# ROS2入门

目录

[一、 ROS2入门学习 1](#_Toc32173)

[1. Linux基础 1](#_Toc18901)

[（1） 指令基础 1](#_Toc14055)

[（2） 基础工具的使用 3](#_Toc11650)

[二、 ROS2的简介及安装 4](#_Toc14571)

[1. ROS简介 4](#_Toc3830)

[2. ROS2与ROS的区别 5](#_Toc4659)

[3. ROS2安装 7](#_Toc28112)

[三、 ROS2基础学习 8](#_Toc2732)

[1. 工作空间（workplace） 8](#_Toc17901)

[2. 节点：机器人细胞 10](#_Toc15324)

[3. 话题（Topic） 11](#_Toc2928)

[4. 服务（service）通信 12](#_Toc20285)

[5. 通信接口（interface） 13](#_Toc22566)

[6. 动作（action） 14](#_Toc15527)

[7. 参数（parameter） 15](#_Toc23776)

[8. 分布式通信 15](#_Toc28851)

[9. 数据分发服务（DDS） 15](#_Toc32435)

[四、 ROS2进阶和应用 18](#_Toc17272)

[1. Launch 18](#_Toc1866)

[2. URDF（Unified Robot Description Format）统一机器人描述格式 20](#_Toc10786)

[3.Gazebo 21](#_Toc18599)

[4. Rviz2 22](#_Toc8737)

## ROS2入门学习

在ROS2的学习及其使用过程中，会遇到很多问题。举个例子，你想写一个简单的“打印Hello world”的程序，不会去编写或者是调试运行。我的建议是：要学习一些基础的指令，然后在学习ROS2的基础上要运用到什么指令再去学习。

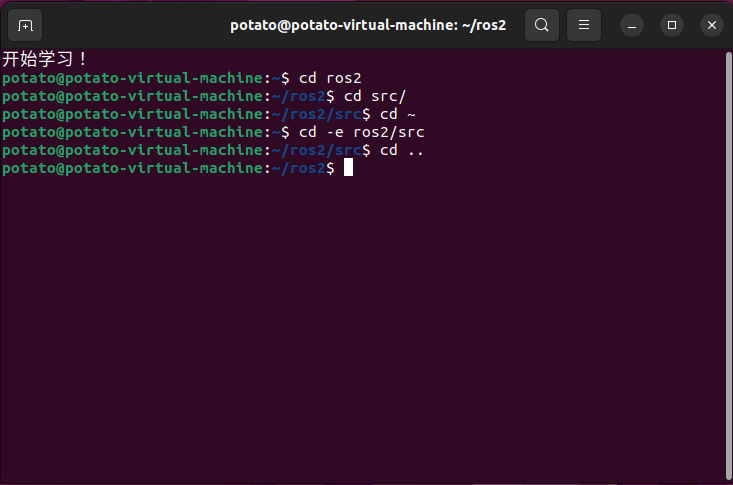
实用小技巧：终端中可以使用tab键进行自动补全，可以节约很多时间。在进行指令输入的时候，可以使用数字键盘旁边的上下键进行查看前/后一条指令，也可节约很多时间。

1. Linux基础
2. 指令基础

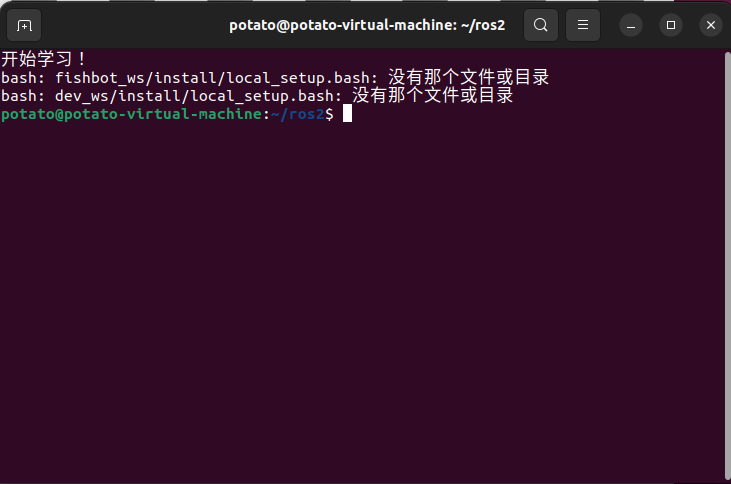
* cd（change directory）

$ cd filename（进入名为filename的文件夹下）$ cd（~） （切换到主目录/home/用户目录）

$ cd filename1/filename2（进入filename1/filename2）$ cd ..（返回上一级文件夹）



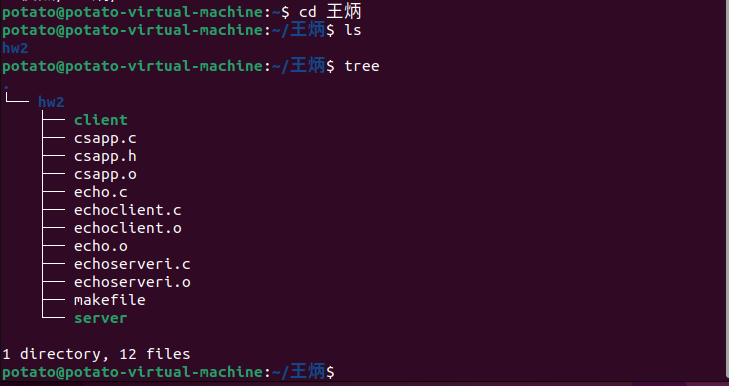
打开终端的方式可以右键打开终端，也可以使用快键键CTRL+ALT+T。在此不建议进入某个文件夹然后右键打开终端，因为在之后的学习中，会设计到配置环境的问题（.bashrc文件），一些环境的配置涉及到文件的路径，而右键进入终端会导致路径出错，又要自己重新配置。所以建议通过CTRL+ALT+T进入终端，然后再不断的cd进入文件夹，或者在配置环境路径的时候，就在固定的文件夹下设置路径（比较容易记不住）。



* ls（list）和tree

$ ls（查看当前文件夹下有什么文件） $ ls ..（查看上一级文件夹下有什么文件）

$ tree（以树状图列出文件目录结构）



* mkdir（make directory）

创建：对于文件夹的创建，就需要用到一个mkdir指令：

$ mkdir filename（用于创建单个文件夹） $ mkdir filename1 filename2（用于创建多个文件夹）

$mkdir -p file1/file2/file3（用于创建多级目录）

此外，对于mkdir指令还有mkdir -m的指令是用于创建目录并且设置权限。

* rm（remove ）

删除：对于文件或目录的删除，就需要用到rm指令：

假如现在有一个“Helloworld.txt”——$rm Helloworld.txt（删除该目录）

$mkdir a（创建一个名为“a”的文件夹） $rm -r a（删除文件夹必须用-r）

* touch

创建：对于目录的创建需要用到“touch”

$touch Helloworld.c（创建一个C文件） $touch Helloworld.txt（创建一个文本文件）

1. 基础工具的使用

* vi/vim、gedit等编辑器的使用（编辑器还有nano）

[Linux—vi/vim命令详解（3000字）\_linux vi-CSDN博客](https://blog.csdn.net/m0_54958293/article/details/127536929?ops_request_misc=%7B%22request%5Fid%22%3A%22f1a55670eb3036bc95b2b03a1f32d588%22%2C%22scm%22%3A%2220140713.130102334..%22%7D&request_id=f1a55670eb3036bc95b2b03a1f32d588&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~top_positive~default-1-127536929-null-null.142^v100^pc_search_result_base9&utm_term=vi&spm=1018.2226.3001.4187)

$vi/vim directory（进入编辑名为directory）

vi/vim有三种模式：命令行模式，插入模式，底行模式

默认vi/vim的时候是命令行模式；进入命令行模式后，按“I”进入插入模式；进入插入模式后按“ESC”退出插入模式切换为命令行模式；按“：”进入底行模式。

底行模式 :w [file-name]：[ 以指定文件名称 ] 保存文件

:w!：强制保存文件

:wq [file-name]：[ 以指定文件名称 ] 保存文件并退出

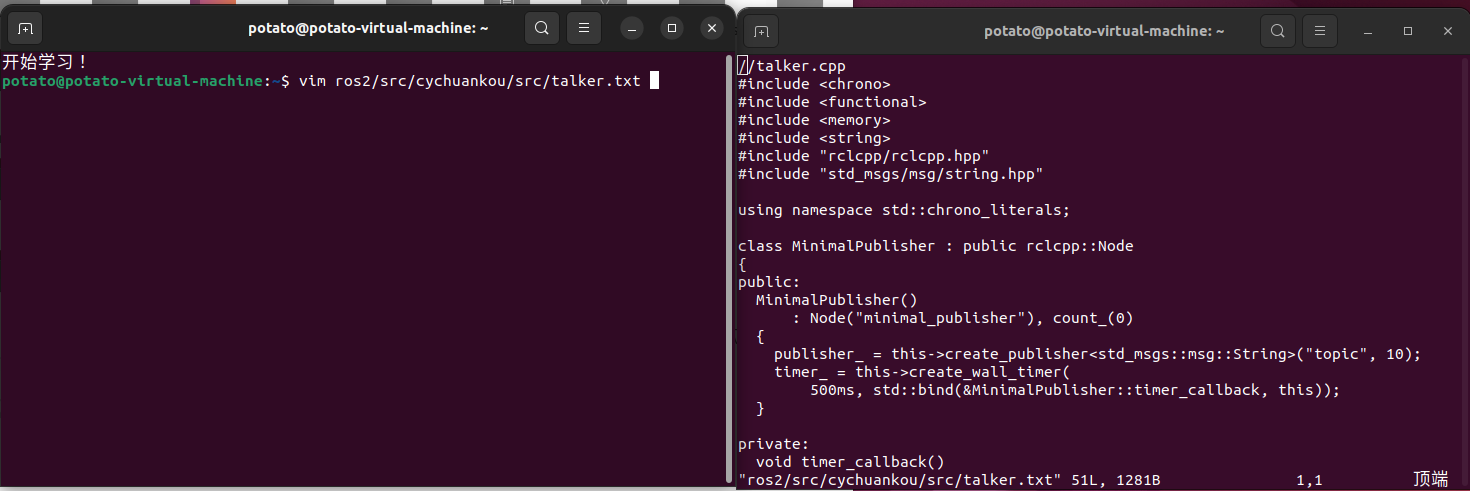
:wq!：强制保存文件并退出

:q：退出文件

:q!：不保存文件并强制退出

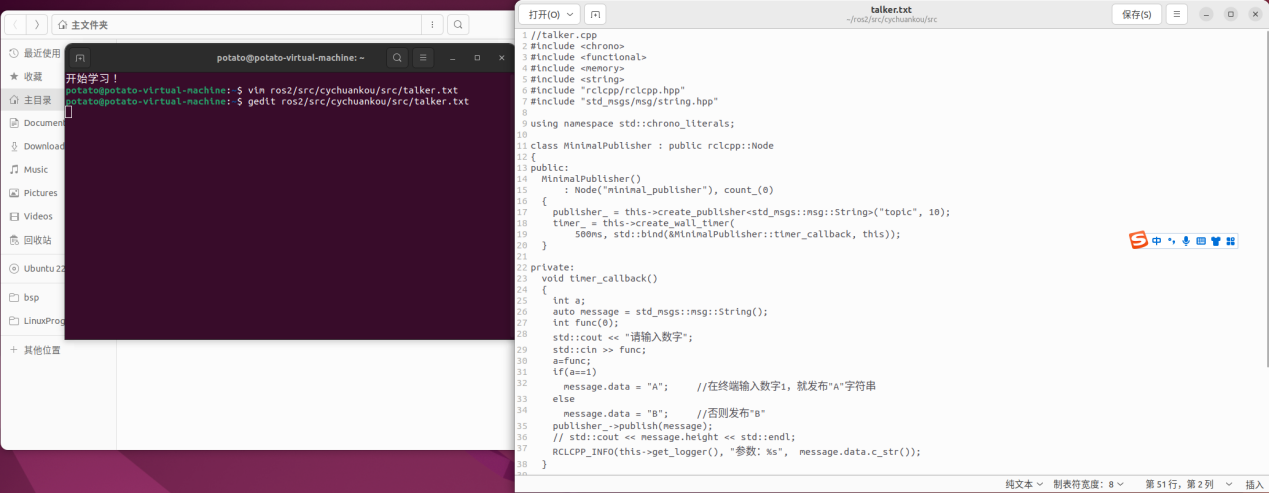
:r [file-name]：在当前文件中读入另一个文件的数据加到光标所在行后面

:2,3 w [file-name]：将第 2 行 到第 3 行的内容保存成 filename 这个文件



$gedit directory（进入编辑名为directory）

此方式为进入文本编辑器，这是最简单的编辑方法，输入“gedit directoy”便可进行编辑，保存也很方便。



## ROS2的简介及安装

1. ROS简介

ROS（Robot Operating System）是一个用于编写机器人软件程序的灵活的软件框架。它提供了一系列的工具、库和约定，旨在简化复杂机器人行为的编程实现过程。

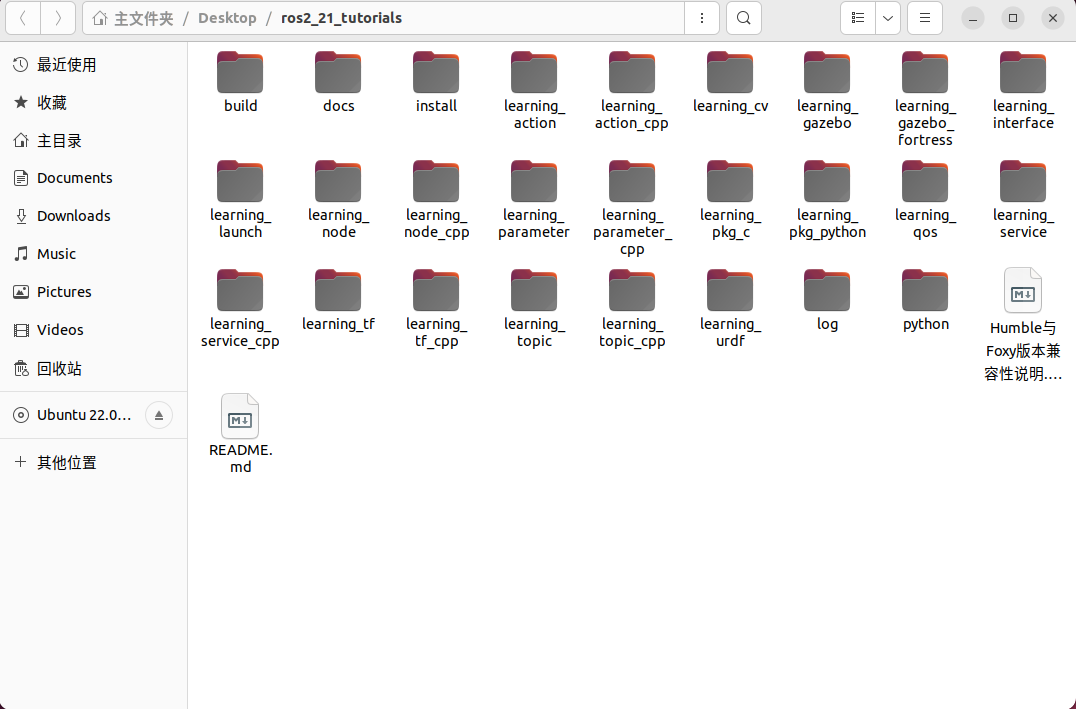
从本质上来说，它不是一个传统意义上的操作系统，如 Windows 或 Linux，而是构建在现有操作系统之上的一个中间件，主要用于机器人开发。它使得开发人员可以专注于机器人的功能实现，而不用花费大量精力在底层通信、硬件驱动适配等复杂的基础工作上。

ROS是一个分布式的框架，这意味着它可以在多个计算机或者处理器核心上运行不同的节点（Node）。例如在一个复杂的机器人系统中，一个节点可以负责机器人手臂的运动控制，另一个节点可以处理视觉传感器的数据采集和处理，这些节点可以通过网络通信相互协作。

节点之间通过消息（Message）传递机制进行通信。消息是一种轻量级的数据结构，例如，机器人的位置信息可以通过包含 x、y、z 坐标的消息在节点之间传递。这种通信方式使得各个功能模块可以独立开发和测试，然后再集成到整个系统中。

ROS 的软件系统是由一系列的功能包（Package）组成的。功能包是一个独立的软件单元，包含了实现特定功能的代码、配置文件和文档等。比如，有专门用于激光雷达数据处理的功能包，也有用于机器人路径规划的功能包。

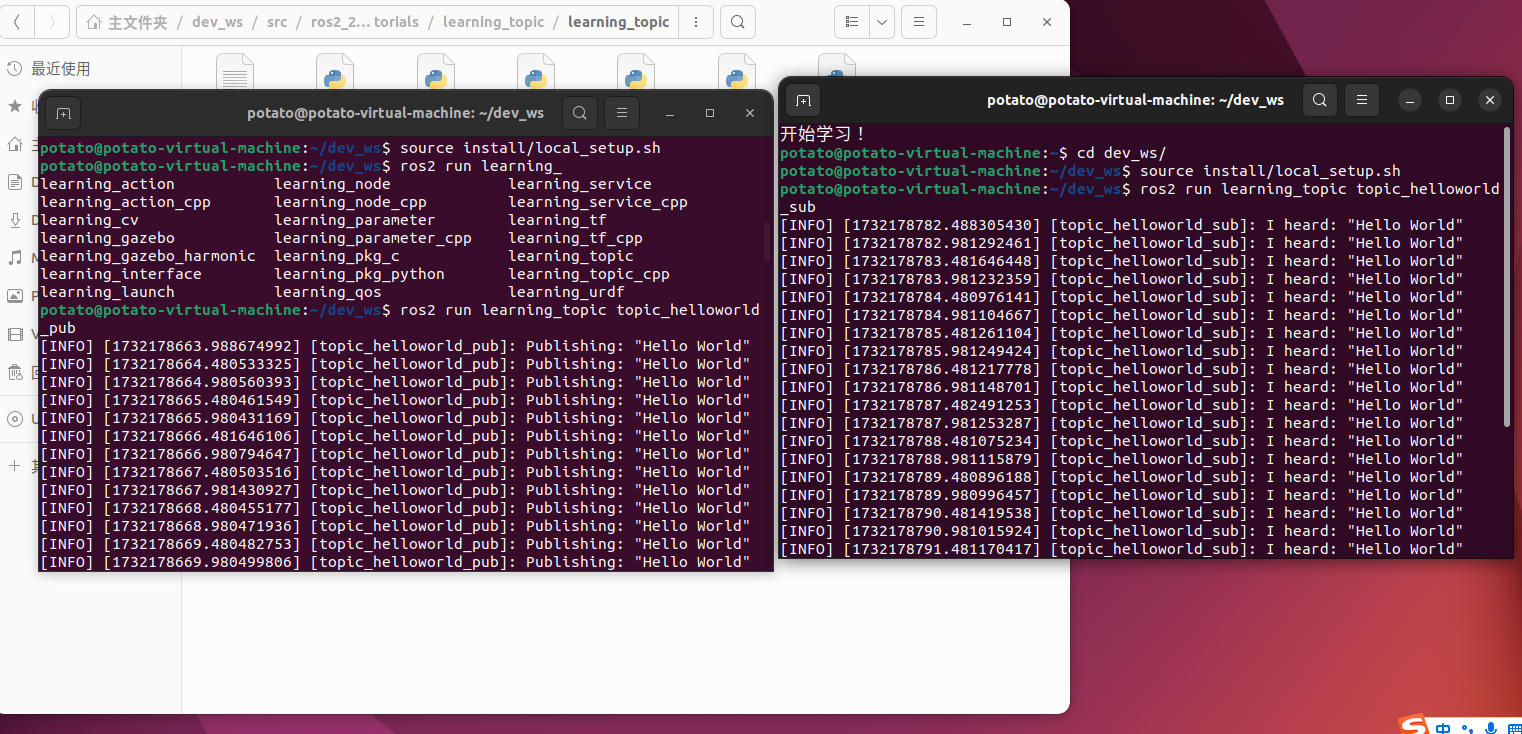
开发人员可以方便地利用现有的功能包来构建自己的机器人应用程序，也可以创建新的功能包来扩展系统的功能。这种模块化的设计使得代码的复用性很高，大大提高了机器人软件开发的效率。

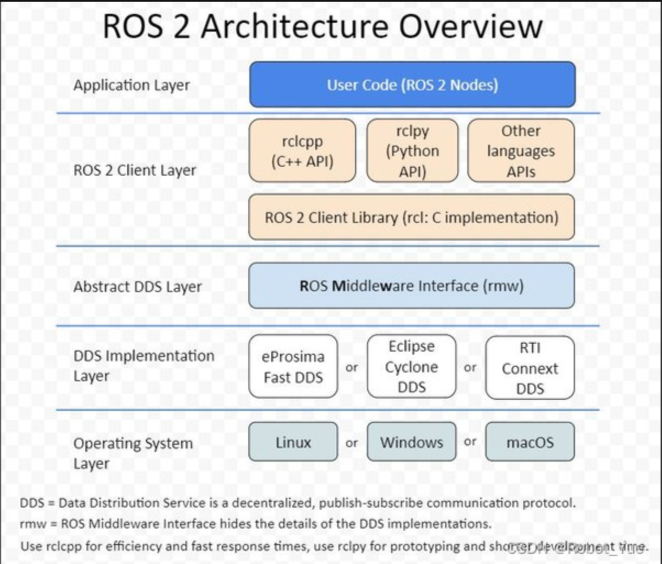


如上图官方例程所示，是一个独立的软件单元，包含了实现特定功能的代码、配置文件和文档等。对于模块化的测试有很好的便捷性。

1. ROS2与ROS的区别

ROS2 采用了数据分发服务（DDS，Data - Distribution Service）架构。DDS 是一种高性能、以数据为中心的分布式通信中间件标准。在这种架构下，节点之间可以直接进行通信，无需像 ROS 那样依赖一个中央的 Master 节点。每个节点都可以独立地发现其他节点并建立通信连接。这使得系统更加健壮，即使部分节点出现故障，其他节点之间的通信依然可以正常进行。例如，在一个由多个机器人组成的协作系统中，即使其中一个机器人的某个节点出现故障，其他机器人之间的通信和协作任务不会立刻受到干扰。





ROS1使用Catkin编译系统，在经历catkin编译后会产生四个空间：src代码空间——用于存储所有功能包的代码；build编译空间——用于存储编译过程产生的缓存和中间文件；devel开发空间：用于放置编译生成的可执行文件，install安装空间：非必须的，可在工作目录下用catkin\_make install生成，可以将可执行文件安装到此空间。ROS2使用colcon编译，编译后产生的四个文件夹分别为：src代码空间——用于存储所有功能包的代码、脚本等；build编译空间——用于存储编译过程产生的缓存和中间文件；install安装空间——用于放置编译生成的可执行文件和脚本；log日志空间——用于保存编译和运行过程中警告、报错等日志的空间。

1. ROS2安装
2. [【ROS2】ROS2安装详细教程（以Humble为例）\_ros2教程-CSDN博客](https://blog.csdn.net/maizousidemao/article/details/144008825?ops_request_misc=%7B%22request%5Fid%22%3A%220f49e947ad49b359be30437ac40855b3%22%2C%22scm%22%3A%2220140713.130102334..%22%7D&request_id=0f49e947ad49b359be30437ac40855b3&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduend~default-2-144008825-null-null.142^v100^pc_search_result_base9&utm_term=ROS2%E7%9A%84%E5%AE%89%E8%A3%85&spm=1018.2226.3001.4187)
3. 一键安装：wget http://fishros.com/install -O fishros && . fishros

[ROS篇——Ubuntu快速一键安装ROS或ROS2（通用）\_ros一键安装-CSDN博客](https://blog.csdn.net/u011391361/article/details/131582593?ops_request_misc=%7B%22request%5Fid%22%3A%2258aac22d3d951fb1987b308f55afc13d%22%2C%22scm%22%3A%2220140713.130102334..%22%7D&request_id=58aac22d3d951fb1987b308f55afc13d&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~top_positive~default-1-131582593-null-null.142^v100^pc_search_result_base9&utm_term=ROS2%E4%B8%80%E9%94%AE%E5%AE%89%E8%A3%85&spm=1018.2226.3001.4187)

1. 安装好虚拟机后其实就可以再安装好一个vmware Tools。这个就相当于virtual box的增强。安装后不仅可以对性能有一个提升（包括网络方面），而且也有许多功能可以使用。比如主机与虚拟机之间的复制粘贴等。

如果你不安装的话想从主机中拷贝一个文件到虚拟机，就可能要使用到挂载的方法——挂载指令：$ sudo vmhgfs-fuse -o allow\_other -o auto\_unmount .host:/ /mnt/hgfs（该指令要在虚拟机设置里设置共享文件夹的路径，然后要在虚拟机文件夹中的计算机文件夹中创建/mnt/hgfs），然后就可以实现主机与虚拟机文件的共享。

[虚拟机VMware Tools安装步骤\_vm tools-CSDN博客](https://blog.csdn.net/m0_69057918/article/details/130575353?ops_request_misc=%7B%22request%5Fid%22%3A%2203ab11c96b681aedb0c90be908461837%22%2C%22scm%22%3A%2220140713.130102334..%22%7D&request_id=03ab11c96b681aedb0c90be908461837&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~top_positive~default-1-130575353-null-null.142^v100^pc_search_result_base9&utm_term=vmware tools&spm=1018.2226.3001.4187)

## ROS2基础学习

1. 工作空间（workplace）

以开发一个机器人或者一个机器小车为例，将开发所需要的代码，参数，脚本等放在一个统一的文件夹进行管理，这个文件夹就是工作空间。工作空间创建好之后，对其进行编译。colcon build——集体编译

colcon build --packages-select YOUR\_PKG\_NAME

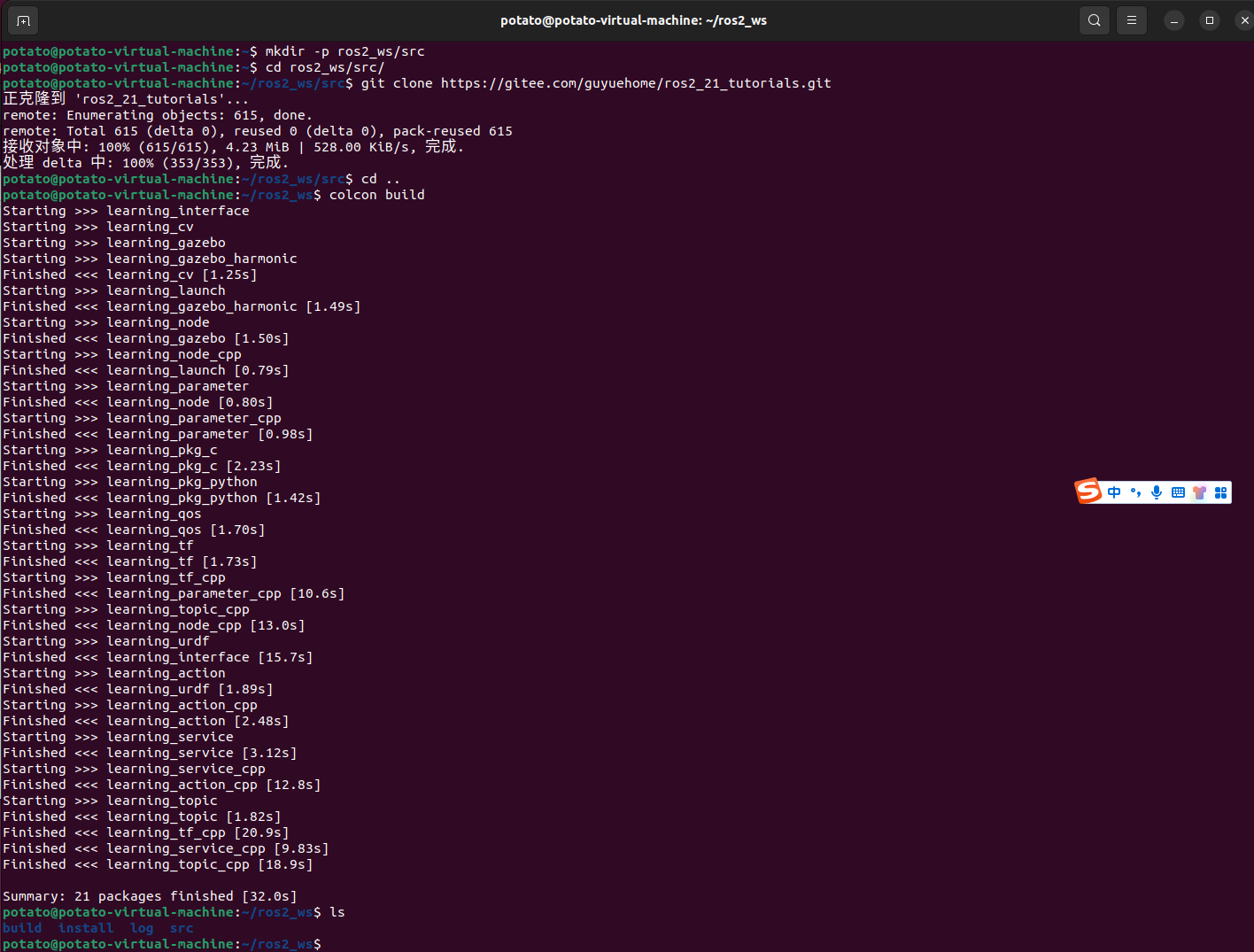
接下来我们新建一个工作空间，将它命名为“ros2\_ws”，然后创建一个“src”的代码空间，接下来把官方例程拷贝进来。

$ git clone https://gitee.com/guyuehome/ros2\_21\_tutorials.git

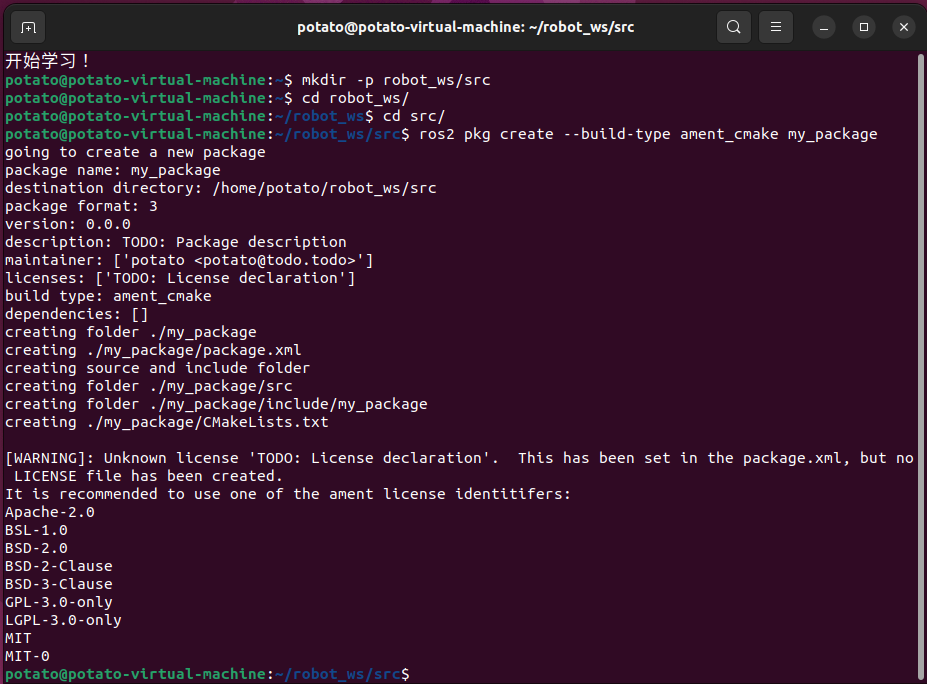
随后进入工作空间进行编译。编译之后，工作空间下的目录会额外产生三个文件夹，上述我们讲了，分别是“install”，“build”，“log”。在编译之后很重要的一个步骤就是加载配置文件——$. install/setup.bash。这一步之后，我们才能把我们写好的节点变成可执行文件，否则，我们在ros2 run……的时候是找不到该节点的。当然我们也可以在打开终端的时候就配置好这一点。

* 找到工作空间下的install文件夹下setup.bash的路径。
* 然后在主文件夹下找到.bashrc文件（该文件夹是启动脚本，在打开终端的时候，Ubuntu会自动启动该脚本）。在最后的代码中加入 $source ros2\_ws/install/setup.bash。如果在主文件夹下没有找到.bashrc文件，“Ctrl”+“H”可以显示隐藏的文件夹。

建议使用第一次编译的时候使用第一种编译方法，在工作空间中可能有很多软件包，可检查其他功能包是否有错误，也不影响自己想要检查的功能包的编译或者使用功能包。



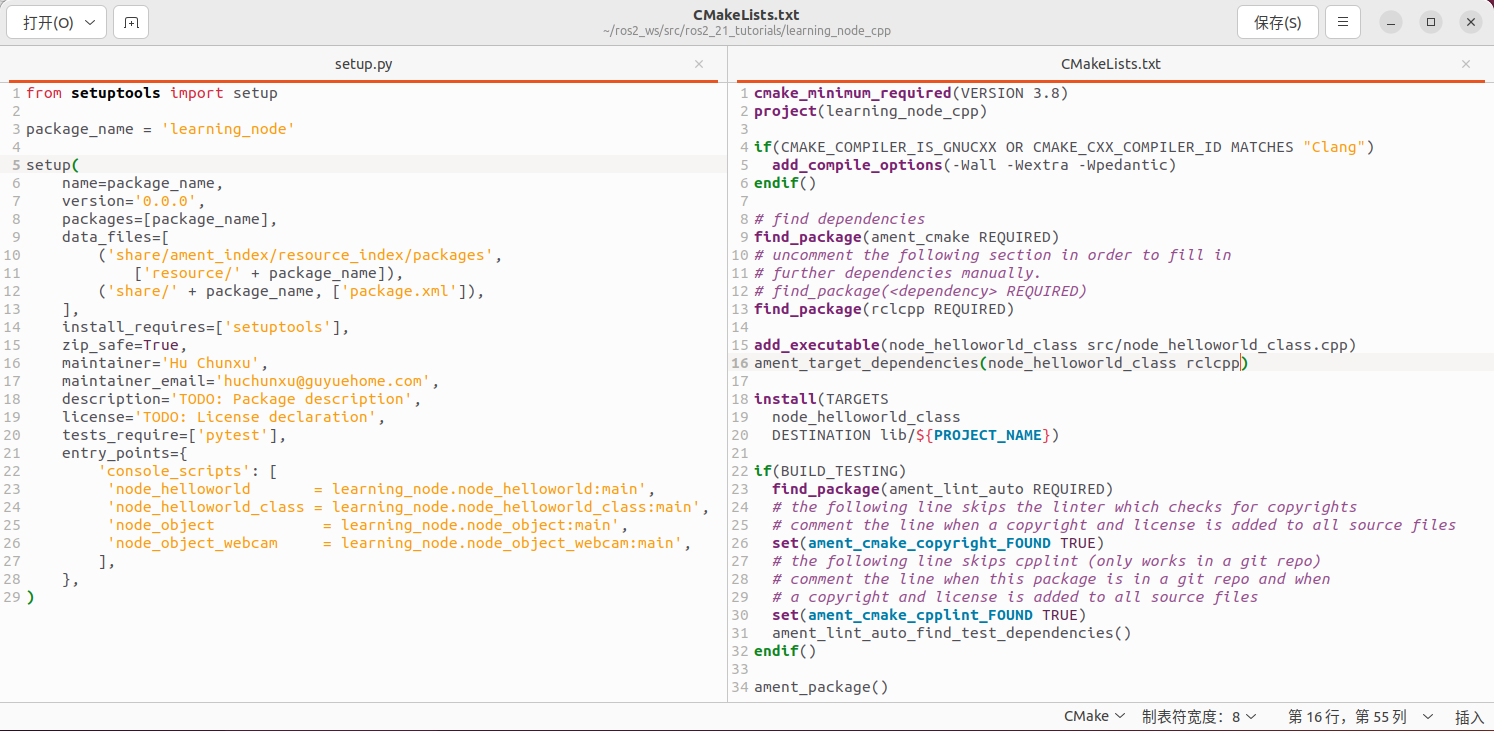
这里使用到了“git”相关的指令，克隆了官方例程的软件包。为ROS2基础的学习可以进行参考。我们在自我进行开发的时候可以自己创建软件包。 $ ros2 pkg create --build-type ament\_cmake <package name> $ ros2 pkg create --build-type ament\_python <package name>

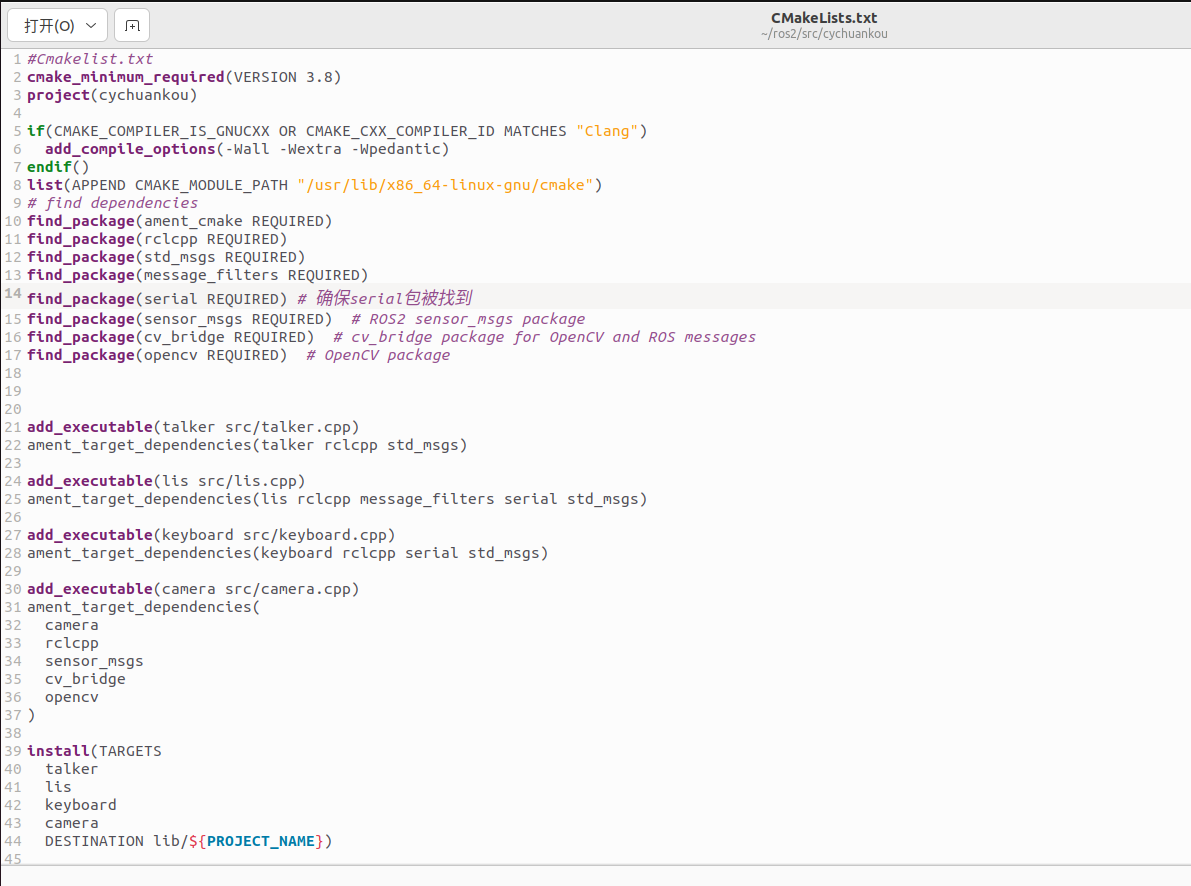


1. 节点：机器人细胞

节点类比来说可以是一个程序的进程，它是一个可以独立运行的“可执行文件”。每一个节点都负责一个单独的模块化的功能。

节点可使用不同的编程语言来进行编写，结点们也可分布式的在不同的主机上运行。因此，我们对一个机器人的操控，就简化成了对每一个独立可执行的节点的控制。对于每个节点都有不同的名称，于是就会通过节点名称进行管理。注意的是对于节点，需要在setup.py（Cmakelist.txt）中增加节点的名字绑定节点程序（配置信息）。





接下来我们以官方例程的几个结点为例进行演示。

$ ros2 run learning\_node node\_object

$ ros2 run learning\_node node\_object\_webcam

（运行ros2 run learning\_node node\_object -> setup.bash）

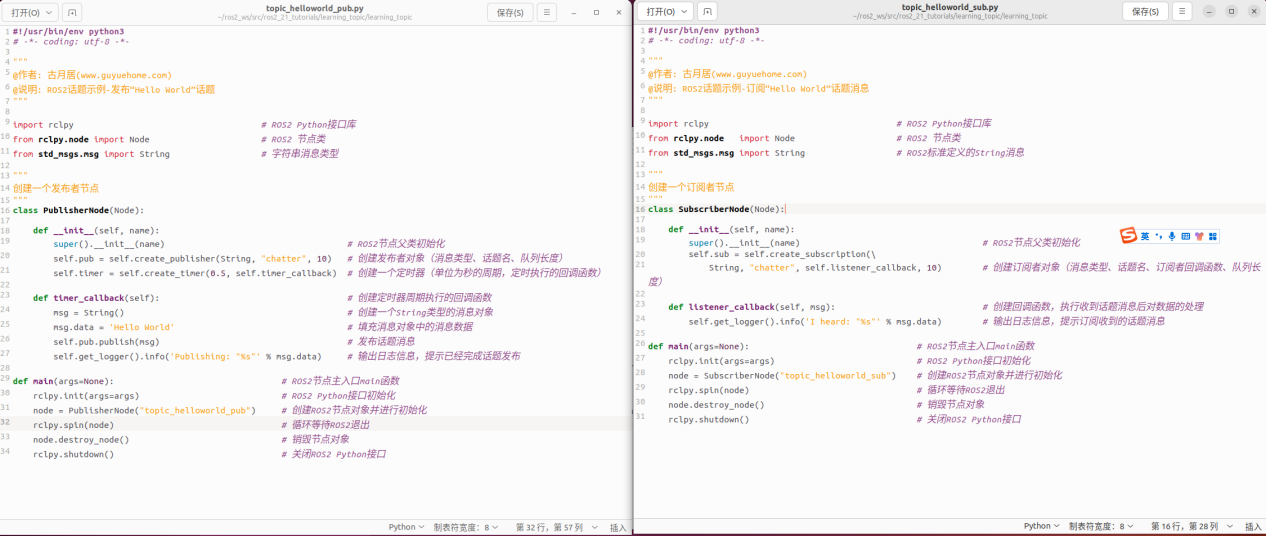
1. 话题（Topic）

节点是独立存在的，但不是孤立的。节点与节点之间也是需要通信的，在ROS2中有四种通信方式，其中一种就是话题。话题采用的是订阅发布模型，一个节点将数据发布到某个话题上，另一个节点就可以通过订阅话题接收到那个节点所发布的数据。这种话题方式并不是一对一的，可以是一对多，多对一，多对多的。

$ros2 run learning\_topic topic\_helloworld\_pub

$ros2 run learning\_topic topic\_helloworld\_sub





要注意的是，以上述为例，在topic\_helloworld\_pub和topic\_helloworld\_sub两个程序中，他们都是生成了两个结点。创建的方法都是类似的，分别创建发布/订阅者对象，包括其消息类型，话题名，回调函数，队列长度等。通过回调函数，执行收到话题信息后的处理。

另外，还有一个问题就出现了。举个例子：现在有一个节点A发布了一个话题topic\_A，然后节点B订阅了节点A发布的这个话题TopicA，然后呢就实现了A节点到B节点的通信，B节点便通过订阅A节点的这个话题收到来自A节点传输的数据，然后通过回调函数做出一系列的动作，比如“topic\_helloworld\_sub”就是做出了“I heard:hello world！”，只实现了A节点到B节点的单向传输，A节点并没有收到来自B节点对A节点的反馈，这个时候A节点也不能作出下一个动作的决策。

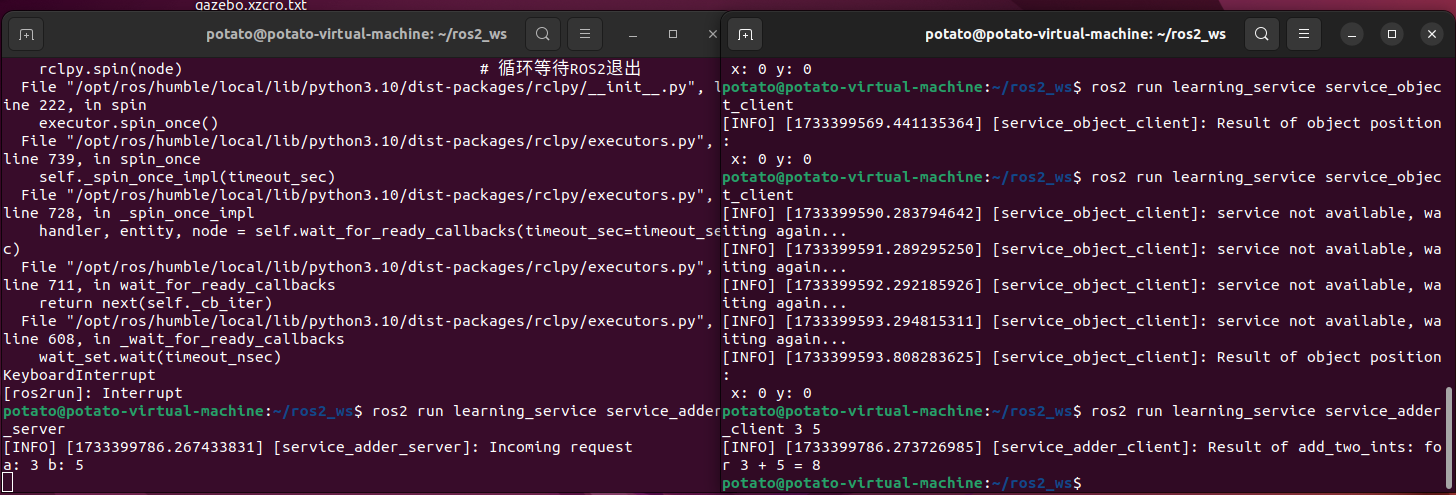
1. 服务（service）通信

很多时候，只有在某些特定的情况下才会需要节点提供信息，此时由一个节点发送一个请求信息，另一个节点反馈一个应答，这样的通信机制在ros中称为服务。这种由一个结点发出请求，另一个节点又进行一个反馈的模型就是我们所熟悉的客户端/服务器（C/S）模型，是一种一对多的通信模型（一个服务器，多个客户端），这种机制也是一种同步通信机制。

以下是官方的例程。我们先开启服务器，服务器就等待客户端的连接， 在客户端中，要发出一个请求，这个请求在回调函数的定义就是实现一个两个数字的加法，所以客户端在请求的时候要带两个参数，然后客户端发出请求的时候便立即得到了服务器的反馈。这种服务通信实现了来自两个节点的双向通信。

$ros2 run learning\_service service\_adder\_server

$ros2 run learning\_service service\_adder\_client



1. 通信接口（interface）

我们创建话题的时候，需要指定话题的类型，创建服务的时候也需要指定客户端和服务端服务的类型，不管是话题还是服务，这些类型都是[结构体](https://marketing.csdn.net/p/3127db09a98e0723b83b2914d9256174?pId=2782&utm_source=glcblog&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/m0_60274660/article/details/_blank)。而这些通信的对象，不管是发起端还是接收端，都要指定[数据通信](https://so.csdn.net/so/search?q=%E6%95%B0%E6%8D%AE%E9%80%9A%E4%BF%A1&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/m0_60274660/article/details/_blank)的接口类型，而且只有发起端和接收端的接口类型一致，才能进行正常的数据通信。在可能实际的开发过程中，会遇到一些问题，比如小车上有一个激光雷达，摄像头，他们传输的数据类型不一样，因此他们不能做到双方的通信，所以我们要统一他们的数据类型（实质上是统一结构体）。

[ros2节点](https://so.csdn.net/so/search?q=ros2%E8%8A%82%E7%82%B9&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/klp1358484518/article/details/_blank)之间通讯一般使用std\_msgs/下的标准数据类型，当我们要定义自己的消息类型的时候，需要定义通讯接口（数据类型）文件。自定义的通讯数据类型可以用已有的通讯数据类型进行组装，也可以使用ros2规定的原始数据类型组装。已有的通讯接口定义可以使用ros2的命令查看：$ ros2 interface list。

* 话题接口msg：话题的传输是单向传输，所以只需要描述一次传输的数据就可以了，比如
* int32 x;

int32 y;

* 服务接口srv:：服务便要包括请求和应答，以上述为例

int64 a;int64 b;#请求的数据

---

int64 sum;#响应的数据

* 动作接口action：动作接口文件则需要目标+结果+之间状态反馈

# Goal: 要移动的距离

float32 distance

---

# Result: 最终的位置

float32 pose

---

# Feedback: 中间反馈的位置和状态

float32 pose

uint32 status

uint32 STATUS\_MOVEING = 3

uint32 STATUS\_STOP = 4



在定义完接口之后，我们和之前一样，也要进行Cmakelist/setup.py的更改进行接口路径的增加。

以Cmake为例：

1. 添加生成接口的cmake接口库：

find\_package(rosidl\_default\_generators REQUIRED)

1. 添加接口文件中依赖的其他接口库：

uint32 STATUS\_MOVEING = 1

uint32 STATUS\_STOP = 2

uint32 status

geometry\_msgs/Pose pose

cmake：find\_package(geometry\_msgs REQUIRED)

1. 利用cmake的接口将指定的接口文件生成接口

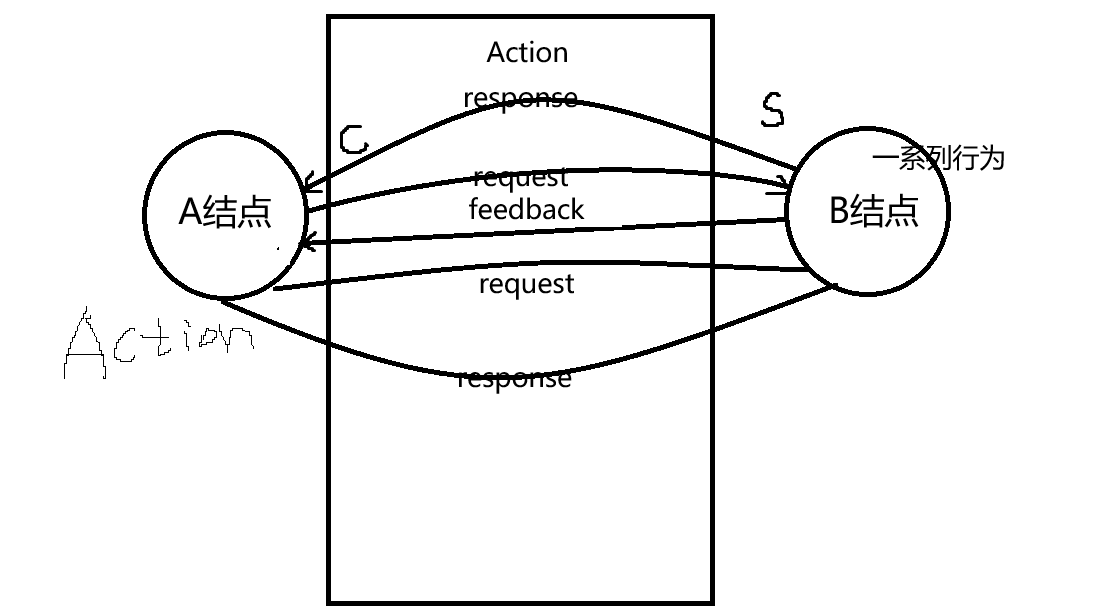
rosidl\_generate\_interfaces(${PROJECT\_NAME}"msg/RobotPose.msg" "msg/RobotStatus.msg""srv/MoveRobot.srv" DEPENDENCIES geometry\_msgs )

1. 动作（action）

对于实际应用场景，我们希望使用一个指令控制机械臂抓起面前的方块。这种情况可以用话题，机械臂节点订阅该话题，我们向话题中发布消息，从而使机械臂移动。也可以使用服务，客户端向机械臂节点发送消息，机械臂作为服务端在确认收到消息后返回消息，告知消息已接收。

但以上两种操作中，我们都无法得知机械臂具体进行了什么操作。于是动作的出现，完美解决了这个问题。动作的本质是在发送了请求后，接收端重复周期性返还数据，以告知消息发送者当前动作进行到了哪里，以此来实现对动作的全程跟踪。那么显然，所应用的场景也是客户端/服务器模型。

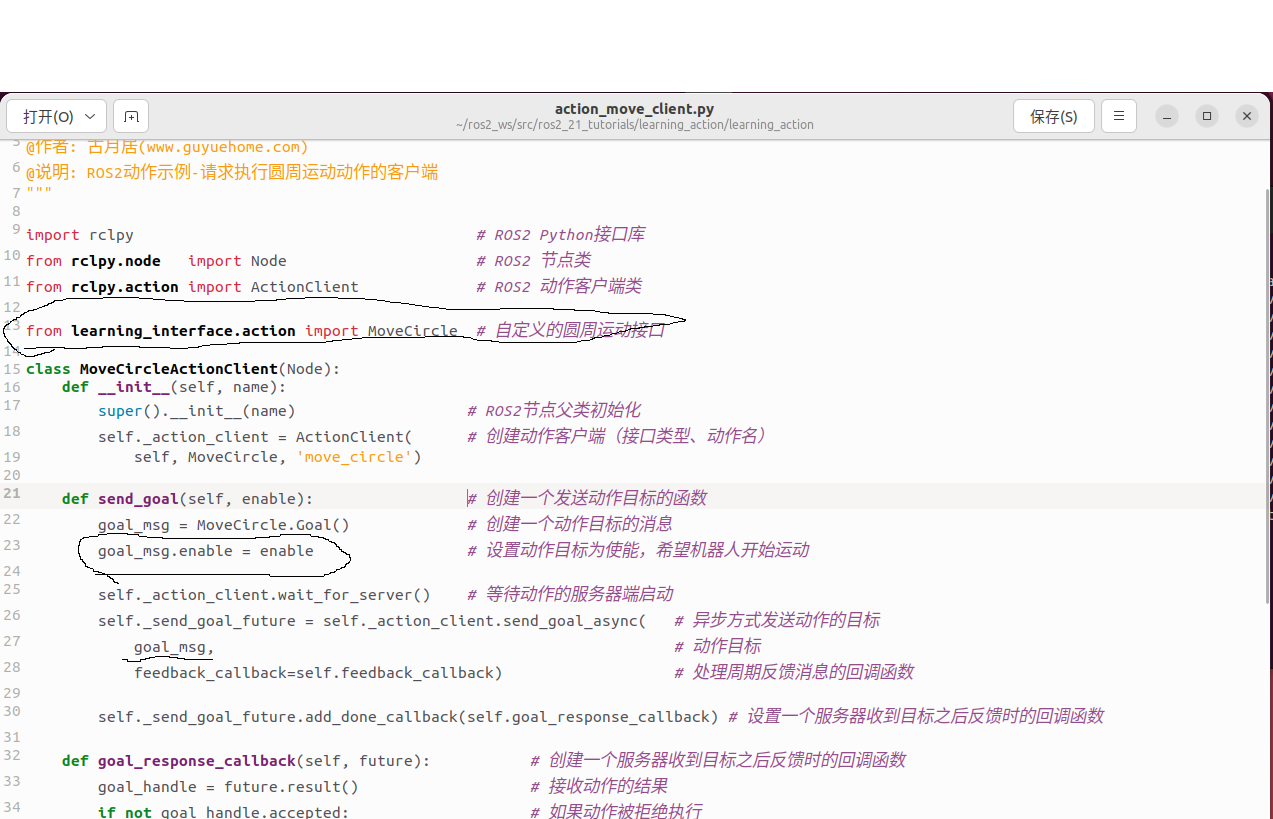
举个形象的例子：



上图中的feedback是一个循环，人为控制。具体关键的流程如上，在代码上的体现是在回调函数的应用上。从代码上更容易理解。

$ros2 run learning\_action action\_move\_server

$ros2 run learning\_action action\_move\_client



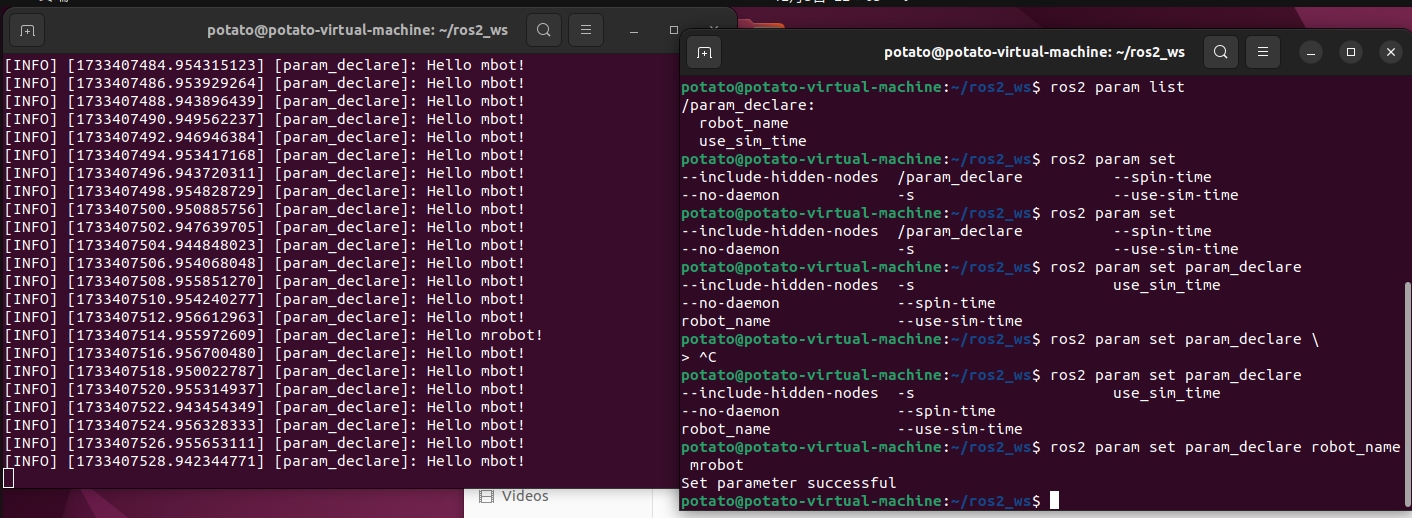
1. 参数（parameter）

在ROS2系统中，参数是以全局字典的形态存在的，由键和值组成，合成键值。类似于全局变量，可以在多个程序中共享某一些数据。参数则是ROS机器人系统中的全局字典，可以运行多个节点中共享数据。

ROS2 中的参数（Parameters）是一种用于配置节点（Nodes）行为的机制。参数可以被看作是节点的配置变量，通过这些参数，可以在不重新编译代码的情况下调整节点的功能特性。例如，在一个机器人的运动控制节点中，可以通过参数来设置机器人的最大速度、加速度等。这样，当需要在不同场景下（如室内环境需要较低速度，室外环境允许较高速度）使用机器人时，只需要修改参数值，而不需要修改和重新编译运动控制代码。

$ ros2 run learning\_parameter param\_declare

$ ros2 run param set param\_declare robot\_name mrobot



1. 分布式通信

ROS2当中，机器人的各种功能是通过节点来控制实现的。但是机器人的节点可能分布在不同的计算机上。这种方式的好处就是可以将原来复杂大量的任务分配到不同的平台上面来减轻一些压力。这就是ROS2中分布式通信框架典型的一个应用。

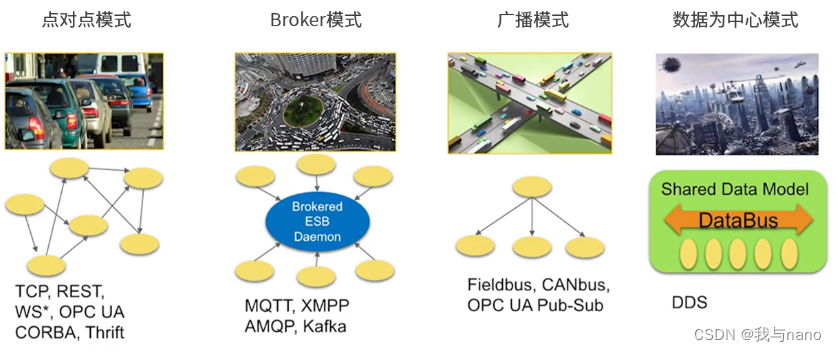
在这种框架下一定要注意的是要让两主机处于在同一个网络下，这样才能实现通信。比如，你的树莓派开发板和你的远程显示器进行连接，要通过ssh指令来进行连接，比如ssh username@ip\_address，也可以指定目的端口号ssh -p 2222 user@hostname进行远程连接。

在ROS2中提供了一种DOMAIN的机制，就类似于分组的意思，处于同一个DOMAIN的计算机才能进行通信。

$export ROS\_DOMAIN\_ID=<your\_domain\_id>，相同的计算机上只有设置成相同的ROS\_DOMAIN\_ID才能进行相互通信，如果分配的ID不同，则两者无法实现通信。

1. 数据分发服务（DDS）

ROS2中最为重大的变化——DDS，我们在前边的话题、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。DDS的核心是通信，能够实现通信的模型和[软件](https://marketing.csdn.net/p/3127db09a98e0723b83b2914d9256174?pId=2782&utm_source=glcblog&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/qq_51963216/article/details/_blank)框架非常多，这里我们列出常用的四种模型。

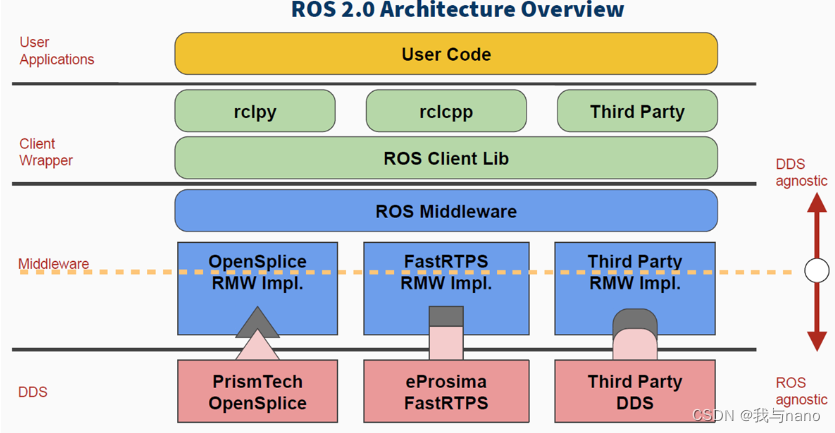


第一种，点对点模型，许多客户端连接到一个服务端，每次通信时，通信双方必须建立一条连接。当通信节点增多时，连接数也会增多。而且每个客户端都需要知道服务器的具体地址和所提供的服务，一旦服务器地址发生变化，所有客户端都会受到影响。

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。



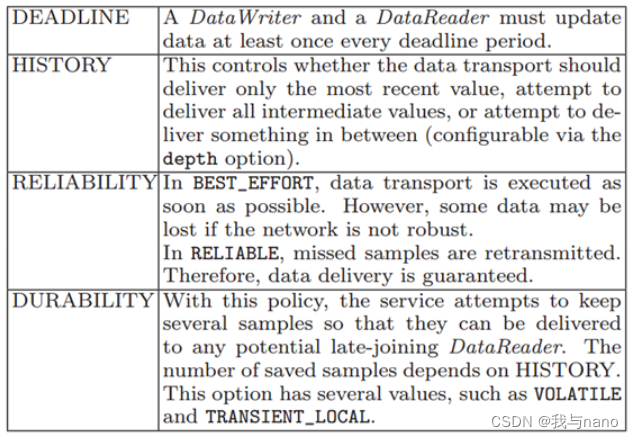
重点是，DDS是一种通信的标准，就跟3G,4G,5G一样，大家都可以使用按照这个标准实现对应的功能DDS也是一样，能够按照DDS标准实现的通信系统很多，这里每一个红色模块，就是某一企业或组织实现的一种DDS系统。

既然可选用的DDS很多，ROS2的原则就是尽量兼容，让用户根据使用场景选择，比如个人开发，我们选择一个开源版本的DDS就行，如果是工业应用，那可能得选择一个商业授权的版本了。

为了实现对多个DDS的兼容，ROS设计了一个Middleware中间件，也就是一个统一的标准，不管我们用那个DDS，保证上层编程使用的函数接口都是一样的。此时兼容性的问题就转移给了DDS厂商，如果他们想让自己的DDS系统进入ROS生态，就得按照ROS的接口标准，开发一个驱动，也就是这个部分。

说了这么多听着也有些空洞，不知道这个DDS到底是个啥东西。这里介绍一下DDS的Domain和QoS。DDS中的基本结构是Domain，它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。



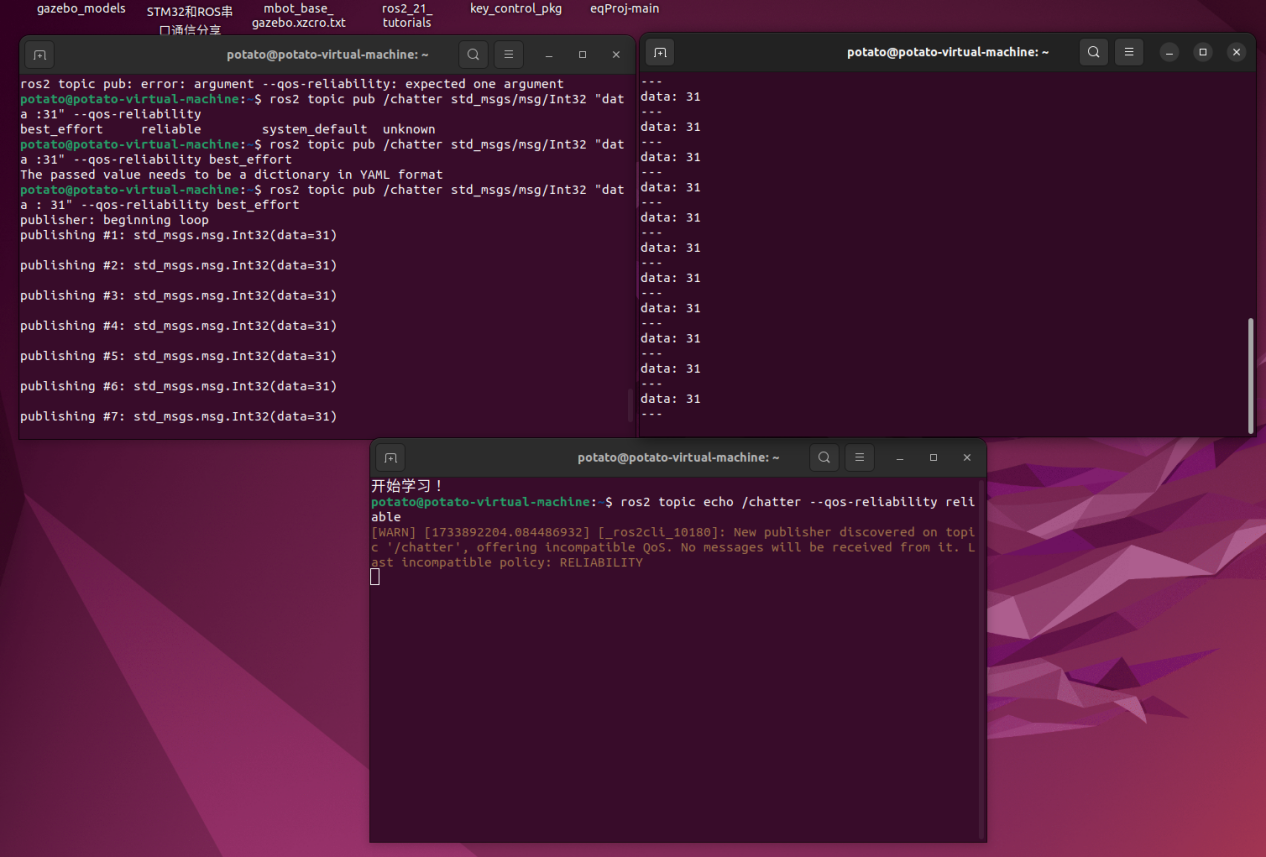
DEADLINE策略，表示通信数据必须要在每次截止时间内完成一次通信；

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。

Learning\_qos



## ROS2进阶和应用

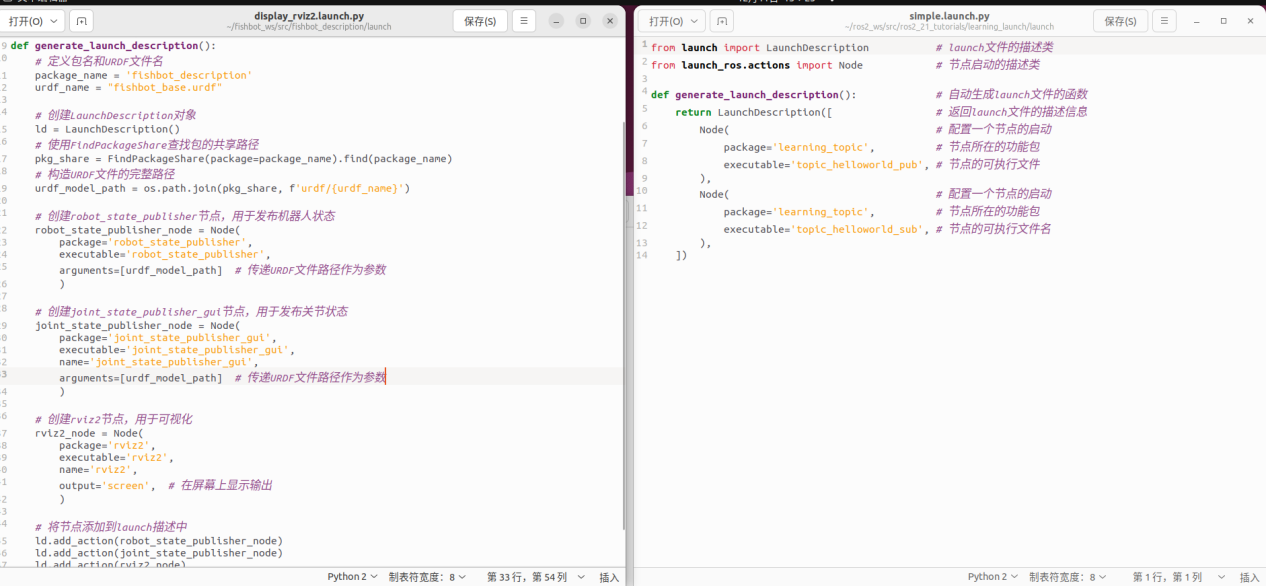
目前为止，我们理解到了ROS2的一个大概框架。但是ROS2上手难的原因是，我们只是会理解到了ROS2整个运行的框架，通过官方例程对ROS2有个认识，有了初步应用（能够使用）。但是从零开始依然还是很难，接下来以一个小车仿真为例，进一个深入理解ROS2的应用。

1. Launch

我们目的是做一个机器人，或者是一个小车。去实现一个避障，视觉识别等等。那么我们对于这些功能都有一个清晰的思路：就是去写一个节点——那么这个节点的作用可能就是“打开雷达扫描”，“打开摄像头识别”等等，然后需要的时候就启动节点，就完成了该功能。但是如果功能一旦变得越快越多，那么我们就需要打开的节点就越多，们每次运行一个节点都需要打开一个终端，然后运行一个命令，可以说是非常的麻烦，而且他们之间有各种关联。于是就有了launch启动文件，可以配置多个节点的启动。

$ ros2 launch learning\_launch simple.launch.py

$ ros2 launch fishbot\_description display\_rviz2.launch.py



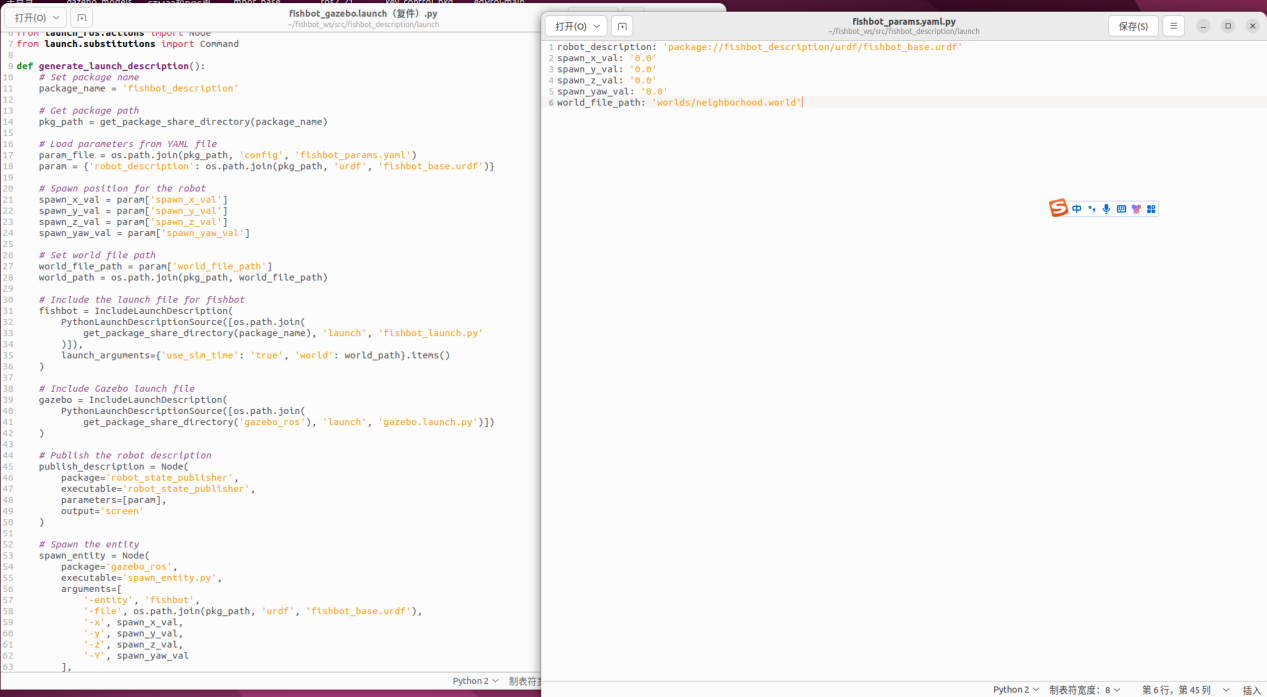
\*\*\*要注意的是，在ROS2里可能会有多个程序同时运行一个节点的情况（也有可能是需要数据），但是ROS2里是不允许同名节点的存在，所以可以重新命名。如图一中rviz的节点，功能包，可执行程序，节点名字，都是rviz，可执行程序可运行不止一个，但是节点不可以——若是有别的程序也要需要名为“rviz”的节点，这个时候就会报错，所以要重命名。  
 为了简化参数的管理，特别是在参数较多的情况下，使用参数文件是一个好方法。你可以将所有的参数存储在一个YAML文件中，然后在launch文件中加载这个文件。

$ros2 launch learning\_launch parameters.launch.py

$ros2 launch learning\_launch parameters\_yaml.launch.py

$ ros2 param get /turtlesim2/sim background\_b





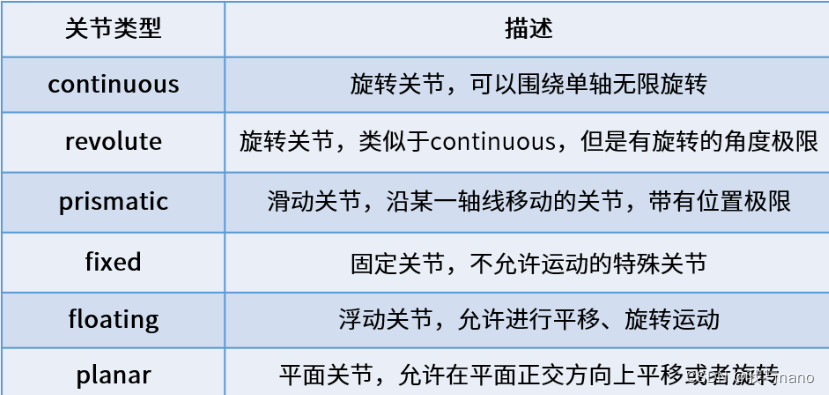
后续我们会提到一种机制，就是launch中包含launch文件，这种机制在我们开发一个项目中有版本备份的作用等。

1. URDF（Unified Robot Description Format）

ROS专门提供了一种机器人建模方法——URDF，用来描述机器人外观、性能等各方面属性。建模描述机器人的过程中，我们自己需要先熟悉机器人的组成和参数，比如机器人一般是由硬件结构、驱动系统、传感器系统、控制系统四大部分组成，市面上一些常见的机器人，无论是移动机器人还是机械臂，我们都可以按照这四大组成部分进行分解。

这里主要介绍连杆link和关节joint。

在建模中，机器人的大臂和小臂或者是小车的轴就类似机器人的这些独立的刚体部分，称为连杆Link，手肘或者是轮胎电机等就类似于机器人电机驱动部分，称为关节joint。所以在URDF建模过程中，关键任务就是理解清楚每一个连杆和关节的描述信息。Link标签就是用来描述机器人某个刚体部分的外观和物理属性，外观包括尺寸、颜色、形状，物理属性包括质量、惯性矩阵、碰撞参数等。机器人模型中的刚体最终要通过关节joint连接之后，才能产生相对运动。其运动类型有几种分别是continuous，revolute，prismatic，fixed，floating，以下是简单的描述各种运动。



接下来以我写的一个小车模型为例。<link>标签定义了机器人的各个部件，而<joint>标签定义了这些部件之间的连接关系。

$ ros2 launch fishbot\_description display\_rviz2.launch.py

$ ros2 run fishbot\_description rotate\_wheel

$ rqt

以下是各个<link>和<joint>之间的关系：

* Base Footprint and Chassis Link:

<link name="base\_footprint"/>: 这是一个虚拟的链接，通常用作机器人在世界坐标系中的参考点。

<joint name="base\_joint" type="fixed">: 这是一个固定类型的关节，它将 `base\_footprint` 链接到 `chassis\_link`。这意味着底盘（chassis）被视为机器人的物理位置和参考坐标系的固定部分。

* Chassis Link:

<link name="chassis\_link">: 定义了机器人的底盘，是机器人的主要结构部分。

* Laser Link and Joint:

<link name="laser\_link">: 定义了激光传感器的物理形状和外观。

<joint name="laser\_joint" type="fixed">: 固定关节将激光传感器（`laser\_link`）连接到底盘（`chassis\_link`）。激光传感器的位置和方向是固定的。

* IMU Link and Joint:

<link name="imu\_link">: 定义了惯性测量单元的物理形状和外观。

<joint name="imu\_joint" type="fixed">: 固定关节将IMU连接到底盘。IMU的位置和方向是固定的。

* Wheel Links and Joints:

<link="front\_right\_wheel\_link">,<linkname="front\_left\_wheel\_link">,<linkname="back\_right\_wheel\_link">, <link name="back\_left\_wheel\_link">`: 这些链接定义了机器人的四个轮子的物理形状和外观。

<jointname="front\_right\_wheel\_joint"type="continuous">,<jointname="front\_left\_wheel\_joint" type="continuous">, <joint name="back\_right\_wheel\_joint" type="continuous">, <joint name="back\_left\_wheel\_joint" type="continuous">: 这些连续关节允许轮子无限旋转。每个关节都连接一个轮子到底盘，并定义了轮子旋转的轴。

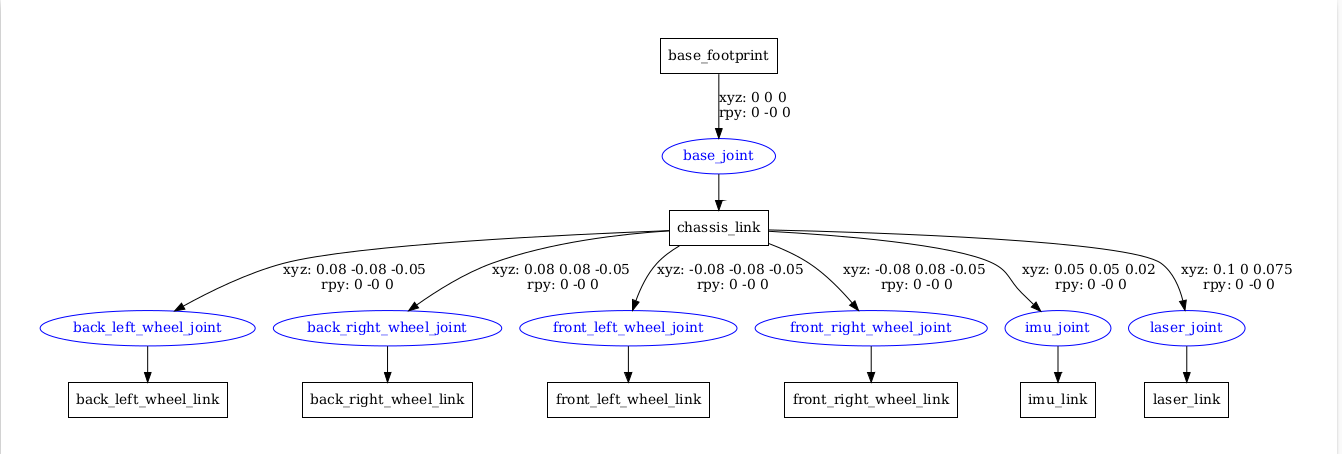
总结来说，<link>定义了机器人的各个物理部件，而<joint>定义了这些部件如何连接和相对于彼此的运动。固定关节（type="fixed"）意味着连接的部分是不动的，而连续关节（type="continuous"）允许连接的部分进行连续的旋转运动，这对于轮子来说是必要的。

查看URDF结构：

$cd fishbot\_ws/src/fishbot\_description/urdf

$urdf\_to\_graphviz fishbot\_base.urdf

运行成功后会产生一个pdf文件，打开之后就可以看到URDF模型分析的结果。



1. Gazebo

Gazebo是ROS系统中最为常用的三维物理仿真平台，支持动力学引擎，可以实现高质量的图形渲染，不仅可以模拟机器人及周边环境，还可以加入摩擦力、弹性系数等物理属性。

$ sudo apt install ros-humble-gazebo-\*

$ ros2 launch gazebo\_ros gazebo.launch.py

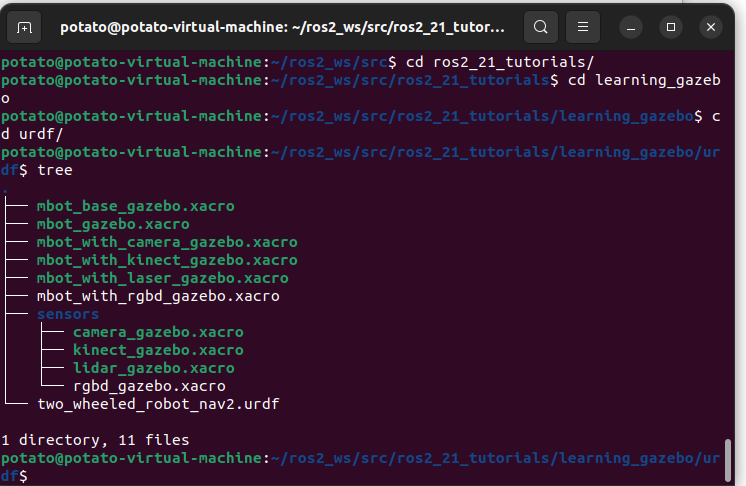
安装后之后要在启动配置文件里（.bashrc）中设置以下命令，因为在虚拟机下启动gazeb 的时候会有一点问题（硬盘不会），然后就强制用CPU来渲染gazebo里面的仿真。

export SVGA VGPU10=0

$ gazebo

此前设计好的小车模型或者机器人模型还不能直接放进gazebo里进行仿真，需要我们做一些优化：URDF文件格式的升级版本——XACRO文件。这里就不详细解释了，和URDF是同样的机制，它提供了一种更好的框架，加入了更多编程化的实现方法，可以让模型创建更友好。比如：

* 宏定义，一个小车有4个轮子，每个轮子都一样，我们就没必要创建4个一样的link，像[函数](https://marketing.csdn.net/p/3127db09a98e0723b83b2914d9256174?pId=2782&utm_source=glcblog&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/qq_51963216/article/details/_blank)定义一样，做一个可重复使用的模块就可以了。
* 文件包含，复杂机器人的模型文件可能会很长，为了切分不同的模块，比如底盘、传感器，我们还可以把不同模块的模型放置在不同的文件中，然后再用一个总体文件做包含调用。
* 可编程接口，比如在XACRO模型文件中，定义一些常量，描述机器人的尺寸，定义一些变量，在调用宏定义的时候传递数据，还可以在模型中做数据计算，甚至加入条件语句。



以mbot\_with\_camera\_gazebo.xacro为例，打开它的代码，我们可以发现会有这么两行

<xacro:include filename="$(find learning\_gazebo)/urdf/mbot\_base\_gazebo.xacro" />

<xacro:include filename="$(find learning\_gazebo)/urdf/sensors/camera\_gazebo.xacro" />

很明显，这里使用了嵌套，跟launch中的嵌套比较类似。

$ ros2 launch learning\_gazebo load\_urdf\_into\_gazebo.launch.py

$ ros2 run teleop\_twist\_keyboard teleop\_twist\_keyboard

1. Rviz2

Rviz是一个三维可视化显示平台，最直观的例子就是，他可以将仿真mbot摄像头捕获到的画面可视化出来。

Rviz的核心框架是基于Qt可视化工具打造的一个开放式平台，官方出厂就自带了很多机器人常用的可视化显示插件，只要我们按照ROS中的消息发布对应的话题，就可以看到图形化的效果了。如果我们对显示的效果不满意，或者想添加某些新的显示项，也可以在Rviz这个平台中，开发更多可视化效果，方便打造我们自己的上位机。

$ ros2 launch learning\_gazebo load\_mbot\_camera\_into\_gazebo.launch.py

$ ros2 run teleop\_twist\_keyboard teleop\_twist\_keyboard

$ ros2 run rviz2 rviz2  
 通常，我们在Rviz的仿真平台中，一般是在左侧的Displays窗口中添加我们所想要的选项。比如说之前呈现出来的Robot模型，刚刚呈现的Image，还有TF坐标等等。同样的功能还有很多可以选择，我们只需要去选择订阅的话题，就能看到对应的可视化图形。

