

## 一有关ros节点：

- 1.GMapping:有关建图
- 2.Move\_base: 有关导航
- 3.wpb\_core:控制地盘
- 4.rplidar\_ros:有关驱动雷达

二

/opt/ros/noetic/share下面的都是ros的包

怎么判断的呢？ 答：在ros中package.xml就是一个软件包，也就是需要执行文件

三

老师的板书



- 1、使用catkin\_create\_pkg创建一个软件包
- 2、在软件包的src文件夹下创建一个节点的cpp源码文件
- 3、在节点的源码文件中include包含ROS的头文件
- 4、构建一个main函数，并在函数的开头执行ros::init()
- 5、构建while循环，循环条件为ros::ok()
- 6、在CMakeLists.txt中设置节点源码的编译规则
- 7、编译运行



1. 话题Topic是节点间进行**持续通讯**的一种形式。
2. 话题通讯的两个节点通过**话题的名称**建立起话题通讯连接。
3. 话题中通讯的数据，叫做消息Message。
4. 消息Message通常会按照一定的频率**持续不断**的发送，以保证消息数据的实时性。
5. 消息的**发送方**叫做话题的发布者Publisher。
6. 消息的**接收方**叫做话题的订阅者Subscriber。

## Q&A

- 1.发布者，订阅者，话题，节点之间的关系



ChatGPT

在ROS (Robot Operating System) 中，发布者 (Publisher) 、订阅者 (Subscriber) 、节点 (Node) 和话题 (Topic) 是核心概念，它们一起构成了ROS通信系统。下面是它们之间的关系：

1. 节点 (Node) :

- 节点是ROS中的基本运行单元，可以是一个可执行文件，也可以是一段代码。每个节点都有一个唯一的名称。
- 节点是ROS系统中的独立进程，可以运行在不同的计算机上。

2. 发布者 (Publisher) :

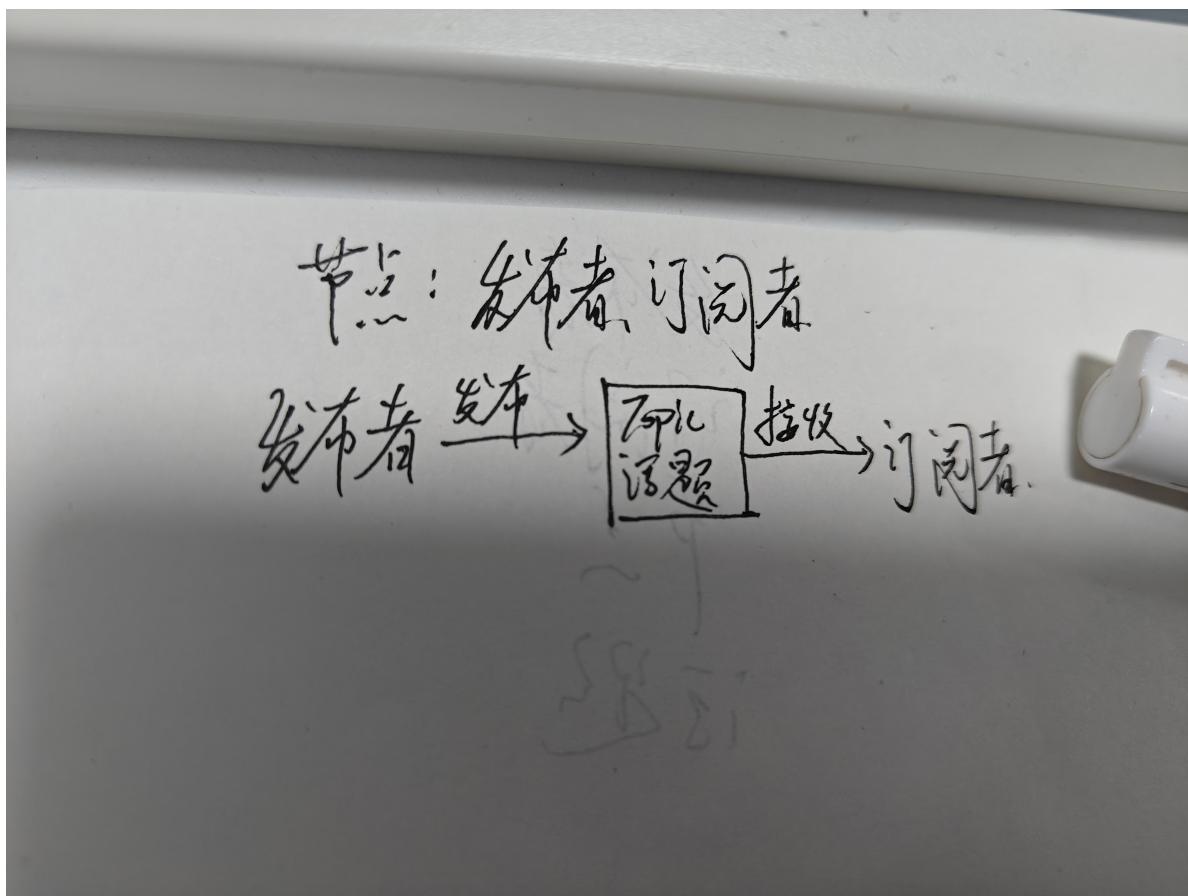
- 发布者是一个节点，负责将数据发布到ROS网络。
- 发布者创建一个话题，并将数据发送到该话题。其他节点可以订阅这个话题以接收数据。

3. 订阅者 (Subscriber) :

- 订阅者也是一个节点，负责接收从发布者发送的数据。
- 订阅者通过订阅特定的话题来接收数据。当发布者发布新数据时，所有订阅该话题的节点都会收到更新。

4. 话题 (Topic) :

- 话题是消息传递的通道，是发布者和订阅者之间进行通信的中介。
- 发布者通过发布到特定的话题，而订阅者通过订阅相同的话题来接收数据。



#### 四. 创建第一个发布者

```
wen_node.cpp  hao_node.cpp  my.launch  tf_node.cpp  CMakeLists.txt  wr_pkg  CMakeLists.txt  ssr_pkg
c> ssr_pkg > src > wen_node.cpp > main(int, char *[])
1 #include<ros/ros.h>
2 #include<std_msgs/String.h>
3 int main(int argc, char *argv[])
4 {
5     ros::init(argc, argv, "wen_node");           //初始化ros
6
7     printf("Hello,World!\n");
8
9     ros::NodeHandle nh;    //创建一个ros “大管家”，帮助进行操作
10    ros::Publisher haha=nh.advertise<std_msgs::String>("my_first_test",5); //创建发布者进行发布信息
11
12    ros::Rate loop_rate(10);   //每秒发布间隔
13
14    while (ros::ok())
15    {
16        std_msgs::String msg; //创建信息对象
17        msg.data="你好";      //数据
18        haha.publish(msg);   //开始发布
19        loop_rate.sleep();   //间隔
20    }
21
22    return 0;
23 }
24
```

## 五. 创建订阅者

```
c> atr_pkg > src > li_node.cpp > main(int, char *[])
1 #include<ros/ros.h>
2 #include<std_msgs/String.h>
3
4 void wen_callback(std_msgs::String msg)
5 {
6     ROS_INFO(msg.data.c_str());
7     printf("\n");
8 }
9
10 void hw_callback(std_msgs::String msg)
11 {
12     ROS_WARN(msg.data.c_str());
13     printf ("\n");
14 }
15
16 int main(int argc, char *argv[])
17 {
18     ros::init(argc, argv, "li_node");
19
20     ros::NodeHandle nh;
21     ros::Subscriber sub=nh.subscribe("my_first_test",5,wen_callback); //订阅者，函数：话题名称 话题缓存长度 回调函数
22     ros::Subscriber sub_2=nh.subscribe("my_second_test",5,hw_callback);
23
24     while (ros::ok())
25     {
26         ros::spinOnce(); //可接收多个发布者的数据，防止有遗漏
27     }
28
29     return 0;
30 }
```

顶部 换行 回退到后台 终端 端口

## 六. 整体框架

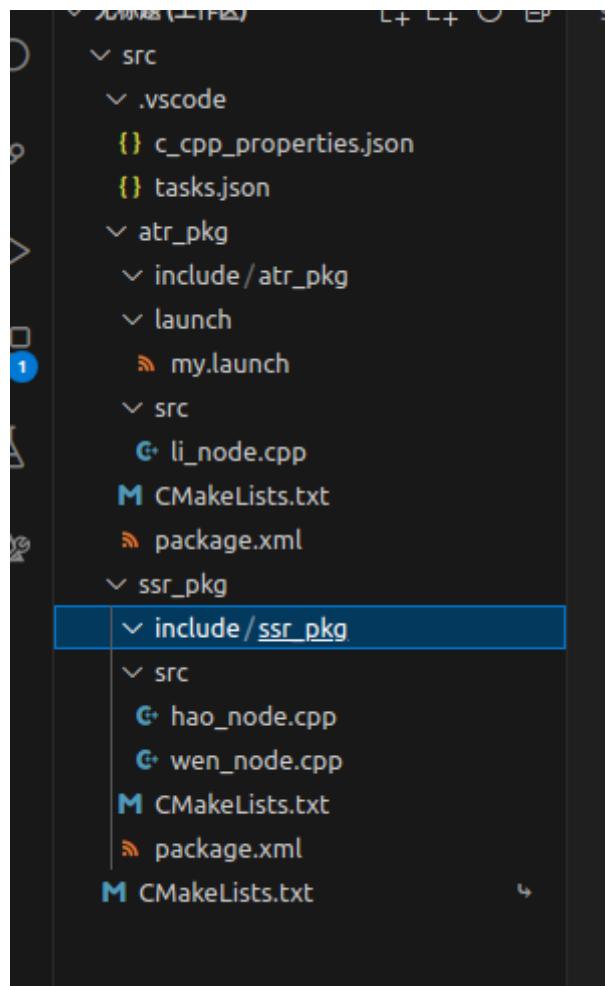
对于我的来说：

节点：

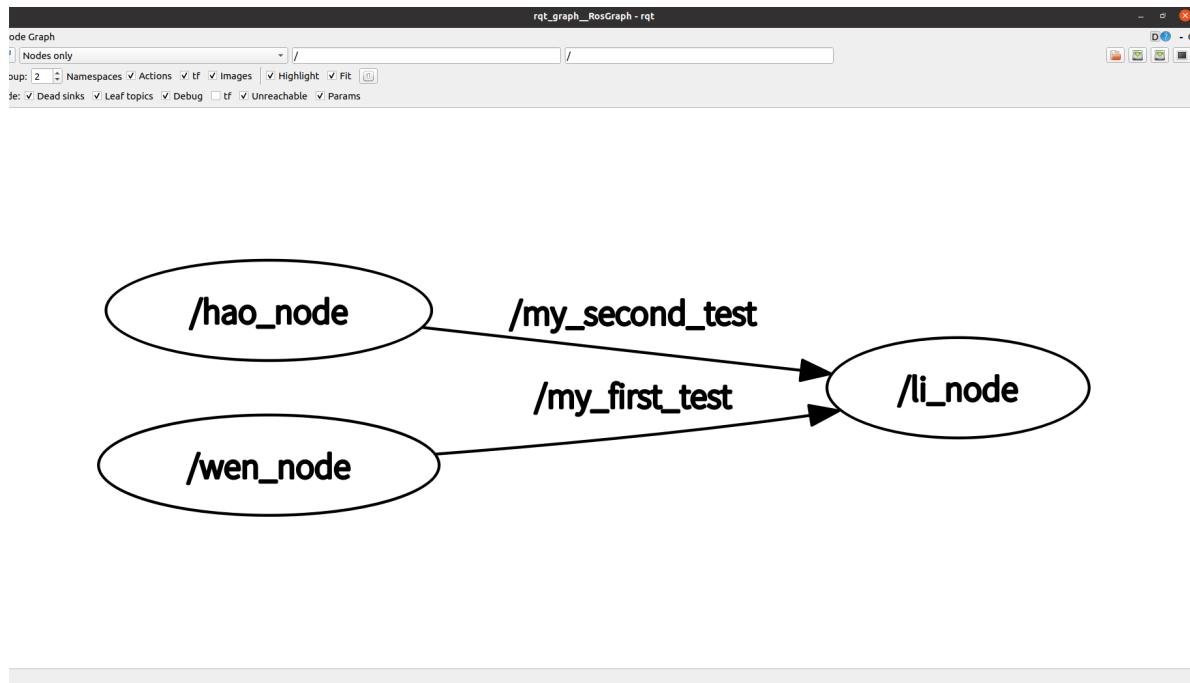
发布者： hao\_node,wen\_node

订阅者： li\_node

话题： my\_first\_test,my\_second\_test



对于文件名可以随意起，针对我的节点之间的关系：



## 七.launch的配置

## Q&A:

为什么要有launch文件?

因为运行一个节点需要运行一下指令 (拿我的例子说) :

1.roscore

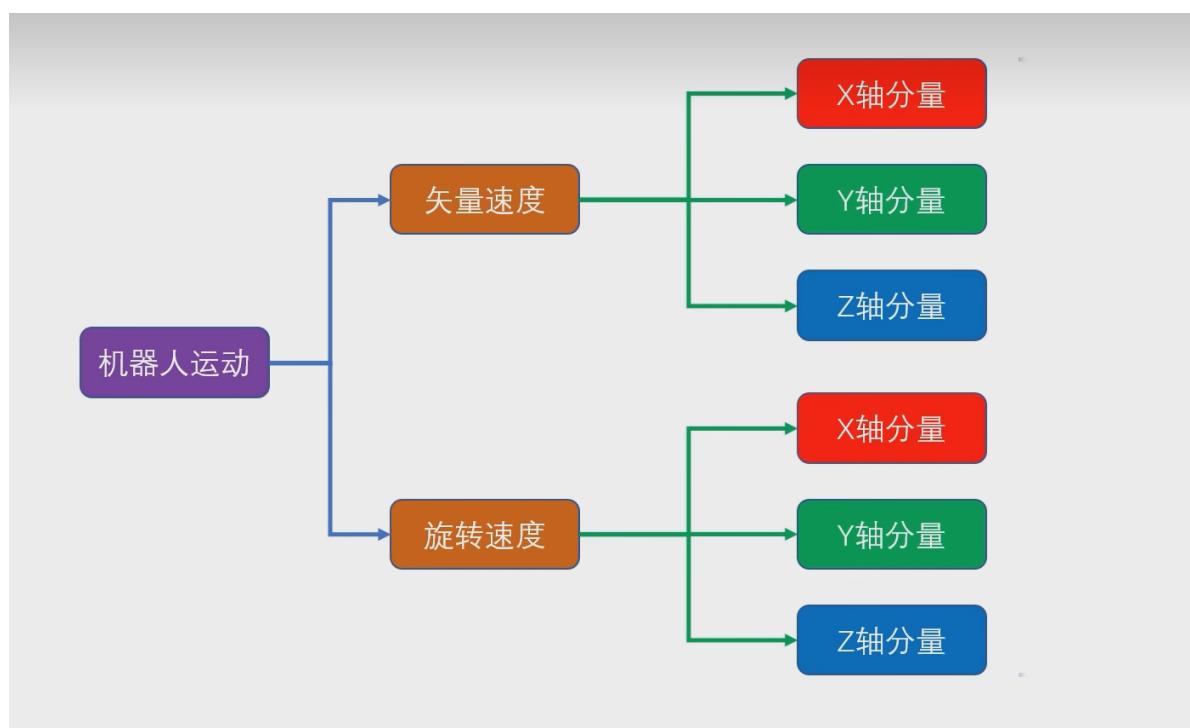
2.roslaunch ssr\_pkg wen\_node

当一个项目有很多节点时运行起来是很麻烦的, 所以有了launch文件, 可以进行一次性指令进行编译

## 八.有关机器人运动

矢量速度是m/s

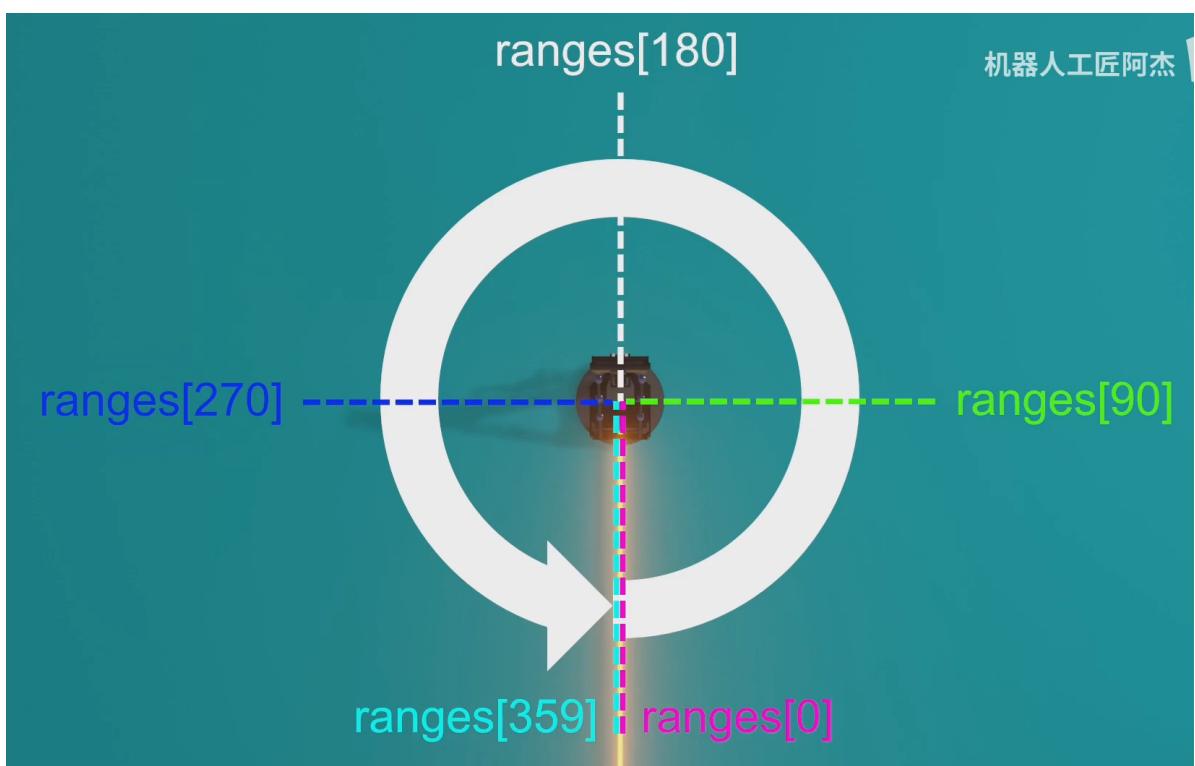
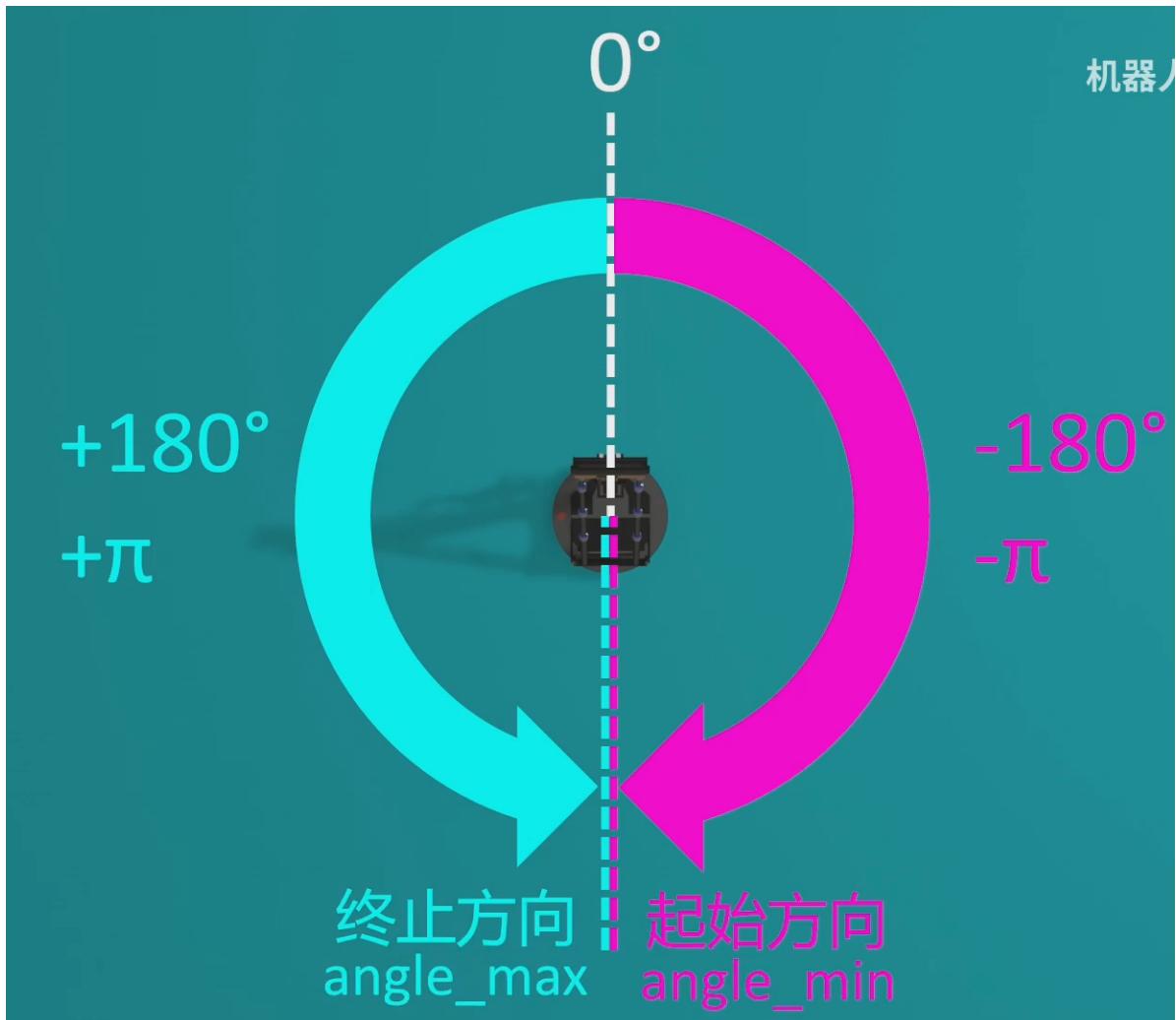
旋转速度是rad/s



## 小tips:

查看雷达信息

```
robot@WP:~$ rostopic echo /scan --noarr
```



创建ros包：

## a.catkin\_create\_pkg 包名 配置文件

e.g.:

```
catkin_create_pkg lidar_pkg rospy roscpp sensor_msg
```

b.关于消息发送：

std\_msgs::String 表示发送字符串信息

geometry\_msgs/Twist 表示发送速度消息

sensor\_msgs/LaserScan 表示发送雷达测距消息

c.有关imu的话题

- imu/data\_raw (sensor\_msgs/Imu)
  - **加速度计输出的矢量加速度 和 陀螺仪输出的旋转角速度。**
- imu/data (sensor\_msgs/Imu)
  - **/imu/data\_raw的数据 再加上 融合后的四元数姿态描述。**
- imu/mag (sensor\_msgs/MagneticField)
  - **磁强计输出磁强数据。**