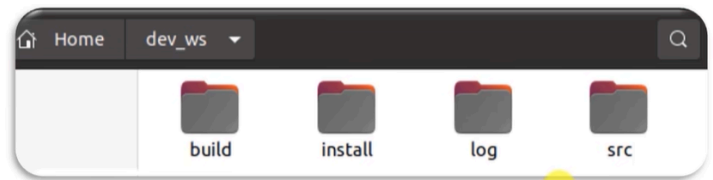


一、工作空间

1. 工作空间 (workspace)

工作空间 (workspace) 是一个存放项目开发相关文件的文件夹；是开发过程的大本营。

- **src**: 代码空间 (Source Space)
- **install**: 安装空间 (Install Space)
- **build**: 编译空间 (Build Space)
- **log**: 日志空间 (Log Space)



ROS2的工作空间结构

编译、依赖安装等工作都要在工作空间根目录 (比如这里的/dev_ws) 进行

2. rosdep——自动安装功能包所需依赖的工具

rosdep 是 ROS (机器人操作系统) 中的一个工具, 用于安装系统依赖项。ROS 是一个用于构建机器人软件的开源框架, 它依赖于许多系统级别的软件包。**rosdep** 的作用是帮助用户轻松地安装这些系统依赖项。

使用 **rosdep** 有一些步骤:

1. 初始化: 在使用 **rosdep** 之前, 你需要初始化 **rosdep** 数据库。这可以通过以下命令完成:

```
sudo rosdep init
rosdep update
```

这将初始化 **rosdep** 并从网络上获取最新的依赖项数据。

2. 安装依赖项: 一旦初始化完成, 你可以使用 **rosdep** 安装系统依赖项。例如:

```
rosdep install --from-paths your_package_directory --ignore-src
```

这将尝试安装你的 ROS 软件包所需的所有系统依赖项。

- 可不可以不 **init** 和 **update**: 在某些情况下, 如果你知道系统中已经存在所需的依赖项, 你可能不需要运行 **rosdep init** 和 **rosdep update**。通常建议在使用 **rosdep** 之前运行这两个命令, 如果你不是用 **rosdep** 则可以用。

(上述 **rosdep** 均替换为 **rosdep**)

1. 针对 ROS 2 Jazzy, 指令应该怎么改?

您在图中看到的指令是: `rosdep install -i --from-path src --rosdistro humble -y`

在 ROS 2 Jazzy 中, 您只需要将 `--rosdistro` 后面的参数改为 `jazzy` 即可。对应的指令为:

```
bash
rosdep install -i --from-path src --rosdistro jazzy -y
```

指令参数解释:

- `install`: 安装依赖。
- `-i` (或 `--ignore-src`): 忽略工作空间 `src` 目录下已经存在的包, 只安装系统缺失的依赖。
- `--from-path src`: 指定从当前目录下的 `src` 文件夹中寻找 `package.xml` 文件来解析依赖。
- `--rosdistro jazzy`: **关键点**。明确告诉工具你当前的 ROS 版本是 Jazzy。
- `-y`: 自动确认, 不再询问是否安装。

3. colcon 编译命令

在工作空间根目录下运行 `colcon build` 命令。编译完后工作空间就从只有 `src` 文件夹到变成4个文件夹了。

注意: 当更改 `src` 中的源代码后, 要再 `colcon build`, 也即再编译后 `install` 里边的程序才会更改。否则 `ros2 run` 的还是老程序。

4. 配置环境变量

配置环境变量后, 运行 `ros2 run` 之类的命令时系统才能找到你的源代码在哪

p. s. 在 Linux 中, `source` 命令的作用是: 在当前终端窗口中运行指定的脚本, 并让脚本里的设置 (如变量、路径) 立即生效。

方法一: `colcon` 编译后, 在工作空间根目录下运行 `source install/local_setup.sh` 命令。这种方法需要在每次打开新终端的时候再写一遍。(因为该命令是在当前终端把指定的包路径临时添加到系统的“搜索名单”里)

方法二: 修改系统根目录下的隐藏文件 `.bashrc`, 在该文件底部添加 `source ~/dev_ws/install/setup.bash` 命令, 这样以后每次打开新终端也不用再写。(因为该文件中的这些语句会在打开终端后自动执行。最好在该文件中只放最常用的工作空间, 避免环境变量混乱)

注: `local_setup.sh` vs `setup.bash`:

`local_setup.sh`: 只加载当前这工作空间的东西。
`setup.bash`: 会加载当前工作空间**以及**它依赖的所有上层空间 (比如 ROS 系统本身)。

建议：在 `.bashrc` 里通常写 `source ~/dev_ws/install/setup.bash`。

二、功能包

1. 创建功能包

• 创建功能包



```
$ ros2 pkg create --build-type <build-type> <package_name>
```

创建功能包

```
$ cd ~/dev_ws/src  
$ ros2 pkg create --build-type ament_cmake learning_pkg_c # C++  
$ ros2 pkg create --build-type ament_python learning_pkg_python # Python
```

编译功能包

```
$ cd ~/dev_ws/src  
$ colcon build # 编译工作空间所有功能包  
$ source install/local_setup.bash
```

2. 功能包文件说明

`package.xml` —— 里边把功能包的依赖写全，`rosdep`工具就看的这个。

FAQ:

2. 是自动写，还是手动写？

答案是：一半自动，一半手动。

- **自动部分**：当你使用命令创建包时，工具会帮你生成模板。例如：`ros2 pkg create --build-type ament_python my_pkg --dependencies rclpy` 这条命令会自动在 `package.xml` 里写好 `rclpy` 的依赖。
- **手动部分**：在开发过程中，如果你突然想用一个新的库（比如你要处理图像，需要 `sensor_msgs`），你**必须手动**打开 `package.xml` 把这个依赖加进去。它不会在你写代码时自动感知并添加。

3. 写的格式是什么样的？

针对不同的用途，有几种常见的标签格式：

① 最通用的格式：<depend>

如果你不确定，用这个准没错。它表示编译时和运行时都需要这个包。

```
● XML ≡ □  
  
<depend>rclcpp</depend>  
<depend>sensor_msgs</depend>
```

② 细分格式（进阶）：

- <build_depend>：仅在编译代码时需要（例如某些头文件）。
- <exec_depend>：编译时不需要，但程序运行（Run）时需要。
- <test_depend>：只在跑自动化测试时需要。

4. 重点：里面写的是“Python库名”吗？

不完全是。这里是新手最容易搞混的地方：

在 `package.xml` 里写的名字，必须是“**rosdep 定义过的键名**”。

- **如果是 ROS 官方包**：直接写包名。例如你想用 Python 版的 ROS 接口，写 `rclpy`；想用雷达数据格式，写 `sensor_msgs`。
- **如果是标准的 Python 库（如 numpy, opencv）**：你不能随便写。你需要查一下对应的系统包名。
 - 想用 `numpy`？在 `package.xml` 里写：`<depend>python3-numpy</depend>`
 - 想用 `opencv`？在 `package.xml` 里写：`<depend>python3-opencv</depend>`
 - 想用 `requests`？在 `package.xml` 里写：`<depend>python3-requests</depend>`

怎么查该写什么名字？ 你可以去 [ROS Index](#) 搜索，或者查看 [roscat](#) 的官方数据库。

(或者直接问AI)

==setup.py ==—— 后面开发的时候要记得往里写程序入口
例：

源代码文件：

```
node_helloworld.py ×
learning_node > learning_node > node_helloworld.py > ...
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  """
5  @作者：古月居(www.guyuehome.com)
6  @说明：ROS2节点示例-发布“Hello World”日志信息，使用面向过程的实现方式
7  """
8
9  import rclpy                                # ROS2 Python接口库
10 from rclpy.node import Node                 # ROS2 节点类
11 import time
12
13 def main(args=None):                         # ROS2节点主入口main函数
14     rclpy.init(args=args)                   # ROS2 Python接口初始化
15     node = Node("node_helloworld")          # 创建ROS2节点对象并进行初始化
16
17     while rclpy.ok():                         # ROS2系统是否正常运行
18         node.get_logger().info("Hello World") # ROS2日志输出
19         time.sleep(0.5)                       # 休眠控制循环时间
20
21     node.destroy_node()                       # 销毁节点对象
22     rclpy.shutdown()                          # 关闭ROS2 Python接口
23
```

setup.py：

```
资源管理器
ROS2_21_TUTORIALS
├── learning_action
├── learning_gazebo
├── learning_interface
├── learning_launch
├── learning_node
│   ├── learning_node
│   │   ├── __init__.py
│   │   ├── apple.jpg
│   │   ├── node_helloworld_class.py
│   │   ├── node_helloworld.py
│   │   ├── node_object_webcam.py
│   │   └── node_object.py
│   ├── resource
│   ├── test
│   ├── package.xml
│   ├── setup.cfg
│   └── setup.py
├── learning_parameter
├── learning_pkg_c
├── learning_pkg_python
├── learning_qos
├── learning_service
├── learning_tf
├── learning_topic
└── learning_urdf

node_helloworld.py  setup.py ×
learning_node > setup.py > ...
1  from setuptools import setup
2
3  package_name = 'learning_node'
4
5  setup(
6      name=package_name,
7      version='0.0.0',
8      packages=[package_name],
9      data_files=[
10         ('share/ament_index/resource_index/packages',
11          ['resource/' + package_name]),
12         ('share/' + package_name, ['package.xml']),
13     ],
14     install_requires=['setuptools'],
15     zip_safe=True,
16     maintainer='Hu Chunxu',
17     maintainer_email='huchunxu@guyuehome.com',
18     description='TODO: Package description',
19     license='TODO: License declaration',
20     tests_require=['pytest'],
21     entry_points={
22         'console_scripts': [
23             'node_helloworld = learning_node.node_helloworld:main',
24             'node_helloworld_class = learning_node.node_helloworld_class:main',
25             'node_object = learning_node.node_object:main',
26             'node_object_webcam = learning_node.node_object_webcam:main',
27         ],
28     },
29 )
```

(路径：函数)

三、节点

- 执行具体任务的进程
- 独立运行的可执行文件
- 可使用不同的编程语言
- 可分布式运行在不同主机
- 通过节点名称进行管理

模块化 解耦

1. 节点程序范式流程:

- 编程接口初始化
- 创建节点并初始化
- 实现节点功能
- 销毁节点并关闭接口

后续更多以**面向对象**的方式编写节点程序

提醒：所有python程序要记得在功能包里的setup.py里写程序接口，这样以后才能用ros2 run运行它。

2. ROS2提供的标准节点

ROS2提供了一些标准功能包（节点），安装后就可以直接使用。比如相机驱动节点，只要服从USB协议就都能用。

• 实现一个更为复杂的话题通信



```
humble-pc:~$ ros2 run usb_cam usb_cam_node_exe
[INFO] [1649825135.943185297] [usb_cam]: camera_name value: default_cam
[INFO] [1649825135.943217151] [usb_cam]: file format is omxlinux
[INFO] [1649825135.947018509] [usb_cam]: using default calibration URL
[INFO] [1649825135.947372158] [usb_cam]: camera calibration URL: file:///home/hcx/.ros/camera_info/default_cam.yaml
[INFO] [1649825135.947372158] [usb_cam]: camera calibration file (/home/hcx/.ros/camera_info/default_cam.yaml) does not exist
[WARN] [1649825135.947372158] [usb_cam]: camera calibration file (/home/hcx/.ros/camera_info/default_cam.yaml) does not exist
[WARN] [1649825135.947372158] [usb_cam]: Starting 'default_cam' (dev/video0) at
someone via mmp: (0/0) 48: 32 FPS
[INFO] [1649825135.986232002] [usb_cam]: Capability flag: 0x1000
[INFO] [1649825135.986232002] [usb_cam]: Set framerate to be 30
[INFO] [1649825135.989274624] [usb_cam]: TMLs Cameras supported Formats:
[INFO] [1649825135.989274624] [usb_cam]: VOV: 4:2:2[Index: 0, Type: 1, Flags: 0]
PixelFormat: S095559
[INFO] [1649825135.992334044] [usb_cam]: width: 640 x height: 480
[INFO] [1649825135.992334044] [usb_cam]: 1.5 / 30
[INFO] [1649825135.992334044] [usb_cam]: width: 320 x height: 240
[INFO] [1649825135.992334044] [usb_cam]: 1.5 / 30
[INFO] [1649825135.992334044] [usb_cam]: width: 320 x height: 240
[INFO] [1649825135.992334044] [usb_cam]: 1.5 / 30
[INFO] [1649825135.992334044] [usb_cam]: width: 1280 x height: 720
[INFO] [1649825135.992334044] [usb_cam]: 1.5 / 30
[INFO] [1649825135.992334044] [usb_cam]: Motion-SPEC[Index: 1, Type: 1, Flags: 0]
```

```
humble-pc:~$ ros2 run learning_topic topic_webcam_sub
[INFO] [1649825151.838578154] [topic_webcam_sub]: Receiving video frame
[INFO] [1649825152.005526196] [topic_webcam_sub]: Receiving video frame
[INFO] [1649825152.032166927] [topic_webcam_sub]: Receiving video frame
[INFO] [1649825152.046512641] [topic_webcam_sub]: Receiving video frame
[INFO] [1649825152.060998825] [topic_webcam_sub]: Receiving video frame
[INFO] [1649825152.213221882] [topic_webcam_sub]: Receiving video frame
[INFO] [1649825152.228328339] [topic_webcam_sub]: Receiving video frame
[INFO] [1649825152.305498019] [topic_webcam_sub]: Receiving video frame
[INFO] [1649825152.405414056] [topic_webcam_sub]: Receiving video frame
[INFO] [1649825152.506596999] [topic_webcam_sub]: Receiving video frame
[INFO] [1649825152.695681960] [topic_webcam_sub]: Receiving video frame
[INFO] [1649825152.712913606] [topic_webcam_sub]: Receiving video frame
[INFO] [1649825152.805608390] [topic_webcam_sub]: Receiving video frame
[INFO] [1649825152.905764674] [topic_webcam_sub]: Receiving video frame
[INFO] [1649825153.013055999] [topic_webcam_sub]: Receiving video frame
[INFO] [1649825153.105981266] [topic_webcam_sub]: Receiving video frame
[INFO] [1649825153.205860840] [topic_webcam_sub]: Receiving video frame
[INFO] [1649825153.312936970] [topic_webcam_sub]: Receiving video frame
```

话题通信

```
$ sudo apt install ros-humble-usb-cam
$ ros2 run usb_cam usb_cam_node_exe
$ ros2 run learning_topic topic_webcam_sub
```

(话题名字、消息结构一样就行)

四、话题

话题：节点间传递数据的桥梁

- 发布/订阅模型
- 订阅者或发布者可以不唯一
- 异步通信机制
- .msg文件定义通信的消息结构

单向传输，适合周期性的信息传递

1.publisher

示例程序：

```
@作者：古月居(www.guyuehome.com)
@说明：ROS2话题示例-发布“Hello World”话题
"""

import rclpy                                     # ROS2 Python接口库
from rclpy.node import Node                       # ROS2 节点类
from std_msgs.msg import String                  # 字符串消息类型

"""
创建一个发布者节点
"""

class PublisherNode(Node):

    def __init__(self, name):
        super().__init__(name)                   # ROS2节点父类初始化
        self.pub = self.create_publisher(String, "chatter", 10) # 创建发布者对象（消息类型、话题名、队列长度）
        self.timer = self.create_timer(0.5, self.timer_callback) # 创建一个定时器（单位为秒的周期，定时执行的回调函数）

    def timer_callback(self):
        msg = String()                            # 创建一个String类型的消息对象
        msg.data = 'Hello World'                  # 填充消息对象中的消息数据
        self.pub.publish(msg)                     # 发布话题消息
        self.get_logger().info('Publishing: "%s"' % msg.data) # 输出日志信息，提示已经完成话题发布

def main(args=None):
    # ROS2节点主入口main函数
    rclpy.init(args=args)                         # ROS2 Python接口初始化
    node = PublisherNode("topic_helloworld_pub")  # 创建ROS2节点对象并进行初始化
    rclpy.spin(node)                              # 循环等待ROS2退出
    node.destroy_node()                           # 销毁节点对象
    rclpy.shutdown()                              # 关闭ROS2 Python接口
```

程序流程：

- 编程接口初始化
- 创建节点并初始化
- 创建发布者对象
- 创建并填充话题消息
- 发布话题消息
- 销毁节点并关闭接口

2.subscriber

示例程序：

```
"""
@作者：古月居(www.guyuehome.com)
@说明：ROS2话题示例-订阅"Hello World"话题消息
"""

import rclpy # ROS2 Python接口库
from rclpy.node import Node # ROS2 节点类
from std_msgs.msg import String # ROS2标准定义的String消息

"""
创建一个订阅者节点
"""
class SubscriberNode(Node):

    def __init__(self, name): # ROS2节点父类初始化
        super().__init__(name)
        self.sub = self.create_subscription(\ # 创建订阅者对象(消息类型、话题名、订阅者回调函数、队列长度)
            String, "chatter", self.listener_callback, 10)

    def listener_callback(self, msg): # 创建回调函数,执行收到话题消息后对数据的处理
        self.get_logger().info('I heard: "%s"' % msg.data) # 输出日志信息,提示订阅收到的话题消息

def main(args=None): # ROS2节点主入口main函数
    rclpy.init(args=args) # ROS2 Python接口初始化
    node = SubscriberNode("topic_helloworld_sub") # 创建ROS2节点对象并进行初始化
    rclpy.spin(node) # 循环等待ROS2退出
    node.destroy_node() # 销毁节点对象
    rclpy.shutdown() # 关闭ROS2 Python接口
```

程序流程：

- 编程接口初始化
- 创建节点并初始化
- 创建订阅者对象
- 回调函数处理话题数据
- 销毁节点并关闭接口

订阅者回调函数：执行收到话题信息后对消息的处理

FAQ:

1. listener_callback是不是只能写msg参数呢（除了self）？

非常好的问题！你已经抓住了关键点：**ROS 2 在触发回调函数时，会自动传入某些参数。**那是不是只能写 `self` 和 `msg` 呢？我们来详细解释。

✓ 简短回答：

对于普通的订阅者（Subscriber）回调函数，ROS 2 只会自动传入一个消息对象（即 `msg` ），所以你的回调方法签名通常只能是 `listener_callback(self, msg)` —— 不能随意加更多参数。

但！有办法“变通”地传递额外数据，我们后面会讲。

2. 添加额外参数的方式？

方法 1: 使用 闭包 (Closure) 或 lambda 表达式

python

编辑 复制 刷新 向上

```
1 def __init__(self):
2     super().__init__('my_subscriber')
3     self.extra_info = "Hello from init!"
4
5     self.sub = self.create_subscription(
6         String,
7         'chatter',
8         lambda msg: self.listener_callback(msg, self.extra_info),
9         10
10    )
11
12 def listener_callback(self, msg, extra):
13     self.get_logger().info(f"Msg: {msg.data}, Extra: {extra}")
```

✔ 优点: 简洁, 适合简单场景。

⚠ 注意: 避免在 lambda 中捕获易变变量 (如循环中的 i)。

方法 2: 把额外数据存为 类的属性 (instance variable)

这是最常见、最清晰的方式!

python

编辑 复制 刷新 向上

```
1 def __init__(self):
2     super().__init__('my_subscriber')
3     self.my_config = {"threshold": 0.5, "name": "sensor1"}
4
5     self.sub = self.create_subscription(
6         String,
7         'chatter',
8         self.listener_callback,
9         10
10    )
11
12 def listener_callback(self, msg):
13     if self.my_config["name"] == "sensor1":
14         self.get_logger().info(f"Processing: {msg.data}")
```

✔ 优点:

- 代码清晰
- 符合面向对象设计
- 可在多个回调中共享状态

方法 3: 使用 `functools.partial` (高级技巧)

```
python
1 from functools import partial
2
3 def __init__(self):
4     super().__init__('my_subscriber')
5     extra_data = "custom info"
6     self.sub = self.create_subscription(
7         String,
8         'chatter',
9         partial(self.listener_callback, extra=extra_data),
10        10
11    )
12
13 def listener_callback(self, msg, extra):
14     self.get_logger().info(f"{msg.data} + {extra}")
```

⚠ 注意: `partial` 会改变函数签名, 但 ROS 2 调用时仍然只传 `msg`, 而 `partial` 会自动补上 `extra`。

3. 为什么说回调函数要轻量 (尤其是高频话题)?

原因	后果	解决方案
阻塞消息处理	系统卡顿、数据丢失	回调只存数据, 另起线程处理
破坏实时性	控制失效、机器人异常	保证回调 < 1ms~10ms
引发竞态条件	数据错乱、崩溃	避免共享状态, 或加锁
队列溢出	重要消息被丢弃	提高处理速度或增大队列 (治标不治本)

🔴 **黄金法则: 回调函数应该像“邮递员”——快速收信, 立刻转交, 绝不拆开慢慢读!**

所以会推荐回调函数只做“接受+转发”, 消息处理由另一个线程去做; 或者高级用法会有精细控制并发等。

p.s. 一个小工具 rqt_graph，用于可视化节点关系

The image shows a terminal window with ROS log output and an rqt_graph window. The terminal output includes:

```
[INFO] [1653460594.169802858] [topic_webcam] 10 FPS
[INFO] [1653460594.268978905] [topic_webcam] [INFO] [1653460430.217016708] [usb_cam]: Capability flag: 0x1000
[INFO] [1653460594.372572663] [topic_webcam] [INFO] [1653460430.246275562] [usb_cam]: Set framerate to be 10
[INFO] [1653460594.468672015] [topic_webcam] [INFO] [1653460430.252371159] [usb_cam]: This Cameras Supported Formats:
[INFO] [1653460594.571191596] [topic_webcam] [INFO] [1653460430.252530329] [usb_cam]: YUYV 4:2:2[Index: 0, Type: 1, Flags: 0, PixelFormat: 56595559]
[INFO] [1653460594.669298507] [topic_webcam] [INFO] [1653460430.252557220] [usb_cam]: width: 640 x height: 480
[INFO] [1653460594.769265125] [topic_webcam] [INFO] [1653460430.252577416] [usb_cam]: 1 1 / 30
[INFO] [1653460594.870020839] [topic_webcam] [INFO] [1653460430.252597054] [usb_cam]: width: 160 x height: 120
[INFO] [1653460594.970354379] [topic_webcam] [INFO] [1653460430.252614675] [usb_cam]: 1 1 / 30
[INFO] [1653460595.069419000] [topic_webcam] [INFO] [1653460430.252632160] [usb_cam]: width: 320 x height: 240
[INFO] [1653460595.169002300] [topic_webcam] [INFO] [1653460430.252649645] [usb_cam]: width: 160 x height: 120
[INFO] [1653460595.268764100] [topic_webcam] [INFO] [1653460430.252667130] [usb_cam]: width: 320 x height: 240
[INFO] [1653460595.369256200] [topic_webcam] [INFO] [1653460430.252684615] [usb_cam]: width: 160 x height: 120
[INFO] [1653460595.469267400] [topic_webcam] [INFO] [1653460430.252702100] [usb_cam]: width: 320 x height: 240
[INFO] [1653460595.569070000] [topic_webcam] [INFO] [1653460430.252719585] [usb_cam]: width: 160 x height: 120
[INFO] [1653460595.669377000] [topic_webcam] [INFO] [1653460430.252737070] [usb_cam]: width: 320 x height: 240
[INFO] [1653460595.767686000] [topic_webcam] [INFO] [1653460430.252754555] [usb_cam]: width: 160 x height: 120
[INFO] [1653460595.867801000] [topic_webcam] [INFO] [1653460430.252772040] [usb_cam]: width: 320 x height: 240
[INFO] [1653460595.969669000] [topic_webcam] [INFO] [1653460430.252789525] [usb_cam]: width: 160 x height: 120
[INFO] [1653460596.069445000] [topic_webcam] [INFO] [1653460430.252807010] [usb_cam]: width: 320 x height: 240
[INFO] [1653460596.169319000] [topic_webcam] [INFO] [1653460430.252824495] [usb_cam]: width: 160 x height: 120
[INFO] [1653460596.268898000] [topic_webcam] [INFO] [1653460430.252841980] [usb_cam]: width: 320 x height: 240
[INFO] [1653460596.368752000] [topic_webcam] [INFO] [1653460430.252859465] [usb_cam]: width: 160 x height: 120
```

The rqt_graph window displays a graph with two nodes: /usb_cam (blue oval) and /topic_webcam_sub (green oval). A red arrow points from /usb_cam to /topic_webcam_sub, labeled /image_raw. The window also shows a search bar, a group selector set to 2, and various filter options like Namespaces, Actions, Images, Highlight, and Fit.

```
hcx@hcx-pc: ~$ rqt
rqt_bag  rqt_graph
hcx@hcx-pc: ~$ rqt_graph
```