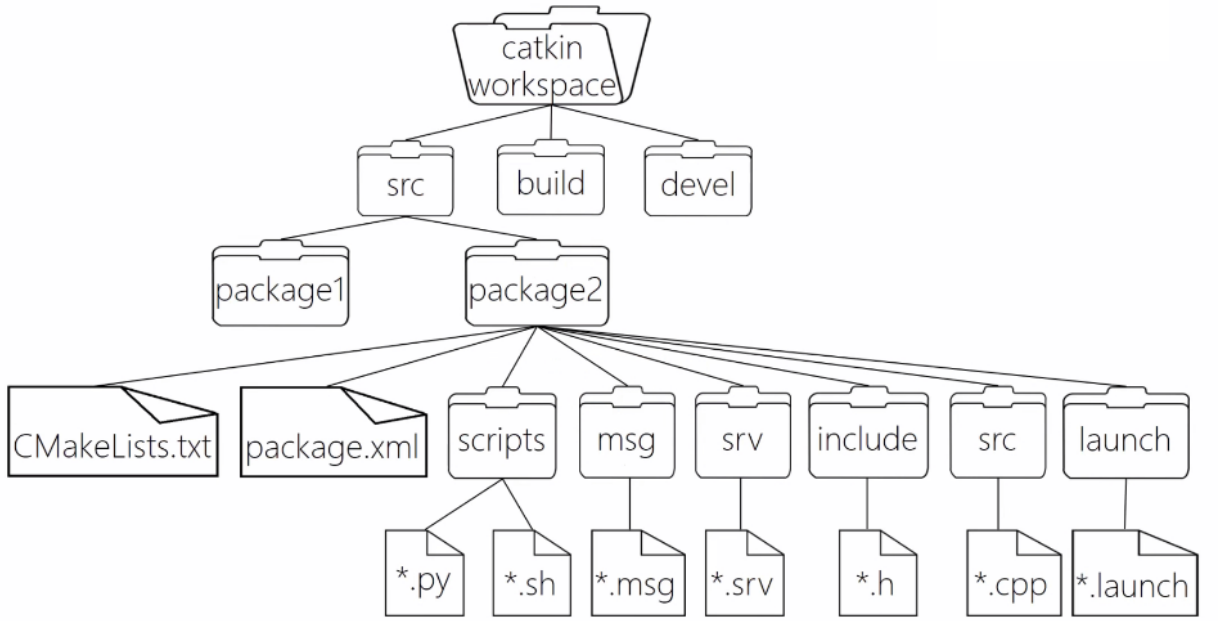
Turtorial2_ROS_part1

0.ROS项目工程结构 & 安装教程



- catkin_workspace : 一个ROS项目的工作空间, 即组织该ROS代码的地方
- src : 存放各种package的地方, 即真正写代码的地方
- build : 存放cmake&catkin缓存信息和中间文件
- devel : 存放目标文件 {头文件, 动态链接库, 静态链接库, 可执行文件}
- package : ROS软件的基本组织形式，catkin编译的基本单元，一个package可以包含多个可执行文件

ROS安装参考链接 :

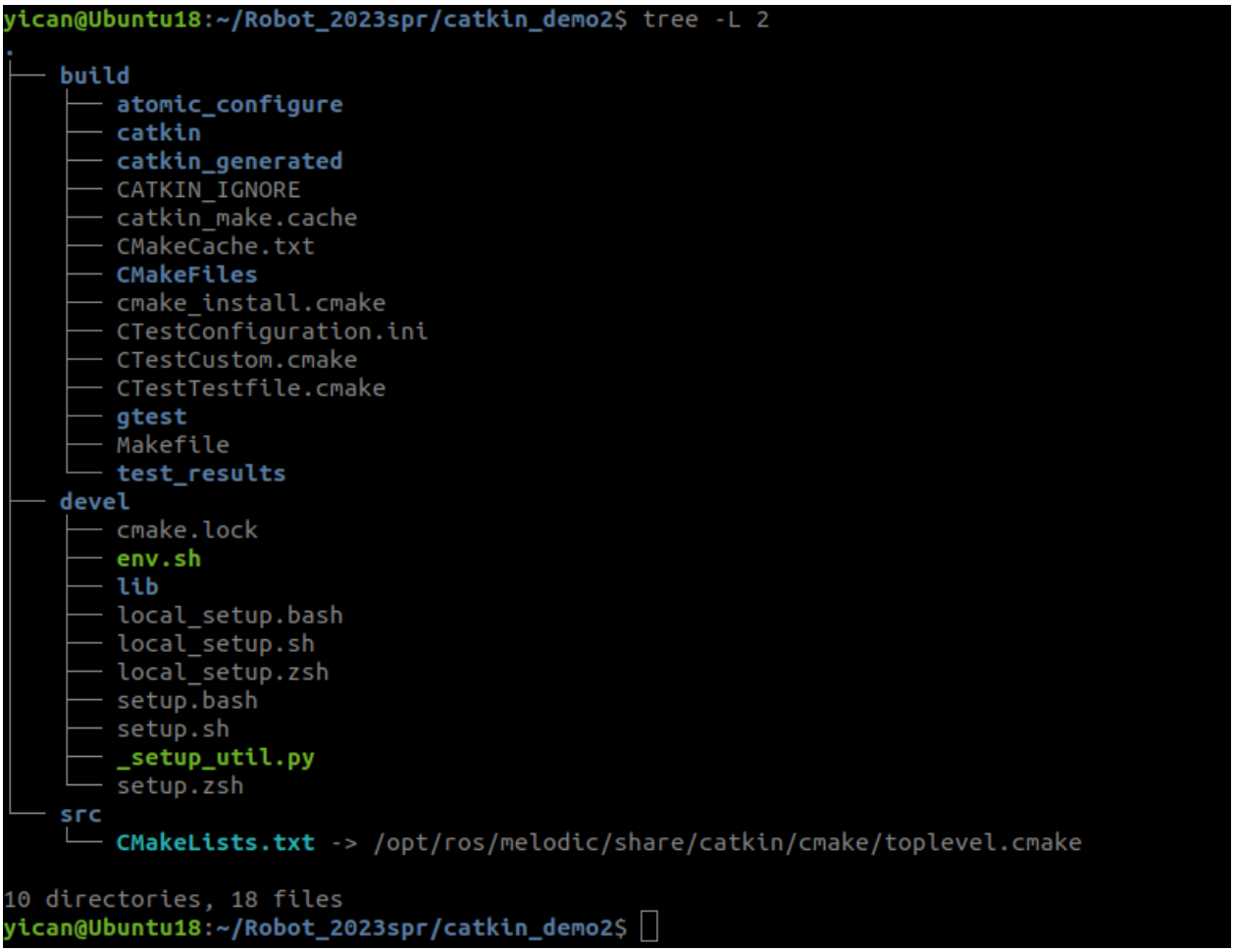
- ubuntu18.04 安装 ROS Melodic
- Ubuntu18.04+Ros Melodic安装及环境配置+Ros安装常见问题解决方法（亲测安装成功）
- ubuntu20.04 安装 ROS Noetic

1. ROS常见指令

- 建立工作空间 catkin_workspace初始化文件结构

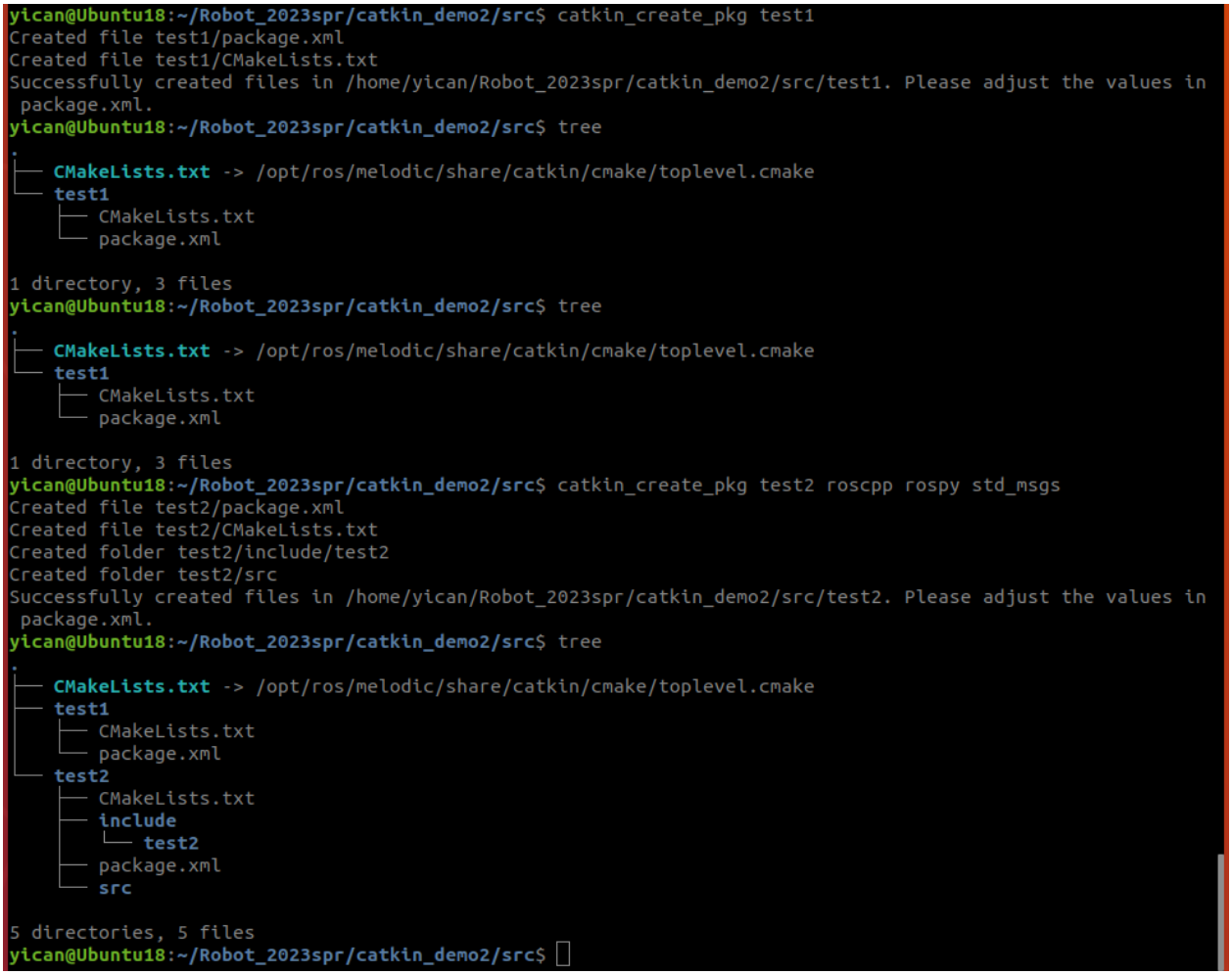
```
1  #建立工作
```

```
mkdir -p catkin_demo/src #-p 表示直接建立多层级目录 注:一定要有src文件,否则编译无法通过
cd catkin_rosdemo/
catkin_make #最主要的功能不是初始化工作空间而是编译 ,一般写完代码要catkin_make一下 (类似cmake) 这样系统就会自动帮我们构建生
source devel/setup.bash #把刚刚编译完的workspace给刷新到环境变量里面去,这样系统才知道我们生成的ros可执行文件放在哪里,才能找到运行
tree -L 2 #查看文件结构 sudo apt install tree 安装tree工具
```



- 创建package及ros常用指令

```
cd src
catkin_creat_pkg test1 # 创建名为test1的package
catkin_creat_pkg test2 roscpp rospy std_msgs # 创建名为test2的并引入cpp,py,std_msgs依赖
```




```
# 其他常用命令
rospack find test1 # # 查找某个pkg的地址 该工作空间需catkin_make + source /devel/setup.bash 才能定位到该功能包
rospack list # 列出本地所有pkg
roscd test1 # 跳转到某个pkg路径下
rosls test1 # 列举某个pkg下的文件信息
rosed test1 package.xml # 编辑pkg中的文件 同vim
rosdep install test1 # 安装某个pkg所需的依赖
```

2.ROS的第一个helloworld程序

1) 首先新建一个文件夹 catkin_ws ,初始化工作区 catkin_init_workspace, 下面就可以在这个工程中编写代码了,此时新建一个 src 文件夹用于存放自己的功能包。(参考上述的指令)

2) 在该功能包的 src 文件夹中,新建一个 hello.cpp 文件,用于编写代码。

```
#include "ros/ros.h"
int main(int argc, char **argv)
```

```
{
    ros::init(argc, argv, "hello");
    ROS_INFO("hello world!");
    return 0;
}
```

3) 接着修改 CMakeLists.txt ,将 src/hello.cpp 编译成一个名为 hello 的可执行文件。其中包含了依赖的功能包 roscpp ,然后需要将该功能包链接到我们的 hello 可执行文件上。

```
cmake_minimum_required(VERSION 3.0.2)
project(hello)
find_package(catkin REQUIRED COMPONENTS
roscpp
)
catkin_package()
include_directories(
# include
${catkin_INCLUDE_DIRS}
)
add_executable(hello src/hello.cpp)
target_link_libraries(hello ${catkin_LIBRARIES})
```

4) 输出结果：

```
like@like-Inspiron-7557:~/catkin_ws$ rosrun hello hello
[ INFO] [1638878081.443561570]: hello world!
```

3.ROS通信架构 & master, node , topic , message

ROS通信架构包括各种数据的处理，进程的运行，消息的传递等等。

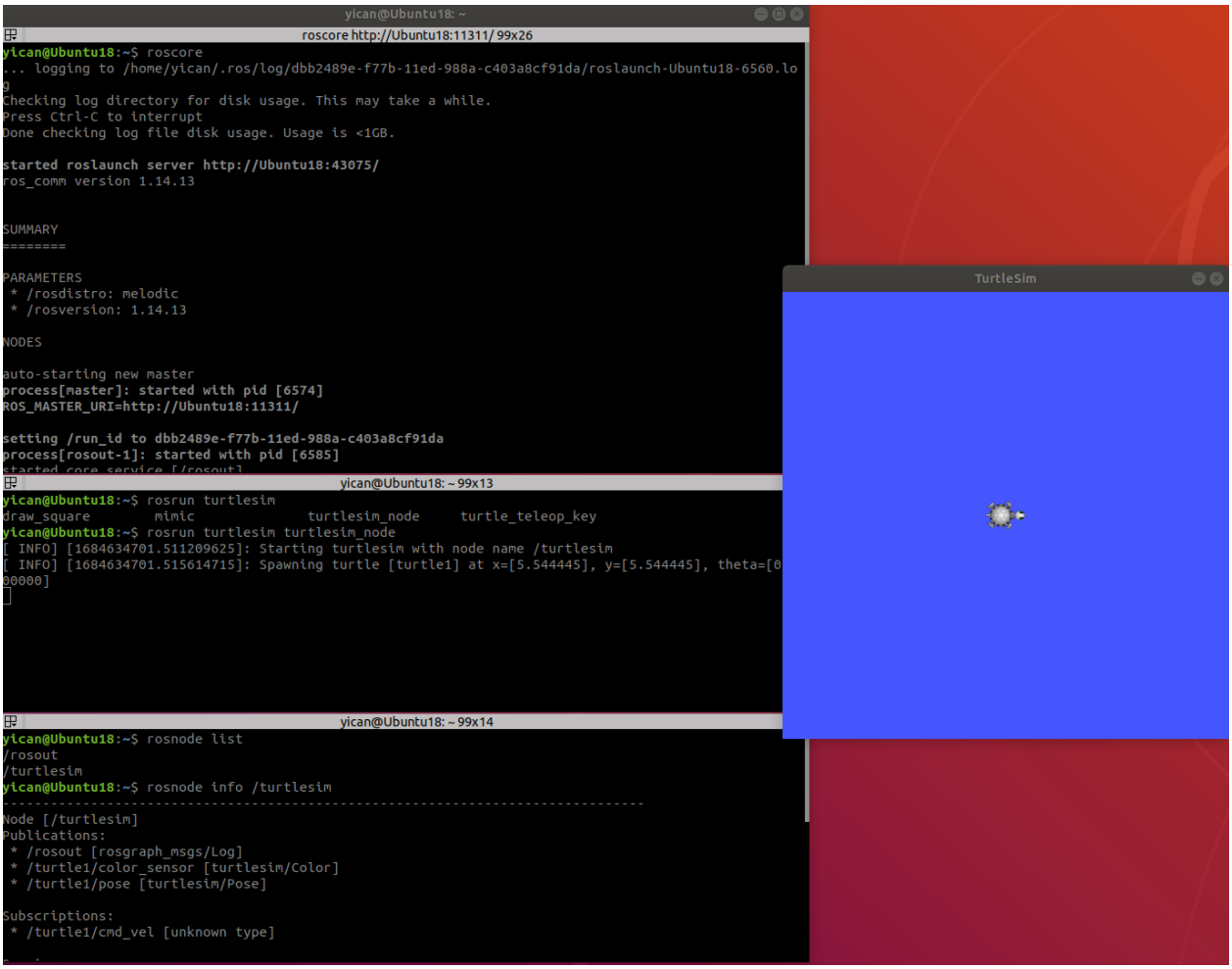
1) Node

- 在一个ROS的工程代码中, 最小的进程单元就是节点(node)
- 一个软件包里可以有多个可执行文件,可执行文件在运行之后就成了一个进程(process),这个进程在ROS中就叫做节点
- 从程序角度来说,node就是一个可执行文件(通常为C++编译生成的可执行文件、Python脚本)被执行，加载到了内存之中

2) Master

- master在整个网络通信架构里相当于管理中心，管理着各个node。
- node首先在master处进行注册，之后master会将该node纳入整个ROS程序中。
- node之间的通信也是先由master进行“牵线”，才能两两的进行点对点通信。
- 当ROS程序启动时，第一步首先启动master，由节点管理器处理依次启动node。

```
roscore # 启动ros master
# 第二步 启动具体的ros程序 也就是node 这个层级的概念
rosrun [pkg_name] [node_name]
rosnode list # 用来查看当前运行的node信息
rosnode info [node_name] # 显示某个node的详细信息
rosnode kill [node_name] # 结束某个node
# 小海龟例程
roscore
rosrun turtlesim turtlesim_node
rosnode list
rosnode info /turtlesim
```



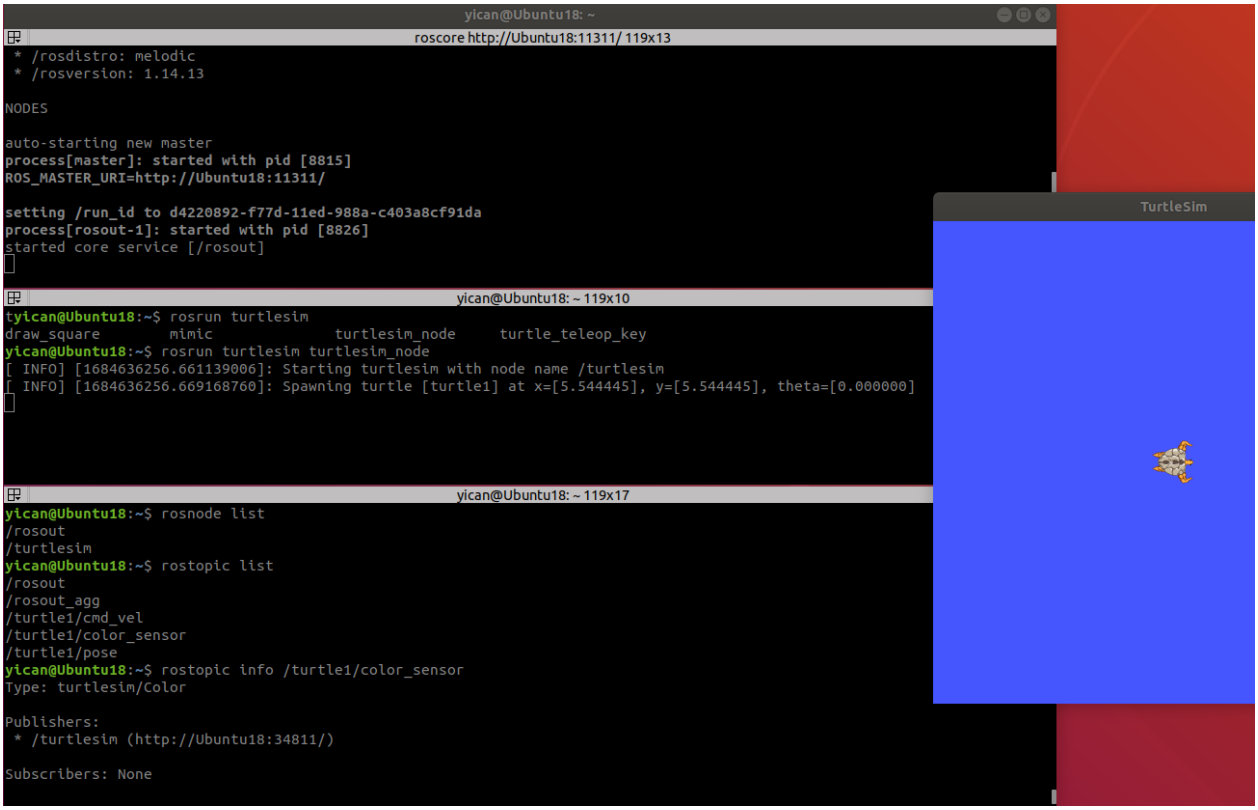
3) Topic (ROS提供最**核心**的**通信方式**之一)

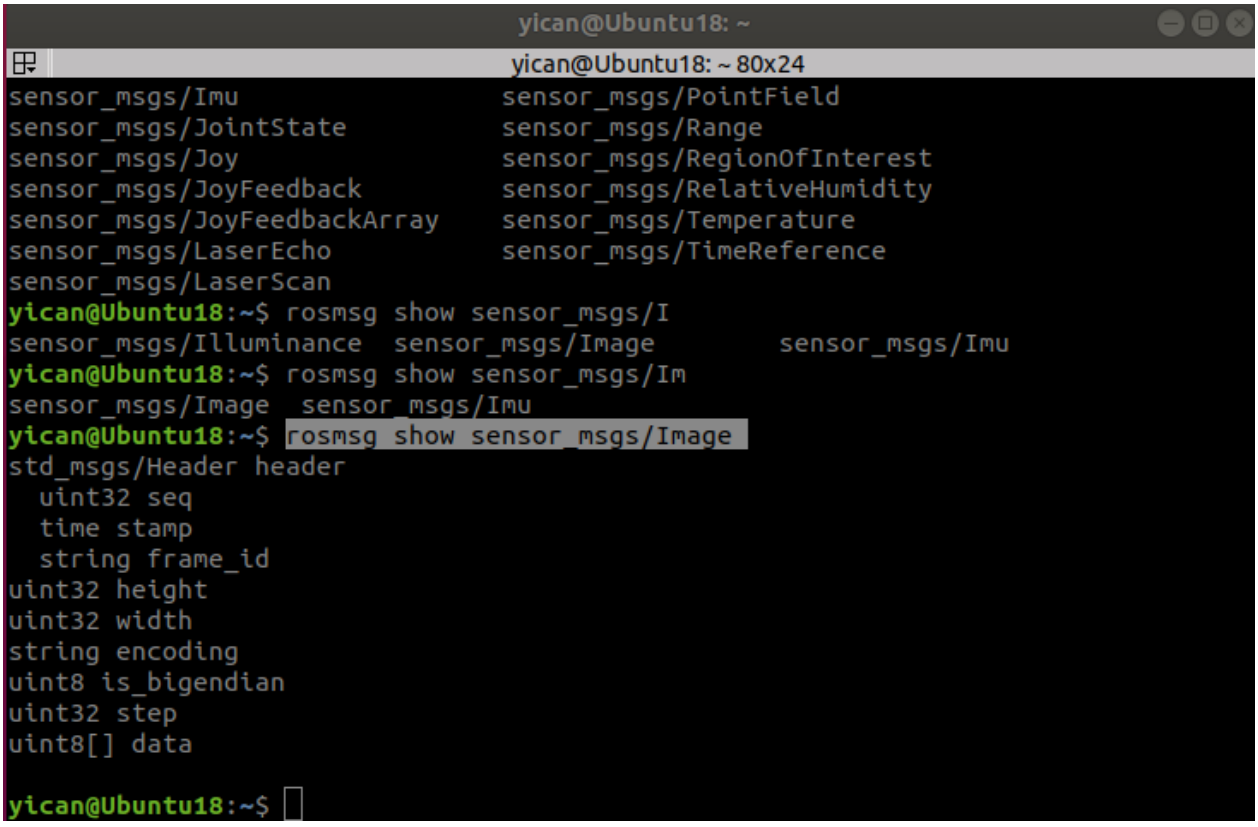
- ROS中的通信方式中，topic是常用的一种。
- 对于实时性、周期性的消息，使用topic来传输是最佳的选择。
- topic是一种点对点的单向通信方式，这里的“点”指的是node，也就是说node之间可以通过topic方式来传递信息。

4) Message

- topic有很严格的格式要求，比如上节的摄像头进程中的rgb图像topic，它就必然要遵循ROS中定义好的rgb图像格式，这种数据格式就是Message。
- Message按照定义解释就是topic内容的数据类型，也称之为topic的格式标准。这里和平常用到的Massage直观概念有所不同，这里的Message不单单指一条发布或者订阅的消息，也指定为topic的格式标准。

```
# rostopic rosmsg 相关命令
rostopic list # 列出当前所有topic
rostopic info /topic_name # 显示某个topic的属性信息
rostopic echo /topic_name # 显示某个topic的内容
rostopic pub /topic_name ... # 向某个topic发布内容
rosmsg list # 列出系统上所有的msg
rosmsg show /msg_name # 显示某个msg内容
# 小海龟例程
roscore
rosrun turtlesim turtlesim_node
rosnode list
rostopic list
rostopic info /turtle1/color_sensor
rostopic echo /turtle1/color_sensor
# .msg格式的查看
rosmsg show sensor_msgs/Image
```


yican@Ubuntu18: ~
roscore http://Ubuntu18:11311/ 119x13
* /rosdistro: melodic
* /rosversion: 1.14.13
NODES
auto-starting new master
process[master]: started with pid [8815]
ROS_MASTER_URI=http://Ubuntu18:11311/
setting /run_id to d4220892-f77d-11ed-988a-c403a8cf91da
process[rosout-1]: started with pid [8826]
started core service [/rosout]
yican@Ubuntu18: ~ 119x10
yican@Ubuntu18:~$ rosrun turtlesim
draw_square mimic turtlesim_node turtle_teleop_key
yican@Ubuntu18:~$ rosrun turtlesim turtlesim_node
[INFO] [1684636256.661139006]: Starting turtlesim with node name /turtlesim
[INFO] [1684636256.669168760]: Spawning turtle [turtle1] at x=[5.544445], y=[5.544445], theta=[0.000000]
yican@Ubuntu18: ~ 119x17
yican@Ubuntu18:~$ rosnode list
/rosout
/turtlesim
yican@Ubuntu18:~$ rostopic list
/rosout
/rosout_agg
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
yican@Ubuntu18:~$ rostopic info /turtle1/color_sensor
Type: turtlesim/Color
Publishers:
* /turtlesim (http://Ubuntu18:34811/)
Subscribers: None
TurtleSim


yican@Ubuntu18: ~
yican@Ubuntu18: ~ 80x24
sensor_msgs/Imu
sensor_msgs/JointState
sensor_msgs/Joy
sensor_msgs/JoyFeedback
sensor_msgs/JoyFeedbackArray
sensor_msgs/LaserEcho
sensor_msgs/LaserScan
sensor_msgs/PointField
sensor_msgs/Range
sensor_msgs/RegionOfInterest
sensor_msgs/RelativeHumidity
sensor_msgs/Temperature
sensor_msgs/TimeReference
yican@Ubuntu18:~$ rosmsg show sensor_msgs/I
sensor_msgs/Illuminance sensor_msgs/Image sensor_msgs/Imu
yican@Ubuntu18:~$ rosmsg show sensor_msgs/Im
sensor_msgs/Image sensor_msgs/Imu
yican@Ubuntu18:~$ rosmsg show sensor_msgs/Image
std_msgs/Header header
uint32 seq
time stamp
string frame_id
uint32 height
uint32 width
string encoding
uint8 is_bigendian
uint32 step
uint8[] data
yican@Ubuntu18:~$