14、ROS2 DDS

1、DDS简介

DDS的全称是Data Distribution Service，也就是数据分发服务，2004年由对象管理组织OMG发布和维护，是一套专门为实时系统设计的数据分发/订阅标准，最早应用于美国海军， 解决舰船复杂网络环境中大量软件升级的兼容性问题，现在已经成为强制标准。

DDS强调以数据为中心，可以提供丰富的服务质量策略，以保障数据进行实时、高效、灵活地分发，可满足各种分布式实时通信应用需求。

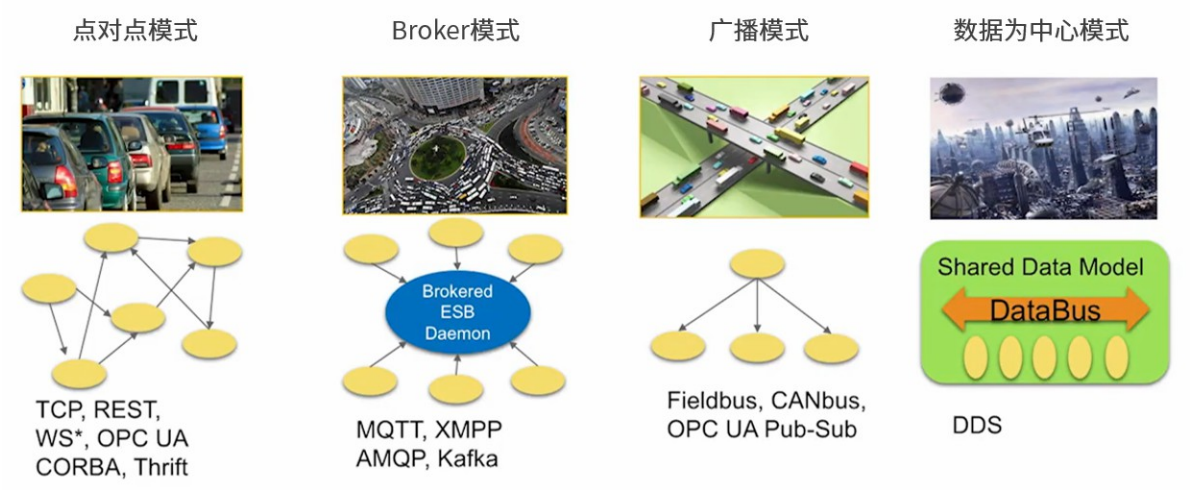
参考资料：

- Fast DDS官网文档：3.3.0
- ros2官方文档DDS进阶功能：点击跳转

2、通信模型

我们在前边课程中学习的话题、服务、动作，他们底层通信的具体实现过程，都是靠DDS来完成的，它相当于是ROS机器人系统中的神经网络。

DDS的核心是通信，能够实现通信的模型和软件框架非常多，这里我们列出常用的四种模型。



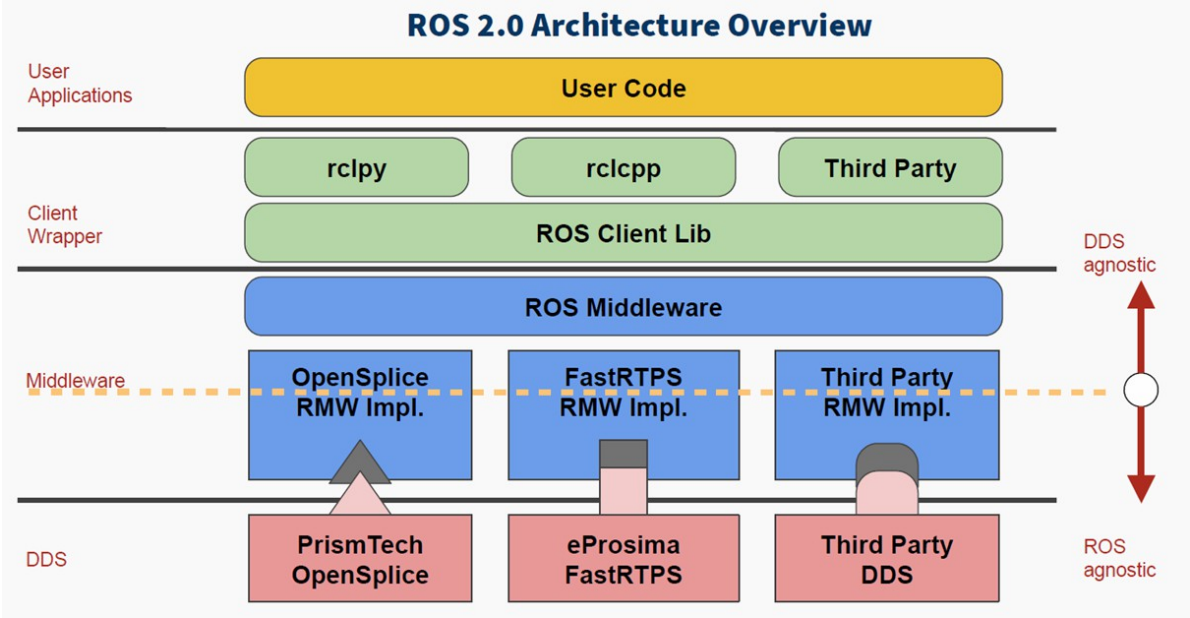
- 第一种，**点对点模型**，许多客户端连接到一个服务端，每次通信时，通信双方必须建立一条连接。当通信节点增多时，连接数也会增多。而且每个客户端都需要知道服务器的具体地址和所提供的服务，一旦服务器地址发生变化，所有客户端都会受到影响。
- 第二种，**Broker模型**，针对点对点模型进行了优化，由Broker集中处理所有人的请求，并进一步找到真正能响应该服务的角色。这样客户端就不用关心服务器的具体地址了。不过问题也很明显，Broker作为核心，它的处理速度会影响所有节点的效率，当系统规模增长到一定程度，Broker就会成为整个系统的性能瓶颈。更麻烦是，如果Broker发生异常，可能导致整个系统都无法正常运转。之前的ROS1系统，使用的就是类似这样的架构。
- 第三种，**广播模型**，所有节点都可以在通道上广播消息，并且节点都可以收到消息。这个模型解决了服务器地址的问题，而且通信双方也不用单独建立连接，但是广播通道上的消息太多了，所有节点都必须关心每条消息，其实很多是和自己没有关系的。
- 第四种，就是**以数据为中心的DDS模型**了，这种模型与广播模型有些类似，所有节点都可以在DataBus上发布和订阅消息。但它的先进之处在于，通信中包含了很多并行的通路，每个节点可以

只关心自己感兴趣的消息，忽略不感兴趣的消息，有点像是一个旋转火锅，各种好吃的都在这个DataBus传送，我们只需要拿自己想吃的就行，其他的和我们没有关系。

可见，在这几种通信模型中，DDS的优势更加明显。

3、DDS在ROS2中的应用

DDS在ROS2系统中的位置至关重要，所有上层建设都建立在DDS之上。在这个ROS2的架构图中，蓝色和红色部分就是DDS。



在ROS的四大组成部分中，由于DDS的加入，大大提高了分布式通信系统的综合能力，这样我们在开发机器人的过程中，就不需要纠结通信的问题，可以把更多时间放在其他部分的应用开发上。

4、质量服务策略QoS

DDS中的基本结构是Domain，Domain将各个应用程序绑定在一起进行通信，回忆下之前我们配置树莓派和电脑通信的时候，配置的那个DOMAIN ID，就是对全局数据空间的分组定义，只有处于同一个DOMAIN小组中的节点才能互相通信。这样可以避免无用数据占用的资源。

DDS中另外一个重要特性就是质量服务策略：QoS。

QoS是一种网络传输策略，应用程序指定所需要的网络传输质量行为，QoS服务实现这种行为要求，尽可能地满足客户对通信质量的需求，可以理解为数据提供者和接收者之间的合约。

策略如下：

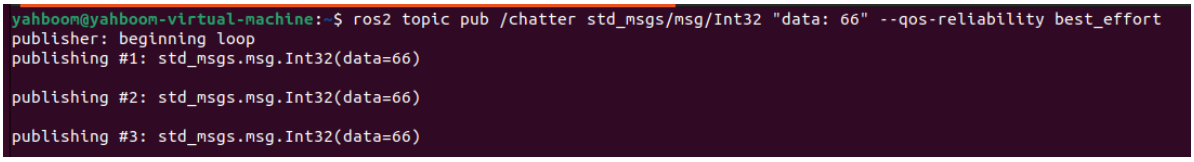
- **DEADLINE**策略，表示通信数据必须要在每次截止时间内完成一次通信；
- **HISTORY**策略，表示针对历史数据的一个缓存大小；
- **RELIABILITY**策略，表示数据通信的模式，配置成BEST_EFFORT，就是尽力传输模式，网络情况不好的时候，也要保证数据流畅，此时可能会导致数据丢失，配置成RELIABLE，就是可信赖模式，可以在通信中尽量保证图像的完整性，我们可以根据应用功能场景选择合适的通信模式；
- **DURABILITY**策略，可以配置针对晚加入的节点，也保证有一定的历史数据发送过去，可以让新节点快速适应系统。

5. 测试案例

5.1 案例1—通过命令行配置DDS

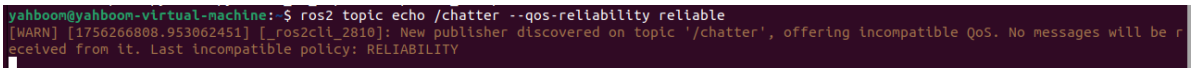
- 打开第一个终端，使用以下命令发布话题：

```
ros2 topic pub /chatter std_msgs/msg/Int32 "data: 66" --qos-reliability
best_effort
```



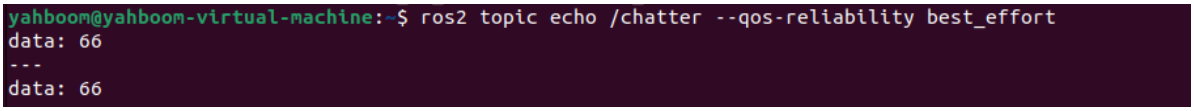
- 再次打开一个终端，使用不同的Qos去打印话题，如果使用的Qos策略与发布方不同，则会出现警告信息不能正常接收到话题数据：

```
ros2 topic echo /chatter --qos-reliability reliable
```



- 我们使用和话题发布方一样的Qos策略即可接收到话题数据

```
ros2 topic echo /chatter --qos-reliability best_effort
```



5.2 案例2—编写话题节点配置Qos服务策略

- 使用以下命令新建功能包

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

- 新建一个dds_controller_pub.py文件，作为话题通信的发布方，填入一下内容：

```
import rclpy
from rclpy.node import Node
from std_msgs.msg import String
# 导入QoS相关类
from rclpy.qos import QoSProfile, QoSReliabilityPolicy, QoSHistoryPolicy

class ControllerPublisher(Node):
    def __init__(self, name):
        super().__init__(name)
        # 1. 配置QoS策略：可靠传输，保留最后1条历史数据
        self.qos_profile = QoSProfile(
            reliability=QoSReliabilityPolicy.RELIABLE,  # 可靠传输（重传丢失数据）
            history=QoSHistoryPolicy.KEEP_LAST,         # 保留最后N条数据
            depth=1                                     # 保留1条历史数据
        )
        # 2. 创建发布者：话题名/robot_cmd，消息类型String，QoS策略
        self.publisher = self.create_publisher(
```

```
            String,
            "/robot_cmd",
            self.qos_profile
        )
        # 3. 创建定时器：每秒发送一次指令
        self.timer = self.create_timer(1.0, self.timer_callback)
        self.cmd_list = ["forward", "backward", "stop"]  # 指令列表
        self.cmd_index = 0  # 指令索引，循环切换


    def timer_callback(self):
        # 循环切换指令（前进→后退→停止→前进...）
        current_cmd = self.cmd_list[self.cmd_index % 3]
        # 创建消息并填充数据
        msg = String()
        msg.data = current_cmd
        # 发布消息
        self.publisher.publish(msg)
        # 打印日志（显示发布的指令）
        self.get_logger().info(f"发布控制指令：{msg.data}")
        # 更新指令索引
        self.cmd_index += 1


def main(args=None):
    # 初始化ROS2
    rclpy.init(args=args)
    # 创建发布者节点
    node = ControllerPublisher("robot_controller_pub")
    # 循环运行节点
    rclpy.spin(node)
    # 销毁节点并关闭ROS2
    node.destroy_node()
    rclpy.shutdown()


if __name__ == "__main__":
    main()
```

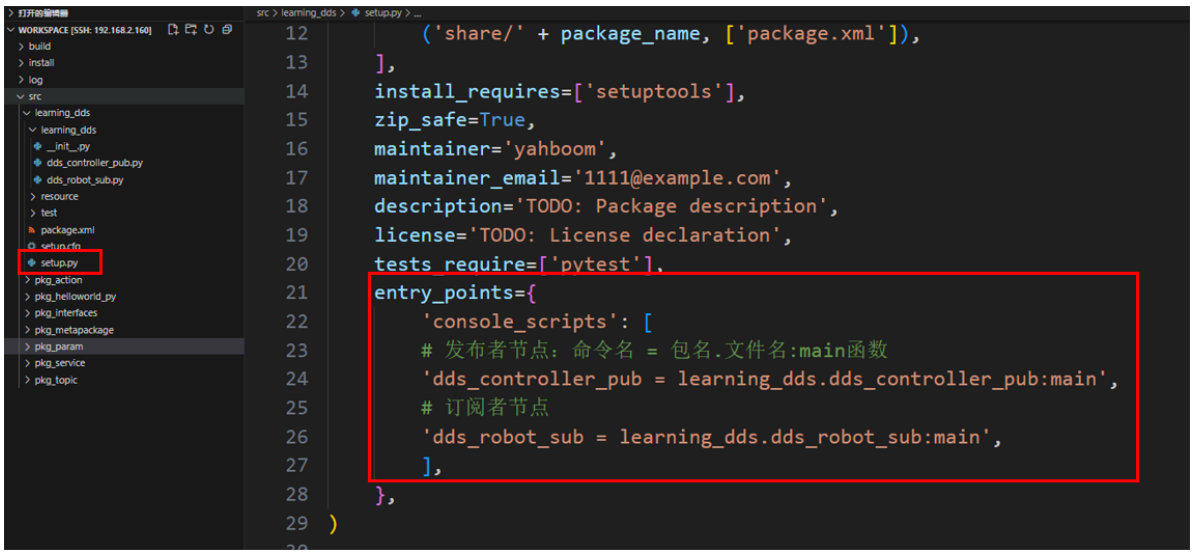
- 订阅方代码实现，新建一个dds_robot_sub.py文件，填入下方订阅方实现代码

```
import rclpy
from rclpy.node import Node
from std_msgs.msg import String
from rclpy.qos import QoSProfile, QoSReliabilityPolicy, QoSHistoryPolicy

class RobotSubscriber(Node):
    def __init__(self, name):
        super().__init__(name)
        # 1. 配置与发布者兼容的QoS策略
        self.qos_profile = QoSProfile(
            reliability=QoSReliabilityPolicy.BEST_EFFORT,
            history=QoSHistoryPolicy.KEEP_LAST,
            depth=1
        )
```

```
        # 2. 创建订阅者：话题名/robot_cmd，回调函数，QoS策略
        self.subscription = self.create_subscription(
            String,
            "/robot_cmd",
            self.cmd_callback,  # 接收到数据后执行的回调函数
            self.qos_profile
        )

    def cmd_callback(self, msg):
        # 回调函数：处理接收到的指令
        self.get_logger().info(f"接收控制指令：{msg.data} → 执行对应动作")


def main(args=None):
    rclpy.init(args=args)
    node = RobotSubscriber("robot_subscriber")
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()


if __name__ == "__main__":
    main()
```

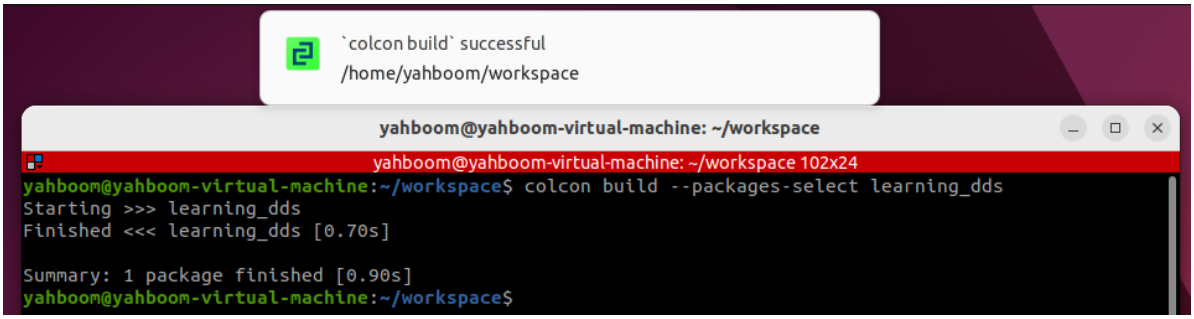
- 配置编译文件（setup.py）

```
entry_points={
    'console_scripts': [
        # 发布者节点：命令名 = 包名.文件名:main函数
        'dds_controller_pub = learning_dds.dds_controller_pub:main',
        # 订阅者节点
        'dds_robot_sub = learning_dds.dds_robot_sub:main',
    ],
},
```



- 在工作空间目录下打开终端，编译功能包

```
colcon build --packages-select learning_dds
```



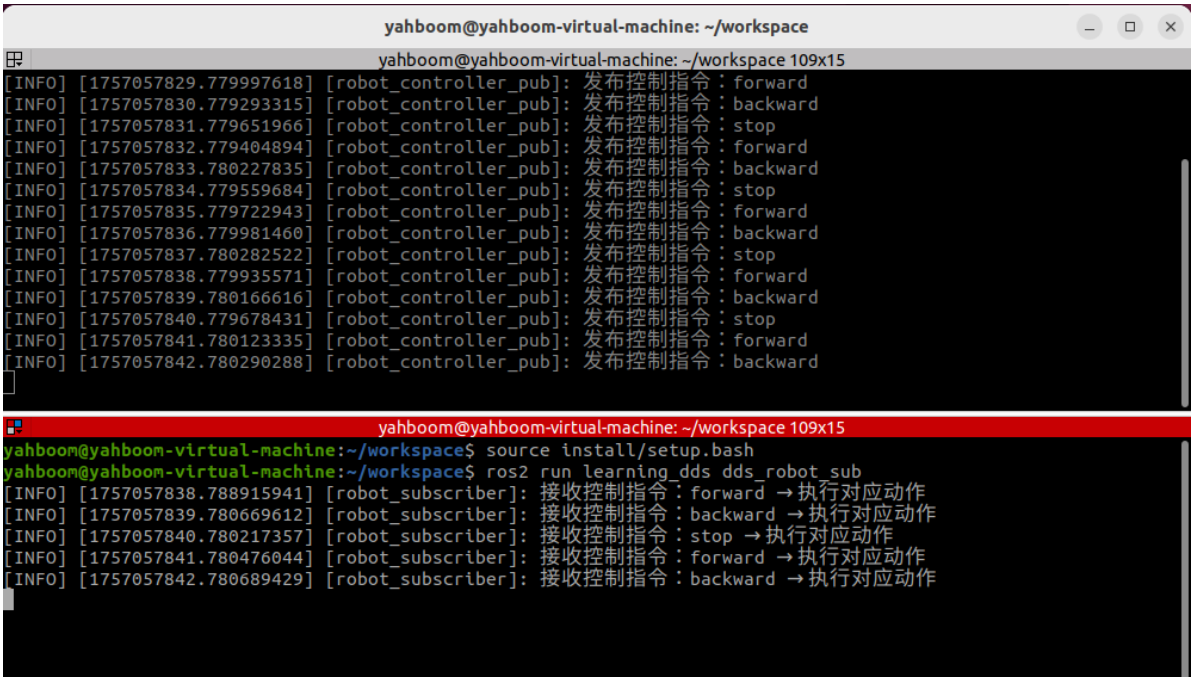
- 设置环境变量，每次重新编译后都需要设置环境变量

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

- 运行发布方和订阅方节点

```
ros2 run learning_dds dds_controller_pub
```

```
ros2 run learning_dds dds_robot_sub
```



机器人开发进阶：

- 通常情况下，话题通信的发布方和订阅方需要保持一样的Qos通信策略，避免一些隐性的通信层问题；
- Qos通信策略的设置要根据实际的应用场景来选择；
- 如果应用场景涉及：无人机通信、加密通信、实时通信等对通信层有特殊的要求场景，在本节教程简介中找到Fast-DDS的官方手册，对DDS功能有更详尽的说明；

What is *Fast DDS*?

eProsima Fast DDS is a C++ implementation of the DDS (Data Distribution Service) Specification, a protocol defined by the Object Management Group (OMG). The *eProsima Fast DDS* library provides both an Application Programming Interface (API) and a communication protocol that deploy a Data-Centric Publisher-Subscriber (DCPS) model, with the purpose of establishing efficient and reliable information distribution among Real-Time Systems.

Commercial support

Looking for commercial support? Write us to info@eprosima.com

Find more about us at eProsima's webpage.

Key features