

Ros2文档

1.开发环境配置

2.ros2基础命令

3.订阅和发布-话题通信

同一个节点，既有话题又有话题订阅案例

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:

发布信息

订阅信息

4.服务端和客户端-服务与参数通信

python实现服务端与客户端

自定义服务接口

创建python服务端客户端功能包

创建c++服务端客户端功能包

使用launch启动脚本

5.ros2常用开发工具

坐标交换工具

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 |

```
1  sudo apt update
2
3  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 /turtle/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 |
1  # setup.py需要进行配置注册Python节点
2  from setuptools import find_packages, setup
3
4  package_name = 'demo_python_pkg_test'
5
6  setup(
7      name=package_name,
8      version='0.0.0',
9      packages=find_packages(exclude=['test']),
10     data_files=[
11         ('share/ament_index/resource_index/packages',
12          ['resource/' + package_name]),
13         ('share/' + package_name, ['package.xml']),
14     ],
15     install_requires=['setuptools'],
16     zip_safe=True,
17     maintainer='zme',
18     maintainer_email='zranguai@gmail.com',
19     description='TODO: Package description',
20     license='Apache-2.0',
21     tests_require=['pytest'],
22     entry_points={
23         'console_scripts': [
24             "python_node = demo_python_pkg_test.python_node:main",
25             "person_node = demo_python_pkg_test.person_node:main",
26             "writer_node = demo_python_pkg_test.writer_node:main",
27         ],
28     },
29 )
30
```

package.xml中添加依赖声明

package.xml
XML

```

1  <?xml version="1.0"?>
2  <?xml-model href="http://download.ros.org/schema/package_format3.xsd" schematypens="http://www.w3.org/2001/XMLSchema"?>
3  <package format="3">
4    <name>demo_python_pkg_test</name>
5    <version>0.0.0</version>
6    <description>TODO: Package description</description>
7    <maintainer email="zranguai@gmail.com">zme</maintainer>
8    <license>Apache-2.0</license>
9    <depend>rclpy</depend>
10   <!-- <depend>demo_cpp_pkg_test</depend> -->
11
12   <test_depend>ament_copyright</test_depend>
13   <test_depend>ament_flake8</test_depend>
14   <test_depend>ament_pep257</test_depend>
15   <test_depend>python3-pytest</test_depend>
16
17   <export>
18     <build_type>ament_python</build_type>
19   </export>
20 </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

```
1  cmake_minimum_required(VERSION 3.8)
2  project(demo_cpp_pkg_test)
3
4  if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
5      add_compile_options(-Wall -Wextra -Wpedantic)
6  endif()
7
8  # find dependencies
9  find_package(ament_cmake REQUIRED)
10 # uncomment the following section in order to fill in
11 # further dependencies manually.
12 # find_package(<dependency> REQUIRED)
13
14 # 1. 查找 rclcpp 头文件和库
15 find_package(rclcpp REQUIRED)
16 # 2. 添加可执行文件 cpp_node
17 add_executable(cpp_node src/cpp_node.cpp)
18 # 3. 为 cpp_node 添加依赖
19 ament_target_dependencies(cpp_node rclcpp)
20
21 add_executable(person_node src/person_node.cpp)
22 ament_target_dependencies(person_node rclcpp)
23
24 add_executable(learn_auto src/learn_auto.cpp)
25 ament_target_dependencies(learn_auto rclcpp)
26
27 # 4. 将 cpp_node 拷贝到 install 目录
28 install(TARGETS
29     cpp_node
30     person_node
31     learn_auto
32     DESTINATION lib/${PROJECT_NAME}
33 )
34
35
36 if(BUILD_TESTING)
37     find_package(ament_lint_auto REQUIRED)
38     # the following line skips the linter which checks for copyrights
39     # comment the line when a copyright and license is added to all source f
40     # files
41     set(ament_cmake_copyright_FOUND TRUE)
42     # the following line skips cpplint (only works in a git repo)
43     # comment the line when this package is in a git repo and when
44     # a copyright and license is added to all source files
45     set(ament_cmake_cpplint_FOUND TRUE)
```

```

45     ament_lint_auto_find_test_dependencies()
46 endif()
47
48 ament_package()
49

```

在package.xml中添加依赖声明

package.xml XML

```

1  <?xml version="1.0"?>
2  <?xml-model href="http://download.ros.org/schema/package_format3.xsd" sche
    matypens="http://www.w3.org/2001/XMLSchema"?>
3  <package format="3">
4      <name>demo_cpp_pkg_test</name>
5      <version>0.0.0</version>
6      <description>TODO: Package description</description>
7      <maintainer email="zranguai@gmail.com">zme</maintainer>
8      <license>Apache-2.0</license>
9      <depend>rclcpp</depend>
10
11     <buildtool_depend>ament_cmake</buildtool_depend>
12
13     <test_depend>ament_lint_auto</test_depend>
14     <test_depend>ament_lint_common</test_depend>
15
16     <export>
17         <build_type>ament_cmake</build_type>
18     </export>
19 </package>
20

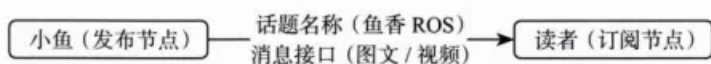
```

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 service 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. 查看指定节点参数描述: `ros2 param describe /turtlesim background_r`

16. 获取/修改节点的值: `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

```

1  #include "geometry_msgs/msg/twist.hpp"
2  #include "rclcpp/rclcpp.hpp"
3  #include "turtlesim/msg/pose.hpp"
4
5  class TurtleController : public rclcpp::Node
6  {
7  public:
8      TurtleController() : Node("turtle_controller")
9      {
10         velocity_publisher_ = this->create_publisher<geometry_msgs::msg::Twist>(
11             "/turtle1/cmd_vel", 10);
12         pose_subscription_ = this->create_subscription<turtlesim::msg::Pose>(
13             "/turtle1/pose", 10,
14             std::bind(&TurtleController::on_pose_received_, this, std::placeholders::_1));
15     }
16
17 private:
18     void on_pose_received_(const turtlesim::msg::Pose::SharedPtr pose) {
19         auto message = geometry_msgs::msg::Twist();
20         // 1.记录当前位置
21         double current_x = pose->x;
22         double current_y = pose->y;
23         RCLCPP_INFO(this->get_logger(), "当前位置:(x=%f,y=%f)", current_x,
24             current_y);
25
26         // 2.计算距离目标的距离,与当前海龟朝向的角度差
27         double distance =
28             std::sqrt((target_x_ - current_x) * (target_x_ - current_x) +
29                 (target_y_ - current_y) * (target_y_ - current_y));
30         double angle =
31             std::atan2(target_y_ - current_y, target_x_ - current_x) - pose->theta;
32
33         // 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
34         if (distance > 0.1) {
35             if(fabs(angle)>0.2)
36             {
37                 message.angular.z = fabs(angle);
38             }else{
39                 // 通过比例控制器计算输出速度
40                 message.linear.x = k_ * distance;

```

```

41     }
42     }
43
44     // 4.限制最大值并发布消息
45     if (message.linear.x > max_speed_) {
46         message.linear.x = max_speed_;
47     }
48     velocity_publisher_>publish(message);
49 }
50
51
52 private:
53     rclcpp::Subscription<turtlesim::msg::Pose>::SharedPtr pose_subscription_;
54     rclcpp::Publisher<geometry_msgs::msg::Twist>::SharedPtr velocity_publisher_;
55     double target_x_{1.0}; // 目标位置X,设置默认值1.0
56     double target_y_{1.0}; // 目标位置Y,设置默认值1.0
57     double k_{1.0}; // 比例系数,控制输出=误差*比例系数
58     double max_speed_{3.0}; // 最大线速度,设置默认值3.0
59 };
60
61 int main(int argc, char **argv)
62 {
63     rclcpp::init(argc, argv);
64     auto node = std::make_shared<TurtleController>();
65     rclcpp::spin(node);
66     rclcpp::shutdown();
67     return 0;
68 }

```

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

```
1 builtin_interfaces/Time stamp # 记录时间戳 builtin_interfaces接口下的Time类型
2 string host_name # 系统名称
3 float32 cpu_percent # CPU使用率
4 float32 memory_percent # 内存使用率
5 float32 memory_total # 内存总量
6 float32 memory_available # 剩余有效内存
7 float64 net_sent # 网络发送数据总量
8 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

```
1 bool
2 byte
3 char
4 float32,float64
5 int8,uint8
6 int16,uint16
7 int32,uint32
8 int63,uint63
9 string
```

- 在CMakeLists.txt中进行注册，声明为消息接口文件

```
1  cmake_minimum_required(VERSION 3.8)
2  project(status_interfaces)
3
4  if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
5      add_compile_options(-Wall -Wextra -Wpedantic)
6  endif()
7
8  # find dependencies
9  find_package(ament_cmake REQUIRED)
10 find_package(rosidl_default_generators REQUIRED)
11 find_package(builtin_interfaces REQUIRED)
12
13 rosidl_generate_interfaces(${PROJECT_NAME}
14     "msg/SystemStatus.msg"
15     DEPENDENCIES builtin_interfaces
16 )
17
18 if(BUILD_TESTING)
19     find_package(ament_lint_auto REQUIRED)
20     # the following line skips the linter which checks for copyrights
21     # comment the line when a copyright and license is added to all source f
22     # files
23     set(ament_cmake_copyright_FOUND TRUE)
24     # the following line skips cpplint (only works in a git repo)
25     # comment the line when this package is in a git repo and when
26     # a copyright and license is added to all source files
27     set(ament_cmake_cpplint_FOUND TRUE)
28     ament_lint_auto_find_test_dependencies()
29 endif()
30
31 ament_package()
```

```
1 <?xml version="1.0"?>
2 <?xml-model href="http://download.ros.org/schema/package_format3.xsd" sche
  matypens="http://www.w3.org/2001/XMLSchema"?>
3 <package format="3">
4   <name>status_interfaces</name>
5   <version>0.0.0</version>
6   <description>TODO: Package description</description>
7   <maintainer email="zranguai@gmail.com">zme</maintainer>
8   <license>Apache-2.0</license>
9   <member_of_group>roslint_interface_packages</member_of_group>
10
11   <buildtool_depend>ament_cmake</buildtool_depend>
12
13   <depend>roslint_default_generators</depend>
14   <depend>builtin_interfaces</depend>
15
16   <test_depend>ament_lint_auto</test_depend>
17   <test_depend>ament_lint_common</test_depend>
18
19   <export>
20     <build_type>ament_cmake</build_type>
21   </export>
22 </package>
23
```

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

```

1  import rclpy
2  from rclpy.node import Node
3  from status_interfaces.msg import SystemStatus # 导入消息接口
4  import psutil
5  import platform
6
7  class SysStatusPub(Node):
8      def __init__(self, node_name):
9          super().__init__(node_name)
10         self.status_publisher_ = self.create_publisher(
11             SystemStatus, 'sys_status', 10)
12         self.timer = self.create_timer(1, self.timer_callback) #定时器1s调
           用一次
13
14     def timer_callback(self):
15         cpu_percent = psutil.cpu_percent()
16         memory_info = psutil.virtual_memory()
17         net_io_counters = psutil.net_io_counters()
18
19         msg = SystemStatus()
20         msg.stamp = self.get_clock().now().to_msg()
21         msg.host_name = platform.node()
22         msg.cpu_percent = cpu_percent
23         msg.memory_percent = memory_info.percent
24         msg.memory_total = memory_info.total / 1024 / 1024
25         msg.memory_available = memory_info.available / 1024 / 1024
26         msg.net_sent = net_io_counters.bytes_sent / 1024 / 1024
27         msg.net_recv = net_io_counters.bytes_recv / 1024 / 1024
28
29         self.get_logger().info(f'发布:{str(msg)}')
30         self.status_publisher_.publish(msg)
31
32
33     def main():
34         rclpy.init()
35         node = SysStatusPub('sys_status_pub')
36         rclpy.spin(node)
37         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

```

1  #include <QApplication>
2  #include <QLabel>
3  #include <QString>
4  #include "rclcpp/rclcpp.hpp"
5  #include "status_interfaces/msg/system_status.hpp"
6
7  using SystemStatus = status_interfaces::msg::SystemStatus;
8
9  class SysStatusDisplay : public rclcpp::Node {
10 public:
11     SysStatusDisplay() : Node("sys_status_display") {
12         subscription_ = this->create_subscription<SystemStatus>(
13             "sys_status", 10, [&](const SystemStatus::SharedPtr msg) -> void {
14                 label_->setText(get_qstr_from_msg(msg));
15             });
16         // 创建一个空的 SystemStatus 对象, 转化成 QString 进行显示
17         label_ = new QLabel(get_qstr_from_msg(std::make_shared<SystemStatus>())
18         );
19         label_->show();
20     }
21
22     QString get_qstr_from_msg(const SystemStatus::SharedPtr msg) {
23         std::stringstream show_str;
24         show_str
25             << "=====系统状态可视化显示工具=====\\n"
26             << "数 据 时 间:\\t" << msg->stamp.sec << "\\ts\\n"
27             << "用 户 名:\\t" << msg->host_name << "\\t\\n"
28             << "CPU使用率:\\t" << msg->cpu_percent << "\\t%\\n"
29             << "内存使用率:\\t" << msg->memory_percent << "\\t%\\n"
30             << "内存总大小:\\t" << msg->memory_total << "\\tMB\\n"
31             << "剩余有效内存:\\t" << msg->memory_available << "\\tMB\\n"
32             << "网络发送量:\\t" << msg->net_sent << "\\tMB\\n"
33             << "网络接收量:\\t" << msg->net_recv << "\\tMB\\n"
34             << "=====\\n";
35
36         return QString::fromStdString(show_str.str());
37     }
38
39 private:
40     rclcpp::Subscription<SystemStatus>::SharedPtr subscription_;
41     QLabel* label_;
42 };
43
44 int main(int argc, char* argv[]) {

```

```

45     rclcpp::init(argc, argv);
46     QApplication app(argc, argv);
47     auto node = std::make_shared<SysStatusDisplay>();
48     // 使用多线程 因为rclcpp::spin和app.exec都会阻塞程序运行
49     std::thread spin_thread([&]() -> void { rclcpp::spin(node); });
50     spin_thread.detach();
51     app.exec();
52     rclcpp::shutdown();
53     return 0;
54 }

```

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 r
osidl_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 |

```

1  sensor_msgs/Image image # 原始图像
2  ---
3  int16 number           # 人脸数
4  float32 use_time       # 识别耗时
5  int32[] top            # 人脸在图像中的位置
6  int32[] right
7  int32[] bottom
8  int32[] left

```

- CMakeLists.txt里面进行注册

```
1  cmake_minimum_required(VERSION 3.8)
2  project(chap4_interfaces)
3
4  if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
5      add_compile_options(-Wall -Wextra -Wpedantic)
6  endif()
7
8  # find dependencies
9  find_package(ament_cmake REQUIRED)
10 find_package(rosidl_default_generators REQUIRED)
11 find_package(sensor_msgs REQUIRED)
12
13 rosidl_generate_interfaces(${PROJECT_NAME}
14     "srv/FaceDetector.srv"
15     "srv/Patrol.srv"
16     DEPENDENCIES sensor_msgs
17 )
18
19 if(BUILD_TESTING)
20     find_package(ament_lint_auto REQUIRED)
21     # the following line skips the linter which checks for copyrights
22     # comment the line when a copyright and license is added to all source f
23     set(ament_cmake_copyright_FOUND TRUE)
24     # the following line skips cpplint (only works in a git repo)
25     # comment the line when this package is in a git repo and when
26     # a copyright and license is added to all source files
27     set(ament_cmake_cpplint_FOUND TRUE)
28     ament_lint_auto_find_test_dependencies()
29 endif()
30
31 ament_package()
```

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


```
1 <?xml version="1.0"?>
2 <?xml-model href="http://download.ros.org/schema/package_format3.xsd" schematypens="http://www.w3.org/2001/XMLSchema"?>
3 <package format="3">
4   <name>chap4_interfaces</name>
5   <version>0.0.0</version>
6   <description>TODO: Package description</description>
7   <maintainer email="zranguai@gmail.com">zme</maintainer>
8   <license>Apache-2.0</license>
9
10  <buildtool_depend>ament_cmake</buildtool_depend>
11
12  <depend>roslaunch</depend>
13  <depend>roslaunch</depend>
14  <depend>sensor_msgs</depend>
15
16  <member_of_group>roslaunch_interface_packages</member_of_group>
17
18  <test_depend>ament_lint_auto</test_depend>
19  <test_depend>ament_lint_common</test_depend>
20
21  <export>
22    <build_type>ament_cmake</build_type>
23  </export>
24 </package>
25
```

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。

```
1  from setuptools import find_packages, setup
2  from glob import glob
3  package_name = 'demo_python_service'
4
5  setup(
6      name=package_name,
7      version='0.0.0',
8      packages=find_packages(exclude=['test']),
9      data_files=[
10         ('share/ament_index/resource_index/packages',
11          ['resource/' + package_name]),
12         ('share/' + package_name, ['package.xml']),
13         ('share/' + package_name + "/resource", ['resource/default.jpg', 'resource/test1.jpg']),
14         ('share/' + package_name + '/launch', glob('launch/*.launch.py')),
15     ],
16     install_requires=['setuptools'],
17     zip_safe=True,
18     maintainer='mzebra',
19     maintainer_email='mzebra@foxmail.com',
20     description='TODO: Package description',
21     license='Apache-2.0',
22     tests_require=['pytest'],
23     entry_points={
24         'console_scripts': [
25             'learn_face_detect=demo_python_service.learn_face_detect:main'
26         ],
27         'face_detect_node=demo_python_service.face_detect_node:main',
28         'face_detect_client_node=demo_python_service.face_detect_client_node:main',
29     ],
30 )
```

- 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

```

1  import rclpy
2  from rclpy.node import Node
3  from chap4_interfaces.srv import FaceDetector
4  from ament_index_python.packages import get_package_share_directory
5  from cv_bridge import CvBridge # 用于转换格式
6  import cv2
7  import face_recognition
8  import time
9  from rcl_interfaces.msg import SetParametersResult
10
11 class FaceDetectorionNode(Node):
12     def __init__(self):
13         super().__init__('face_detection_node')
14         self.bridge = CvBridge()
15         # 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
16         self.service = self.create_service(FaceDetector, '/face_detect', s
elf.detect_face_callback)
17         self.default_image_path = get_package_share_directory('demo_python_
service')+ '/resource/default.jpg'
18         self.upsample_times = 1
19         self.model = "hog"
20         # 声明和获取参数
21         self.declare_parameter('face_locations_upsample_times', 1) # 第一个
是参数名称 第二个是默认值
22         self.declare_parameter('face_locations_model', "hog")
23         self.model = self.get_parameter("face_locations_model").value #
获取参数值
24         self.upsample_times = self.get_parameter("face_locations_upsample_
times").value
25         self.set_parameters([rclpy.Parameter('face_locations_model', rclpy
.Parameter.Type.STRING, 'cnn')])
26         self.add_on_set_parameters_callback(self.parameter_callback)
27
28     def parameter_callback(self, parameters):
29         for parameter in parameters:
30             self.get_logger().info(
31                 f'参数 {parameter.name} 设置为: {parameter.value}')
32         if parameter.name == 'face_locations_upsample_times':
33             self.upsample_times = parameter.value
34         if parameter.name == 'face_locations_model':
35             self.model = parameter.value
36         return SetParametersResult(successful=True)
37
38     # request:来自客户端的请求数据, response:防止处理结果, 最后return返回。
39     def detect_face_callback(self, request, response):

```

```

40         if request.image.data:
41             cv_image = self.bridge.imgmsg_to_cv2(
42                 request.image)
43         else:
44             cv_image = cv2.imread(self.default_image_path)
45             start_time = time.time()
46             self.get_logger().info('加载完图像, 开始检测')
47             face_locations = face_recognition.face_locations(cv_image, number_
of_times_to_upsample=self.upsample_times, model=self.model)
48             end_time = time.time()
49             self.get_logger().info(f'检测完成, 耗时{end_time-start_time}')
50             response.number = len(face_locations)
51             response.use_time = end_time - start_time
52         for top, right, bottom, left in face_locations:
53             response.top.append(top)
54             response.right.append(right)
55             response.bottom.append(bottom)
56             response.left.append(left)
57         return response
58
59
60     def main(args=None):
61         rclpy.init(args=args)
62         node = FaceDetectorionNode()
63         rclpy.spin(node)
64         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 |
1 /face_detection_node:
2     face_locations_model
3     face_locations_upsample_times
4     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，并传入更新的参数数组进行更新)

订阅更新参数.py

Python |

```
1  ...
2  from rcl_interfaces.msg import SetParametersResult
3
4  class FaceDetectorionNode(Node):
5      def __init__(self):
6          self.add_on_set_parameters_callback(self.parameter_callback)
7
8      def parameter_callback(self, parameters):
9          # 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
10         for parameter in parameters:
11             self.get_logger().info(
12                 f'参数 {parameter.name} 设置为: {parameter.value}')
13             if parameter.name == 'face_locations_upsample_times':
14                 self.upsample_times = parameter.value
15             if parameter.name == 'face_locations_model':
16                 self.mode = parameter.value
17         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 |

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

- 客户端代码实现。

```

1  import rclpy
2  from rclpy.node import Node
3  from chap4_interfaces.srv import FaceDetector
4  from sensor_msgs.msg import Image
5  from ament_index_python.packages import get_package_share_directory
6  import cv2
7  from cv_bridge import CvBridge
8  from rcl_interfaces.srv import SetParameters
9  from rcl_interfaces.msg import Parameter, ParameterValue, ParameterType
10
11
12  class FaceDetectorClient(Node):
13      def __init__(self):
14          super().__init__('face_detect_client')
15          self.client = self.create_client(FaceDetector, '/face_detect')
16          self.bridge = CvBridge()
17          self.test1_image_path = get_package_share_directory(
18              'demo_python_service')+'/resource/zidane.jpg'
19          self.image = cv2.imread(self.test1_image_path)
20
21      def send_request(self):
22          # 1.判断服务是否上线
23          while self.client.wait_for_service(timeout_sec=1.0) is False:
24              self.get_logger().info(f'等待服务端上线....')
25          # 2.构造 Request
26          request = FaceDetector.Request()
27          request.image = self.bridge.cv2_to_imgmsg(self.image)
28          # 3.发送并 spin 等待服务处理完成
29          future = self.client.call_async(request)
30          rclpy.spin_until_future_complete(self, future) # 执行spin的同时检测
future是否完成
31          # 4.根据处理结果
32          response = future.result()
33          self.get_logger().info(
34              f'接收到响应：图像中共有：{response.number}张脸，耗时{response.use_
time}')
35          # 注释show_face_locations, 防止显示堵塞无法多次请求
36          self.show_face_locations(response)
37
38      def call_set_parameters(self, parameters):
39          # 1. 创建一个客户端，并等待服务上线
40          client = self.create_client(
41              SetParameters, '/face_detection_node/set_parameters')
42          while not client.wait_for_service(timeout_sec=1.0):
43              self.get_logger().info('等待参数设置服务端上线....')

```

```

44         # 2. 创建请求对象
45         request = SetParameters.Request()
46         request.parameters = parameters
47         # 3. 异步调用、等待并返回响应结果
48         future = client.call_async(request)
49         rclpy.spin_until_future_complete(self, future)
50         response = future.result()
51         return response
52
53     def update_detect_model(self, model):
54         # 1. 创建一个参数对象
55         param = Parameter()
56         param.name = "face_locations_model"
57         # 2. 创建参数值对象并赋值
58         new_model_value = ParameterValue()
59         new_model_value.type = ParameterType.PARAMETER_STRING
60         new_model_value.string_value = model
61         param.value = new_model_value
62         # 3. 请求更新参数并处理
63         response = self.call_set_parameters([param])
64         for result in response.results:
65             if result.successful:
66                 self.get_logger().info(f'参数 {param.name} 设置为{model}')
67             else:
68                 self.get_logger().info(f'参数设置失败, 原因为: {result.reason}')
69
70
71     def show_face_locations(self, response):
72         for i in range(response.number):
73             top = response.top[i]
74             right = response.right[i]
75             bottom = response.bottom[i]
76             left = response.left[i]
77             cv2.rectangle(self.image, (left, top),
78                           (right, bottom), (255, 0, 0), 2)
79
80         cv2.imshow('Face Detection Result', self.image)
81         cv2.waitKey(0)
82
83
84     def main(args=None):
85         rclpy.init(args=args)
86         face_detect_client = FaceDetectorClient()
87         face_detect_client.update_detect_model('hog')
88         face_detect_client.send_request()
89         face_detect_client.update_detect_model('cnn')
90         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。


```

1  ...
2  from rcl_interfaces.srv import SetParameters
3  from rcl_interfaces.msg import Parameter, ParameterValue, ParameterType
4
5  class FaceDetectorClient(Node):
6      def __init__(self):
7          ...
8
9      def call_set_parameters(self, parameters):
10         # 1. 创建一个客户端，并等待服务上线
11         client = self.create_client(
12             SetParameters, '/face_detection_node/set_parameters')
13         while not client.wait_for_service(timeout_sec=1.0):
14             self.get_logger().info('等待参数设置服务端上线....')
15         # 2. 创建请求对象
16         request = SetParameters.Request()
17         request.parameters = parameters
18         # 3. 异步调用、等待并返回响应结果
19         future = client.call_async(request)
20         rclpy.spin_until_future_complete(self, future)
21         response = future.result()
22         return response
23
24     def update_detect_model(self, model):
25         # 1. 创建一个参数对象
26         param = Parameter()
27         param.name = "face_locations_model"
28         # 2. 创建参数值对象并赋值
29         new_model_value = ParameterValue()
30         new_model_value.type = ParameterType.PARAMETER_STRING
31         new_model_value.string_value = model
32         param.value = new_model_value
33         # 3. 请求更新参数并处理
34         response = self.call_set_parameters([param])
35         for result in response.results:
36             if result.successful:
37                 self.get_logger().info(f'参数 {param.name} 设置为{model}')
38             else:
39                 self.get_logger().info(f'参数设置失败，原因为: {result.reason}')
40

```

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 |
1 float32 target_x # 目标x值
2 float32 target_y # 目标y值
3 ---
4 int8 SUCCESS = 1 # 定义常量, 表示成功
5 int8 FAIL = 0 # 定义常量, 表示失败
6 int8 result # 处理结果
```

- 服务端代码实现

```

1  #include "geometry_msgs/msg/twist.hpp"
2  #include "rclcpp/rclcpp.hpp"
3  #include "turtlesim/msg/pose.hpp"
4  #include "chap4_interfaces/srv/patrol.hpp"
5  using Patrol = chap4_interfaces::srv::Patrol;
6  #include "rcl_interfaces/msg/set_parameters_result.hpp"
7  using SetParametersResult = rcl_interfaces::msg::SetParametersResult;
8
9
10 class TurtleController : public rclcpp::Node
11 {
12 public:
13   TurtleController() : Node("turtle_controller")
14   {
15     velocity_publisher_ = this->create_publisher<geometry_msgs::msg::Twist>(
16       "/turtle1/cmd_vel", 10);
17     pose_subscription_ = this->create_subscription<turtlesim::msg::Pose>(
18       "/turtle1/pose", 10,
19       std::bind(&TurtleController::on_pose_received_, this, std::placeholders::_1));
20     // 3. 创建服务 <>里面是服务的接口类型
21     patrol_server_ = this->create_service<Patrol>(
22       "patrol",
23       [&](const std::shared_ptr<Patrol::Request> request,
24         std::shared_ptr<Patrol::Response> response) -> void {
25         // 判断巡逻点是否在模拟器边界内
26         if ((0 < request->target_x && request->target_x < 12.0f)
27           && (0 < request->target_y && request->target_y < 12.0f))
28         {
29           target_x_ = request->target_x;
30           target_y_ = request->target_y;
31           response->result = Patrol::Response::SUCCESS;
32         }else{
33           response->result = Patrol::Response::FAIL;
34         }
35       });
36     // 声明和获取参数初始值
37     this->declare_parameter("k", 1.0);
38     this->declare_parameter("max_speed", 1.0);
39     this->get_parameter("k", k_);
40     this->get_parameter("max_speed", max_speed_);
41
42     // 添加参数设置回调
43     parameters_callback_handle_ = this->add_on_set_parameters_callback(

```

```

43         [&](const std::vector<rclcpp::Parameter> &params)
44         -> SetParametersResult {
45             // 遍历参数
46             for (auto param : params) {
47                 RCLCPP_INFO(this->get_logger(), "更新参数 %s 值为: %f", para
48 m.get_name().c_str(), param.as_double());
49                 if (param.get_name() == "k") {
50                     k_ = param.as_double();
51                 } else if (param.get_name() == "max_speed") {
52                     max_speed_ = param.as_double();
53                 }
54             }
55             auto result = SetParametersResult();
56             result.successful = true;
57             return result;
58         });
59     }
60 private:
61 void on_pose_received(const turtlesim::msg::Pose::SharedPtr pose) {
62     auto message = geometry_msgs::msg::Twist();
63     // 1.记录当前位置
64     double current_x = pose->x;
65     double current_y = pose->y;
66     RCLCPP_INFO(this->get_logger(), "当前位置:(x=%f,y=%f)", current_x, cur
67 rent_y);
68
69     // 2.计算距离目标的距离,与当前海龟朝向的角度差
70     double distance =
71     std::sqrt((target_x_ - current_x) * (target_x_ - current_x) +
72     (target_y_ - current_y) * (target_y_ - current_y));
73     double angle =
74     std::atan2(target_y_ - current_y, target_x_ - current_x) - pose->thet
75 a;
76
77     // 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
78     if (distance > 0.1) {
79         if(fabs(angle)>0.2)
80         {
81             message.angular.z = fabs(angle);
82         }else{
83             // 通过比例控制器计算输出速度
84             message.linear.x = k_ * distance;
85         }
86     }
87
88     // 4.限制最大值并发布消息

```

```

88     if (message.linear.x > max_speed_) {
89         message.linear.x = max_speed_;
90     }
91     velocity_publisher_->publish(message);
92 }
93
94
95 private:
96     // 2.添加 Patrol 类型服务共享指针 patrol_server_ 为成员变量
97     rclcpp::Service<Patrol>::SharedPtr patrol_server_;
98
99     rclcpp::Subscription<turtlesim::msg::Pose>::SharedPtr pose_subscripti
100 on_;
101     rclcpp::Publisher<geometry_msgs::msg::Twist>::SharedPtr velocity_publ
102 isher_;
103     double target_x_{1.0}; // 目标位置X,设置默认值1.0
104     double target_y_{1.0}; // 目标位置Y,设置默认值1.0
105     double k_{1.0}; // 比例系数,控制输出=误差*比例系数
106     double max_speed_{3.0}; // 最大线速度,设置默认值3.0
107     OnSetParametersCallbackHandle::SharedPtr parameters_callback_handle_;
108 };
109
110 int main(int argc, char **argv)
111 {
112     rclcpp::init(argc, argv);
113     auto node = std::make_shared<TurtleController>();
114     rclcpp::spin(node);
115     rclcpp::shutdown();
116     return 0;
117 }

```

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。

```

1  ...
2  class TurtleController : public rclcpp::Node {
3      public:
4          TurtleController() : Node("turtle_controller") {
5              ...
6              this->declare_parameter("k", 1.0);
7              this->declare_parameter("max_speed", 1.0);
8              this->get_parameter("k", k_);
9              this->get_parameter("max_speed", max_speed_);
10             ...
11
12             // 添加参数设置回调，需要通过参数回调更新参数
13             parameters_callback_handle_ = this->add_on_set_parameters_callback(
14                 [&](const std::vector<rclcpp::Parameter> &params)
15                 -> SetParametersResult {
16                     // 遍历参数
17                     for (auto param : params) {
18                         RCLCPP_INFO(this->get_logger(), "更新参数 %s 值为: %f",
19                                     param.get_name().c_str(), param.as_double());
20                         if (param.get_name() == "k") {
21                             k_ = param.as_double();
22                         } else if (param.get_name() == "max_speed") {
23                             max_speed_ = param.as_double();
24                         }
25                     }
26                     auto result = SetParametersResult();
27                     result.successful = true;
28                     return result;
29                 });
30             private:
31             OnSetParametersCallbackHandle::SharedPtr parameters_callback_handle_;
32         }

```

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

```
1  cmake_minimum_required(VERSION 3.8)
2  project(demo_cpp_service)
3
4  if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
5    add_compile_options(-Wall -Wextra -Wpedantic)
6  endif()
7
8  # find dependencies
9  find_package(ament_cmake REQUIRED)
10 find_package(chap4_interfaces REQUIRED)
11 find_package(rclcpp REQUIRED)
12 find_package(geometry_msgs REQUIRED)
13 find_package(turtlesim REQUIRED)
14
15 add_executable(turtle_control src/turtle_control.cpp)
16 ament_target_dependencies(turtle_control rclcpp geometry_msgs turtlesim ch
ap4_interfaces)
17
18 add_executable(patrol_client src/patrol_client.cpp)
19 ament_target_dependencies(patrol_client rclcpp geometry_msgs turtlesim cha
p4_interfaces)
20
21 install(TARGETS
22   turtle_control
23   patrol_client
24   DESTINATION lib/${PROJECT_NAME})
25
26 install(DIRECTORY launch
27   DESTINATION share/${PROJECT_NAME})
28
29 if(BUILD_TESTING)
30   find_package(ament_lint_auto REQUIRED)
31   # the following line skips the linter which checks for copyrights
32   # comment the line when a copyright and license is added to all source f
iles
33   set(ament_cmake_copyright_FOUND TRUE)
34   # the following line skips cpplint (only works in a git repo)
35   # comment the line when this package is in a git repo and when
36   # a copyright and license is added to all source files
37   set(ament_cmake_cpplint_FOUND TRUE)
38   ament_lint_auto_find_test_dependencies()
39 endif()
40
41 ament_package()
```

- 客户端实现代码


```

1  #include <chrono>
2  #include <cstdlib>
3  #include <ctime>
4  #include "rclcpp/rclcpp.hpp"
5  #include "chap4_interfaces/srv/patrol.hpp"
6  #include <chrono> // 引入时间相关头文件
7  #include "rcl_interfaces/msg/parameter.hpp"
8  #include "rcl_interfaces/msg/parameter_value.hpp"
9  #include "rcl_interfaces/msg/parameter_type.hpp"
10 #include "rcl_interfaces/srv/set_parameters.hpp"
11
12 // 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
13 using namespace std::chrono_literals;
14 using Patrol = chap4_interfaces::srv::Patrol;
15 using SetP = rcl_interfaces::srv::SetParameters;
16
17 class PatrolClient : public rclcpp::Node
18 {
19 public:
20   PatrolClient() : Node("patrol_client")
21   {
22     patrol_client_ = this->create_client<Patrol>("patrol");
23     timer_ = this->create_wall_timer(10s, std::bind(&PatrolClient::timer_
callback, this));
24     srand(time(NULL)); // 初始化随机数种子, 使用当前时间作为种子
25   }
26   void update_server_param_k(double k)
27   {
28     // 1. 创建一个参数对象
29     auto param = rcl_interfaces::msg::Parameter();
30     param.name = "k";
31     // 2. 创建参数值对象并赋值
32     auto param_value = rcl_interfaces::msg::ParameterValue();
33     param_value.type = rcl_interfaces::msg::ParameterType::PARAMETER_DOU
BLE;
34     param_value.double_value = k;
35     param.value = param_value;
36     // 3. 请求更新参数并处理
37     auto response = call_set_parameters(param);
38     if (response == nullptr)
39     {
40       RCLCPP_WARN(this->get_logger(), "参数修改失败");
41       return;
42     }
43     else

```

```

44     {
45         for (auto result : response->results)
46         {
47             if (result.successful)
48             {
49                 RCLCPP_INFO(this->get_logger(), "参数k 已修改为: %f", k
50 );
51             }
52             else
53             {
54                 RCLCPP_WARN(this->get_logger(), "参数k 失败原因: %s", r
55 esult.reason.c_str());
56             }
57         }
58     }
59     std::shared_ptr<SetP::Response> call_set_parameters(
60     rcl_interfaces::msg::Parameter &parameter)
61     {
62         // 1. 创建客户端等待服务上线
63         auto param_client = this->create_client<SetP>(
64         "/turtle_controller/set_parameters");
65         while (!param_client->wait_for_service(std::chrono::seconds(1)))
66         {
67             if (!rclcpp::ok())
68             {
69                 RCLCPP_ERROR(this->get_logger(), "等待服务的过程中被打断...")
70 );
71                 return nullptr;
72             }
73             RCLCPP_INFO(this->get_logger(), "等待参数设置服务端上线中");
74         }
75         // 2. 创建请求对象
76         auto request =
77         std::make_shared<SetP::Request>();
78         request->parameters.push_back(parameter);
79         // 3. 异步调用、等待并返回响应结果
80         auto future = param_client->async_send_request(request);
81         rclcpp::spin_until_future_complete(this->get_node_base_interface(), f
82 uture);
83         auto response = future.get();
84         return response;
85     }
86     void timer_callback()
87     {
88         // 1. 等待服务端上线

```

```

88     while (!patrol_client_>wait_for_service(1s))
89     {
90         // 等待时检测rclcpp的状态
91         if (!rclcpp::ok())
92         {
93             RCLCPP_ERROR(this->get_logger(), "等待服务的过程中被打断...");
94         };
95         return;
96     }
97     RCLCPP_INFO(this->get_logger(), "等待服务端上线中");
98 }
99 // 2. 构造请求的
100 auto request = std::make_shared<Patrol::Request>();
101 request->target_x = rand() % 15;
102 request->target_y = rand() % 15;
103 RCLCPP_INFO(this->get_logger(), "请求巡逻: (%f,%f)", request->target_x
, request->target_y);
104 // 3. 发送异步请求, 然后等待返回, 返回时调用回调函数
105 patrol_client_>async_send_request(
106     request,
107     [&](rclcpp::Client<Patrol>::SharedFuture result_future) -> void
108     {
109         auto response = result_future.get();
110         if (response->result == Patrol::Response::SUCCESS)
111         {
112             RCLCPP_INFO(this->get_logger(), "目标点处理成功");
113         }
114         else if (response->result == Patrol::Response::FAIL)
115         {
116             RCLCPP_INFO(this->get_logger(), "目标点处理失败");
117         }
118     });
119 }
120
121 private:
122     rclcpp::TimerBase::SharedPtr timer_;
123     rclcpp::Client<Patrol>::SharedPtr patrol_client_;
124 };
125 int main(int argc, char **argv)
126 {
127     rclcpp::init(argc, argv);
128     auto node = std::make_shared<PatrolClient>();
129     node->update_server_param_k(1.5);
130     rclcpp::spin(node);
131     rclcpp::shutdown();
132     return 0;
133 }

```

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。

```

1  import launch
2  import launch_ros
3
4  def generate_launch_description():
5      action_declare_arg_max_spped = launch.actions.DeclareLaunchArgument('l
launch_max_speed', default_value='2.0')
6
7      action_node_turtle_control = launch_ros.actions.Node(
8          package='demo_cpp_service',
9          executable="turtle_control",
10         output='screen',
11         parameters=[{'max_speed': launch.substitutions.LaunchConfiguration
(
12     'launch_max_speed', default='2.0')}]},
13     )
14     action_node_patrol_client = launch_ros.actions.Node(
15         package='demo_cpp_service',
16         executable="patrol_client",
17         output='log',
18     )
19     action_node_turtlesim_node = launch_ros.actions.Node(
20         package='turtlesim',
21         executable='turtlesim_node',
22         output='both',
23     )
24     # 合成启动描述并返回
25     launch_description = launch.LaunchDescription([
26         action_declare_arg_max_spped,
27         action_node_turtle_control,
28         action_node_patrol_client,
29         action_node_turtlesim_node
30     ])
31     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 |

```
1 install(DIRECTORY launch
2 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 |

```
1 ...
2 from glob import glob
3 setup(
4     data_files=[
5         ...
6         ('share/' + package_name+'launch', glob('launch/*.launch.py')),
7     ],
8 )
```

最后启动launch节点

▼ 启动launch节点

Plain Text |

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

5.ros2常用开发工具

坐标交换工具