCIES builtin_interfaces
)

if(BUILD_TESTING)
  find_package(ament_lint_auto REQUIRED)
  # the following line skips the linter which checks for copyrights
  # comment the line when a copyright and license is added to all source f
iles
  set(ament_cmake_copyright_FOUND TRUE)
  # the following line skips cpplint (only works in a git repo)
  # comment the line when this package is in a git repo and when
  # a copyright and license is added to all source files
  set(ament_cmake_cpplint_FOUND TRUE)
  ament_lint_auto_find_test_dependencies()
endif()

ament_package()

```

package.xml XML

```xml
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd" schematypens="http://www.w3.org/2001/XMLSchema"?>
<package format="3">
  <name>status_interfaces</name>
  <version>0.0.0</version>
  <description>TODO: Package description</description>
  <maintainer email="zranguai@gmail.com">zme</maintainer>
  <license>Apache-2.0</license>
  <member_of_group>rosidl_interface_packages</member_of_group>

  <buildtool_depend>ament_cmake</buildtool_depend>

  <depend>rosidl_default_generators</depend>
  <depend>builtin_interfaces</depend>

  <test_depend>ament_lint_auto</test_depend>
  <test_depend>ament_lint_common</test_depend>

  <export>
    <build_type>ament_cmake</build_type>
  </export>
</package>

```

最后可以使用ros2 interface show status_interfaces/msg/SystemStatus查看接口是否构建成功

发布信息

- 创建发布包

`ros2 pkg create status_publish --build-type ament_python --dependencies rclpy status_interfaces --license Apache-2.0`

- 发布的代码

sys_status_pub.py Python

```python
import rclpy
from rclpy.node import Node
from status_interfaces.msg import SystemStatus # 导入消息接口
import psutil
import platform

class SysStatusPub(Node):
    def __init__(self, node_name):
        super().__init__(node_name)
        self.status_publisher_ = self.create_publisher(
            SystemStatus, 'sys_status', 10)
        self.timer = self.create_timer(1, self.timer_callback)  #定时器1s调用一次

    def timer_callback(self):
        cpu_percent = psutil.cpu_percent()
        memory_info = psutil.virtual_memory()
        net_io_counters = psutil.net_io_counters()

        msg = SystemStatus()
        msg.stamp = self.get_clock().now().to_msg()
        msg.host_name = platform.node()
        msg.cpu_percent = cpu_percent
        msg.memory_percent = memory_info.percent
        msg.memory_total = memory_info.total / 1024 / 1024
        msg.memory_available = memory_info.available / 1024 / 1024
        msg.net_sent = net_io_counters.bytes_sent / 1024 / 1024
        msg.net_recv = net_io_counters.bytes_recv / 1024 / 1024

        self.get_logger().info(f'发布:{str(msg)}')
        self.status_publisher_.publish(msg)


def main():
    rclpy.init()
    node = SysStatusPub('sys_status_pub')
    rclpy.spin(node)
    rclpy.shutdown()
```

运行发布节点: ros2 run status_publish sys_status_pub

使用命令行输出/sys_status话题数据: ros2 topic echo /sys_status

订阅信息

- 订阅的代码

sys_status_display.cpp C++

```cpp
#include <QApplication>
#include <QLabel>
#include <QString>
#include "rclcpp/rclcpp.hpp"
#include "status_interfaces/msg/system_status.hpp"

using SystemStatus = status_interfaces::msg::SystemStatus;

class SysStatusDisplay : public rclcpp::Node {
 public:
  SysStatusDisplay() : Node("sys_status_display") {
    subscription_ = this->create_subscription<SystemStatus>(
        "sys_status", 10, [&](const SystemStatus::SharedPtr msg) -> void {
          label_->setText(get_qstr_from_msg(msg));
        });
    // 创建一个空的 SystemStatus 对象，转化成 QString 进行显示
    label_ = new QLabel(get_qstr_from_msg(std::make_shared<SystemStatus>()
));
    label_->show();
  }
QString get_qstr_from_msg(const SystemStatus::SharedPtr msg) {
    std::stringstream show_str;
    show_str
        << "===========系统状态可视化显示工具============\n"
        << "数 据 时 间:\t" << msg->stamp.sec << "\ts\n"
        << "用  户  名:\t" << msg->host_name << "\t\n"
        << "CPU使用率:\t" << msg->cpu_percent << "\t%\n"
        << "内存使用率:\t" << msg->memory_percent << "\t%\n"
        << "内存总大小:\t" << msg->memory_total << "\tMB\n"
        << "剩余有效内存:\t" << msg->memory_available << "\tMB\n"
        << "网络发送量:\t" << msg->net_sent << "\tMB\n"
        << "网络接收量:\t" << msg->net_recv << "\tMB\n"
        << "========================================";

    return QString::fromStdString(show_str.str());
  }


 private:
  rclcpp::Subscription<SystemStatus>::SharedPtr subscription_;
  QLabel* label_;
};


int main(int argc, char* argv[]) {
```

```
    rclcpp::init(argc, argv);
    QApplication app(argc, argv);
    auto node = std::make_shared<SysStatusDisplay>();
    // 使用多线程 因为rclcpp::spin和app.exec都会阻塞程序运行
    std::thread spin_thread([&]() -> void { rclcpp::spin(node); });
    spin_thread.detach();
    app.exec();
    rclcpp::shutdown();
    return 0;
}
```

4.服务端和客户端–服务与参数通信

> 双向的数据传递。服务是基于请求和响应的双向通信机制，而参数主要用于管理节点的设置。

python实现服务端与客户端

自定义服务接口

- 创建接口功能包

`ros2 pkg create chapt4_interfaces --build-type ament_cmake --dependencies rosidl_default_generators sensor_msgs --license Apache-2.0`

- 创建消息文件(---上为Request部分， ---下为Response部分)

FaceDetector.srv — Plain Text

```
sensor_msgs/Image image  # 原始图像
---
int16 number     # 人脸数
float32 use_time # 识别耗时
int32[] top      # 人脸在图像中的位置
int32[] right
int32[] bottom
int32[] left
```

- CMakeLists.txt里面进行注册

CMakeLists.txt Plain Text

```
cmake_minimum_required(VERSION 3.8)
project(chap4_interfaces)

if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
  add_compile_options(-Wall -Wextra -Wpedantic)
endif()

# find dependencies
find_package(ament_cmake REQUIRED)
find_package(rosidl_default_generators REQUIRED)
find_package(sensor_msgs REQUIRED)

rosidl_generate_interfaces(${PROJECT_NAME}
  "srv/FaceDetector.srv"
  "srv/Patrol.srv"
  DEPENDENCIES sensor_msgs
)

if(BUILD_TESTING)
  find_package(ament_lint_auto REQUIRED)
  # the following line skips the linter which checks for copyrights
  # comment the line when a copyright and license is added to all source f
iles
  set(ament_cmake_copyright_FOUND TRUE)
  # the following line skips cpplint (only works in a git repo)
  # comment the line when this package is in a git repo and when
  # a copyright and license is added to all source files
  set(ament_cmake_cpplint_FOUND TRUE)
  ament_lint_auto_find_test_dependencies()
endif()

ament_package()
```

- package.xml里面添加 `<member_of_group>rosidl_interface_packages</member_of_group>`

package.xml XML

```xml
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd" schematypens="http://www.w3.org/2001/XMLSchema"?>
<package format="3">
  <name>chap4_interfaces</name>
  <version>0.0.0</version>
  <description>TODO: Package description</description>
  <maintainer email="zranguai@gmail.com">zme</maintainer>
  <license>Apache-2.0</license>

  <buildtool_depend>ament_cmake</buildtool_depend>

  <depend>rosidl_default_generators</depend>
  <depend>rosidl_default_runtime</depend>
  <depend>sensor_msgs</depend>

  <member_of_group>rosidl_interface_packages</member_of_group>

  <test_depend>ament_lint_auto</test_depend>
  <test_depend>ament_lint_common</test_depend>

  <export>
    <build_type>ament_cmake</build_type>
  </export>
</package>

```

创建python服务端客户端功能包

- 创建demo_python_service功能包

`ros2 pkg create demo_python_service --build-type ament_python --dependencies rclpy chatp4_interfaces --license Apache-2.0`

- setup.py中添加相关配置(图片，launch, 入口等)。其中 `'share/' + package_name+"/resource"` 为模板路径， `['resource/default.jpg','resource/test1.jpg']` 为源文件路径。会将源文件路径拷贝到目标路径下。

setup.py Python

```python
from setuptools import find_packages, setup
from glob import glob
package_name = 'demo_python_service'

setup(
    name=package_name,
    version='0.0.0',
    packages=find_packages(exclude=['test']),
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
        ('share/' + package_name+"/resource", ['resource/default.jpg','resource/test1.jpg']),
        ('share/' + package_name+'/launch', glob('launch/*.launch.py')),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='mzebra',
    maintainer_email='mzebra@foxmail.com',
    description='TODO: Package description',
    license='Apache-2.0',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'learn_face_detect=demo_python_service.learn_face_detect:main',
            'face_detect_node=demo_python_service.face_detect_node:main',
            'face_detect_client_node=demo_python_service.face_detect_client_node:main',
        ],
    },
)
```

- 服务端代码创建。创建demo_python_service.py。其中CvBridge用于转换opencv格式和ros2格式的图像

face_detect_node.py Python

```python
import rclpy
from rclpy.node import Node
from chap4_interfaces.srv import FaceDetector
from ament_index_python.packages import get_package_share_directory
from cv_bridge import CvBridge # 用于转换格式
import cv2
import face_recognition
import time
from rcl_interfaces.msg import SetParametersResult

class FaceDetectorionNode(Node):
    def __init__(self):
        super().__init__('face_detection_node')
        self.bridge = CvBridge()
        # 第一个是消息接口，第二个是服务的名称 第三个是处理请求的回调函数
        self.service = self.create_service(FaceDetector, '/face_detect', self.detect_face_callback)
        self.defaut_image_path = get_package_share_directory('demo_python_service')+'/resource/default.jpg'
        self.upsample_times = 1
        self.model = "hog"
        # 声明和获取参数
        self.declare_parameter('face_locations_upsample_times', 1) # 第一个是参数名称 第二个是默认值
        self.declare_parameter('face_locations_model', "hog")
        self.model = self.get_parameter("face_locations_model").value  # 获取参数值
        self.upsample_times = self.get_parameter("face_locations_upsample_times").value
        self.set_parameters([rclpy.Parameter('face_locations_model', rclpy.Parameter.Type.STRING, 'cnn')])
        self.add_on_set_parameters_callback(self.parameter_callback)

    def parameter_callback(self, parameters):
        for parameter in parameters:
            self.get_logger().info(
                f'参数 {parameter.name} 设置为：{parameter.value}')
            if parameter.name == 'face_locations_upsample_times':
                self.upsample_times = parameter.value
            if parameter.name == 'face_locations_model':
                self.mode = parameter.value
        return SetParametersResult(successful=True)

    # request:来自客户端的请求数据，response:防止处理结果，最后return返回。
    def detect_face_callback(self, request, response):
```

```
        if request.image.data:
            cv_image = self.bridge.imgmsg_to_cv2(
                request.image)
        else:
            cv_image = cv2.imread(self.defaut_image_path)
        start_time = time.time()
        self.get_logger().info('加载完图像，开始检测')
        face_locations = face_recognition.face_locations(cv_image, number_of_times_to_upsample=self.upsample_times, model=self.model)
        end_time = time.time()
        self.get_logger().info(f'检测完成，耗时{end_time-start_time}')
        response.number = len(face_locations)
        response.use_time = end_time - start_time
        for top, right, bottom, left in face_locations:
            response.top.append(top)
            response.right.append(right)
            response.bottom.append(bottom)
            response.left.append(left)
        return response


def main(args=None):
    rclpy.init(args=args)
    node = FaceDetectorionNode()
    rclpy.spin(node)
    rclpy.shutdown()
```

最后测试：`source install/setup.bash` `ros2 service call /face_detect chapt4_interfaces/srv/FaceDetector`

运行节点后，使用 `ros2 param list` 可以查看参数列表

ros2 param list — Plain Text

```
/face_detection_node:
    face_locations_model
    face_locations_upsample_times
    use_sim_time
```

除了启动节点后通过命令行设置参数，还可以在启动节点时指定参数的值，只需要使用 `--ros-args` 和 `-p` 来指定就行，例如下面

`ros2 run demo_python_service face_detect_node --ros-args -p face_locations_model:=cnn`

还可以使用参数回调，订阅更新参数(当参数被更新时，ros2会自动调用这个回调函数，并传入更新的参数组进行更新)

订阅更新参数.py

```python
...
from rcl_interfaces.msg import SetParametersResult

class FaceDetectorionNode(Node):
    def __init__(self):
        self.add_on_set_parameters_callback(self.parameter_callback)

    def parameter_callback(self, parameters):
        # 在该方法中，对传入的参数数组进行遍历，输出参数名称和值
        for parameter in parameters:
            self.get_logger().info(
                f'参数 {parameter.name} 设置为：{parameter.value}')
            if parameter.name == 'face_locations_upsample_times':
                self.upsample_times = parameter.value
            if parameter.name == 'face_locations_model':
                self.mode = parameter.value
        return SetParametersResult(successful=True)
```

此时节点就能够收到参数更新的事件了，在本节点中改变自身节点参数

节点中改变自身节点参数

```python
self.set_parameters([rclpy.Parameter('face_locations_model', rclpy.Parameter.Type.STRING, 'cnn')])
```

- 客户端代码实现。

face_detect_client_node.py Python

```python
import rclpy
from rclpy.node import Node
from chap4_interfaces.srv import FaceDetector
from sensor_msgs.msg import Image
from ament_index_python.packages import get_package_share_directory
import cv2
from cv_bridge import CvBridge
from rcl_interfaces.srv import SetParameters
from rcl_interfaces.msg import Parameter, ParameterValue, ParameterType


class FaceDetectorClient(Node):
    def __init__(self):
        super().__init__('face_detect_client')
        self.client = self.create_client(FaceDetector, '/face_detect')
        self.bridge = CvBridge()
        self.test1_image_path = get_package_share_directory(
            'demo_python_service')+'/resource/zidane.jpg'
        self.image = cv2.imread(self.test1_image_path)

    def send_request(self):
        # 1.判断服务是否上线
        while self.client.wait_for_service(timeout_sec=1.0) is False:
            self.get_logger().info(f'等待服务端上线....')
        # 2.构造 Request
        request = FaceDetector.Request()
        request.image = self.bridge.cv2_to_imgmsg(self.image)
        # 3.发送并 spin 等待服务处理完成
        future = self.client.call_async(request)
        rclpy.spin_until_future_complete(self, future)  # 执行spin的同时检测future是否完成
        # 4.根据处理结果
        response = future.result()
        self.get_logger().info(
            f'接收到响应：图像中共有：{response.number}张脸，耗时{response.use_time}')
        # 注释show_face_locations，防止显示堵塞无法多次请求
        self.show_face_locations(response)

    def call_set_parameters(self, parameters):
        # 1. 创建一个客户端，并等待服务上线
        client = self.create_client(
            SetParameters, '/face_detection_node/set_parameters')
        while not client.wait_for_service(timeout_sec=1.0):
            self.get_logger().info('等待参数设置服务端上线....')
```

```
        # 2. 创建请求对象
        request = SetParameters.Request()
        request.parameters = parameters
        # 3. 异步调用、等待并返回响应结果
        future = client.call_async(request)
        rclpy.spin_until_future_complete(self, future)
        response = future.result()
        return response

    def update_detect_model(self,model):
        # 1.创建一个参数对象
        param = Parameter()
        param.name = "face_locations_model"
        # 2.创建参数值对象并赋值
        new_model_value = ParameterValue()
        new_model_value.type = ParameterType.PARAMETER_STRING
        new_model_value.string_value = model
        param.value = new_model_value
        # 3.请求更新参数并处理
        response = self.call_set_parameters([param])
        for result in response.results:
            if result.successful:
                self.get_logger().info(f'参数 {param.name} 设置为{model}')
            else:
                self.get_logger().info(f'参数设置失败，原因为：{result.reason
}')


    def show_face_locations(self, response):
        for i in range(response.number):
            top = response.top[i]
            right = response.right[i]
            bottom = response.bottom[i]
            left = response.left[i]
            cv2.rectangle(self.image, (left, top),
                          (right, bottom), (255, 0, 0), 2)

        cv2.imshow('Face Detection Result', self.image)
        cv2.waitKey(0)


def main(args=None):
    rclpy.init(args=args)
    face_detect_client = FaceDetectorClient()
    face_detect_client.update_detect_model('hog')
    face_detect_client.send_request()
    face_detect_client.update_detect_model('cnn')
    face_detect_client.send_request()
```

```
91      rclpy.spin(face_detect_client)
92      rclpy.shutdown()
```

这里还可以在客户端更改服务端节点参数的值。

调用服务设置其他节点的参数 Python

```python
...
from rcl_interfaces.srv import SetParameters
from rcl_interfaces.msg import Parameter, ParameterValue, ParameterType

class FaceDetectorClient(Node):
    def __init__(self):
        ...

    def call_set_parameters(self, parameters):
        # 1. 创建一个客户端，并等待服务上线
        client = self.create_client(
            SetParameters, '/face_detection_node/set_parameters')
        while not client.wait_for_service(timeout_sec=1.0):
            self.get_logger().info('等待参数设置服务端上线....')
        # 2. 创建请求对象
        request = SetParameters.Request()
        request.parameters = parameters
        # 3. 异步调用、等待并返回响应结果
        future = client.call_async(request)
        rclpy.spin_until_future_complete(self, future)
        response = future.result()
        return response

    def update_detect_model(self,model):
        # 1.创建一个参数对象
        param = Parameter()
        param.name = "face_locations_model"
        # 2.创建参数值对象并赋值
        new_model_value = ParameterValue()
        new_model_value.type = ParameterType.PARAMETER_STRING
        new_model_value.string_value = model
        param.value = new_model_value
        # 3.请求更新参数并处理
        response = self.call_set_parameters([param])
        for result in response.results:
            if result.successful:
                self.get_logger().info(f'参数 {param.name} 设置为{model}')
            else:
                self.get_logger().info(f'参数设置失败，原因为：{result.reason
}')

```

可以在命令行中展示参数设置接口

`ros2 interface show rcl_interfaces/srv/SetParameters`

创建c++服务端客户端功能包

- 创建自定义服务接口

Patrol.srv Plain Text

```
float32 target_x  # 目标x值
float32 target_y  # 目标y值
---
int8 SUCCESS = 1  # 定义常量，表示成功
int8 FAIL = 0     # 定义常量，表示失败
int8 result       # 处理结果
```

- 服务端代码实现

turtle_control.cpp C++

```cpp
#include "geometry_msgs/msg/twist.hpp"
#include "rclcpp/rclcpp.hpp"
#include "turtlesim/msg/pose.hpp"
#include "chap4_interfaces/srv/patrol.hpp"
using Patrol = chap4_interfaces::srv::Patrol;
#include "rcl_interfaces/msg/set_parameters_result.hpp"
using SetParametersResult = rcl_interfaces::msg::SetParametersResult;


class TurtleController : public rclcpp::Node
{
public:
TurtleController() : Node("turtle_controller")
{
    velocity_publisher_ = this->create_publisher<geometry_msgs::msg::Twist>(
        "/turtle1/cmd_vel", 10);
    pose_subscription_ = this->create_subscription<turtlesim::msg::Pose>(
        "/turtle1/pose", 10,
        std::bind(&TurtleController::on_pose_received_, this, std::placeholders::_1));
    // 3.创建服务 <>里面是服务的接口类型
    patrol_server_ = this->create_service<Patrol>(
        "patrol",
        [&](const std::shared_ptr<Patrol::Request> request,
        std::shared_ptr<Patrol::Response> response) -> void {
            // 判断巡逻点是否在模拟器边界内
            if ((0 < request->target_x && request->target_x < 12.0f)
                && (0 < request->target_y && request->target_y < 12.0f))
{
                target_x_ = request->target_x;
                target_y_ = request->target_y;
                response->result = Patrol::Response::SUCCESS;
            }else{
                response->result = Patrol::Response::FAIL;
            }
        });
    // 声明和获取参数初始值
    this->declare_parameter("k", 1.0);
    this->declare_parameter("max_speed", 1.0);
    this->get_parameter("k", k_);
    this->get_parameter("max_speed", max_speed_);

    // 添加参数设置回调
    parameters_callback_handle_ = this->add_on_set_parameters_callback(
```

```
        [&](const std::vector<rclcpp::Parameter> &params)
        -> SetParametersResult {
            // 遍历参数
            for (auto param : params) {
                RCLCPP_INFO(this->get_logger(), "更新参数 %s 值为: %f",param.get_name().c_str(), param.as_double());
                if (param.get_name() == "k") {
                    k_ = param.as_double();
                } else if (param.get_name() == "max_speed") {
                    max_speed_ = param.as_double();
                }
            }
            auto result = SetParametersResult();
            result.successful = true;
            return result;
        });
}

private:
void on_pose_received_(const turtlesim::msg::Pose::SharedPtr pose) {
    auto message = geometry_msgs::msg::Twist();
    // 1.记录当前位置
    double current_x = pose->x;
    double current_y = pose->y;
    RCLCPP_INFO(this->get_logger(), "当前位置:(x=%f,y=%f)", current_x, current_y);

    // 2.计算距离目标的距离，与当前海龟朝向的角度差
    double distance =
    std::sqrt((target_x_ - current_x) * (target_x_ - current_x) +
    (target_y_ - current_y) * (target_y_ - current_y));
    double angle =
    std::atan2(target_y_ - current_y, target_x_ - current_x) - pose->theta;

    // 3.控制策略：距离大于0.1继续运动，角度差大于0.2则原地旋转，否则直行
    if (distance > 0.1) {
        if(fabs(angle)>0.2)
        {
            message.angular.z = fabs(angle);
      }else{
        // 通过比例控制器计算输出速度
        message.linear.x = k_ * distance;

      }
    }

    // 4.限制最大值并发布消息
```

```
    if (message.linear.x > max_speed_) {
       message.linear.x = max_speed_;
    }
    velocity_publisher_->publish(message);
  }


private:
  // 2.添加 Patrol 类型服务共享指针 patrol_server_ 为成员变量
  rclcpp::Service<Patrol>::SharedPtr patrol_server_;

    rclcpp::Subscription<turtlesim::msg::Pose>::SharedPtr pose_subscription_;
    rclcpp::Publisher<geometry_msgs::msg::Twist>::SharedPtr velocity_publisher_;
    double target_x_{1.0};  // 目标位置X,设置默认值1.0
    double target_y_{1.0};  // 目标位置Y,设置默认值1.0
    double k_{1.0};         // 比例系数，控制输出=误差*比例系数
    double max_speed_{3.0}; // 最大线速度，设置默认值3.0
  OnSetParametersCallbackHandle::SharedPtr parameters_callback_handle_;

};

int main(int argc, char **argv)
{
    rclcpp::init(argc, argv);
    auto node = std::make_shared<TurtleController>();
    rclcpp::spin(node);
    rclcpp::shutdown();
    return 0;
}
```

参数的声明与设置如下:

可以使用 `ros2 param list` 进行查看, 可以使用如下方式进行设置 `ros2 param set /turtle_controller k 2.0` 这里参数的值已经被重新设置，但是还需要通过订阅参数更新事件进行更新。

c++中参数的声明与设置 C++

```cpp
...
class TurtleController : public rclcpp::Node {
    public:
        TurtleController() : Node("turtle_controller") {
            ...
            this->declare_parameter("k", 1.0);
            this->declare_parameter("max_speed", 1.0);
            this->get_parameter("k", k_);
            this->get_parameter("max_speed", max_speed_);
            ...

            // 添加参数设置回调，需要通过参数回调更新参数
            parameters_callback_handle_ = this->add_on_set_parameters_callback(
                [&](const std::vector<rclcpp::Parameter> &params)
                -> SetParametersResult {
                    // 遍历参数
                    for (auto param : params) {
                        RCLCPP_INFO(this->get_logger(), "更新参数 %s 值为: %f",param.get_name().c_str(), param.as_double());
                        if (param.get_name() == "k") {
                            k_ = param.as_double();
                        } else if (param.get_name() == "max_speed") {
                            max_speed_ = param.as_double();
                        }
                    }
                    auto result = SetParametersResult();
                    result.successful = true;
                    return result;
            });
        }
    private:
        OnSetParametersCallbackHandle::SharedPtr parameters_callback_handle_;
}
```

这里的参数回调函数解析: add_on_set_parameters_callback方法只有一个参数就是回调函数。该回调函数的返回值是SetParametersResult消息接口对象，回调函数的参数是rclcpp::Parameter数组的静态引用。在函数体内对数组所有参数进行遍历和输出。在根据参数的名称更新对应属性，构造一个SetParametersResult对象。这个时候使用命令行设置参数就可以更新参数了。除了使用命令行外，还可以使用代码设置自身节点参数 `this->set_parameter(rclcpp::Parameter("k", 2.0))`

- 在CMakeLists.txt里面进行注册

CMakeLists.txt Plain Text

```
cmake_minimum_required(VERSION 3.8)
project(demo_cpp_service)

if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
  add_compile_options(-Wall -Wextra -Wpedantic)
endif()

# find dependencies
find_package(ament_cmake REQUIRED)
find_package(chap4_interfaces REQUIRED)
find_package(rclcpp REQUIRED)
find_package(geometry_msgs REQUIRED)
find_package(turtlesim REQUIRED)

add_executable(turtle_control src/turtle_control.cpp)
ament_target_dependencies(turtle_control rclcpp geometry_msgs turtlesim ch
ap4_interfaces)

add_executable(patrol_client  src/patrol_client.cpp)
ament_target_dependencies(patrol_client rclcpp geometry_msgs turtlesim cha
p4_interfaces)

install(TARGETS
  turtle_control
  patrol_client
  DESTINATION lib/${PROJECT_NAME})

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

if(BUILD_TESTING)
  find_package(ament_lint_auto REQUIRED)
  # the following line skips the linter which checks for copyrights
  # comment the line when a copyright and license is added to all source f
iles
  set(ament_cmake_copyright_FOUND TRUE)
  # the following line skips cpplint (only works in a git repo)
  # comment the line when this package is in a git repo and when
  # a copyright and license is added to all source files
  set(ament_cmake_cpplint_FOUND TRUE)
  ament_lint_auto_find_test_dependencies()
endif()

ament_package()
```

- 客户端实现代码

patrol_client.cpp C++

```cpp
#include <chrono>
#include <cstdlib>
#include <ctime>
#include "rclcpp/rclcpp.hpp"
#include "chap4_interfaces/srv/patrol.hpp"
#include <chrono> // 引入时间相关头文件
#include "rcl_interfaces/msg/parameter.hpp"
#include "rcl_interfaces/msg/parameter_value.hpp"
#include "rcl_interfaces/msg/parameter_type.hpp"
#include "rcl_interfaces/srv/set_parameters.hpp"

// 使用时间单位的字面量，可以在代码中使用 s 和 ms 表示时间
using namespace std::chrono_literals;
using Patrol = chap4_interfaces::srv::Patrol;
using SetP = rcl_interfaces::srv::SetParameters;

class PatrolClient : public rclcpp::Node
{
public:
PatrolClient() : Node("patrol_client")
{
    patrol_client_ = this->create_client<Patrol>("patrol");
    timer_ = this->create_wall_timer(10s, std::bind(&PatrolClient::timer_
callback, this));
    srand(time(NULL)); // 初始化随机数种子，使用当前时间作为种子
}
void update_server_param_k(double k)
{
    // 1.创建一个参数对象
    auto param = rcl_interfaces::msg::Parameter();
    param.name = "k";
    // 2.创建参数值对象并赋值
    auto param_value = rcl_interfaces::msg::ParameterValue();
    param_value.type = rcl_interfaces::msg::ParameterType::PARAMETER_DOUB
LE;
    param_value.double_value = k;
    param.value = param_value;
    // 3.请求更新参数并处理
    auto response = call_set_parameters(param);
    if (response == nullptr)
    {
        RCLCPP_WARN(this->get_logger(), "参数修改失败");
        return;
    }
    else
```

```
    {
        for (auto result : response->results)
            {
                if (result.successful)
                {
                    RCLCPP_INFO(this->get_logger(), "参数k 已修改为: %f", k
);
                }
                else
                {
                    RCLCPP_WARN(this->get_logger(), "参数k 失败原因: %s", r
esult.reason.c_str());
                }
            }
    }
}

std::shared_ptr<SetP::Response> call_set_parameters(
rcl_interfaces::msg::Parameter &parameter)
{
    // 1. 创建客户端等待服务上线
    auto param_client = this->create_client<SetP>(
    "/turtle_controller/set_parameters");
    while (!param_client->wait_for_service(std::chrono::seconds(1)))
        {
            if (!rclcpp::ok())
            {
                RCLCPP_ERROR(this->get_logger(), "等待服务的过程中被打断..."
);
                return nullptr;
            }
            RCLCPP_INFO(this->get_logger(), "等待参数设置服务端上线中");
        }
    // 2. 创建请求对象
    auto request =
    std::make_shared<SetP::Request>();
    request->parameters.push_back(parameter);
    // 3. 异步调用、等待并返回响应结果
    auto future = param_client->async_send_request(request);
    rclcpp::spin_until_future_complete(this->get_node_base_interface(), f
uture);
    auto response = future.get();
    return response;
}

void timer_callback()
{
    // 1.等待服务端上线
```

```
    while (!patrol_client_->wait_for_service(1s))
        {
            // 等待时检测rclcpp的状态
            if (!rclcpp::ok())
            {
                RCLCPP_ERROR(this->get_logger(), "等待服务的过程中被打断...");
                return;
            }
            RCLCPP_INFO(this->get_logger(), "等待服务端上线中");
        }
    // 2.构造请求的
    auto request = std::make_shared<Patrol::Request>();
    request->target_x = rand() % 15;
    request->target_y = rand() % 15;
    RCLCPP_INFO(this->get_logger(), "请求巡逻: (%f,%f)", request->target_x, request->target_y);
    // 3.发送异步请求，然后等待返回，返回时调用回调函数
    patrol_client_->async_send_request(
        request,
        [&](rclcpp::Client<Patrol>::SharedFuture result_future) -> void
        {
          auto response = result_future.get();
          if (response->result == Patrol::Response::SUCCESS)
          {
            RCLCPP_INFO(this->get_logger(), "目标点处理成功");
          }
          else if (response->result == Patrol::Response::FAIL)
          {
            RCLCPP_INFO(this->get_logger(), "目标点处理失败");
          }
        });
  }

private:
  rclcpp::TimerBase::SharedPtr timer_;
  rclcpp::Client<Patrol>::SharedPtr patrol_client_;
};
int main(int argc, char **argv)
{
  rclcpp::init(argc, argv);
  auto node = std::make_shared<PatrolClient>();
  node->update_server_param_k(1.5);
  rclcpp::spin(node);
  rclcpp::shutdown();
  return 0;
}
```

这里可以通过其他节点(例如客户端节点)设置服务端参数。

在头文件部分引入消息和服务相关接口的头文件并为接口建立别名SeP。

在`call_set_parameters`方法中，创建一个客户端，等待服务上线，服务上线后创建请求对象，将参数放到对象数组中，然后发送异步请求等待响应并将结果返回。

使用launch启动脚本

> ros2支持使用Python,XML,YAML三种格式来编写launch脚本，其中Python更加灵活。

- 在节点下面创建launch目录，并创建demo.launch.py

如下面代码的黑体所示: launch可以将参数传递给节点。首先添加一个参数声明的动作action_declare_arg_max_speed,然后再action_node_turtle_control中添加parameters,最后使用launch中的max_speed值替换节点中的max_speed参数值。

demo.launch.py Python

```python
import launch
import launch_ros

def generate_launch_description():
    action_declare_arg_max_spped = launch.actions.DeclareLaunchArgument('launch_max_speed', default_value='2.0')

    action_node_turtle_control = launch_ros.actions.Node(
        package='demo_cpp_service',
        executable="turtle_control",
        output='screen',
        parameters=[{'max_speed': launch.substitutions.LaunchConfiguration(
  'launch_max_speed', default='2.0')}],
    )
    action_node_patrol_client = launch_ros.actions.Node(
        package='demo_cpp_service',
        executable="patrol_client",
        output='log',
    )
    action_node_turtlesim_node = launch_ros.actions.Node(
        package='turtlesim',
        executable='turtlesim_node',
        output='both',
    )
   # 合成启动描述并返回
    launch_description = launch.LaunchDescription([
        action_declare_arg_max_spped,
        action_node_turtle_control,
        action_node_patrol_client,
        action_node_turtlesim_node
    ])
    return launch_description
```

launch工具在运行python格式的启动脚本时，会在文件中搜索generate_launch_description的函数来获取对启动内容的描述。所有上面代码定义这个函数。

然后依次创建launch_ros.actions.Node类的对象，其中package参数用于指定功能包名称，executable参数指定可执行文件名称，output表示指定日志输出的位置，screen表示屏幕，log表示日志，both表示前两者同时输出。最后将这些启动对象合成数组。

然后在CMakeLists.txt中添加如下命令

CMakeLists.txt中添加命令 Plain Text

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

或者在setup.py文件下添加如下命令

setup.py中添加命令 Plain Text

```
...
from glob import glob
setup(
      data_files=[
        ...
        ('share/' + package_name+'/launch', glob('launch/*.launch.py')),
    ],
)
```

最后启动launch节点

启动launch节点 Plain Text

```
source install/setup.bash
ros2 launch demo_cpp_service demo.launch.py
```

5.ros2常用开发工具

坐标交换工具