

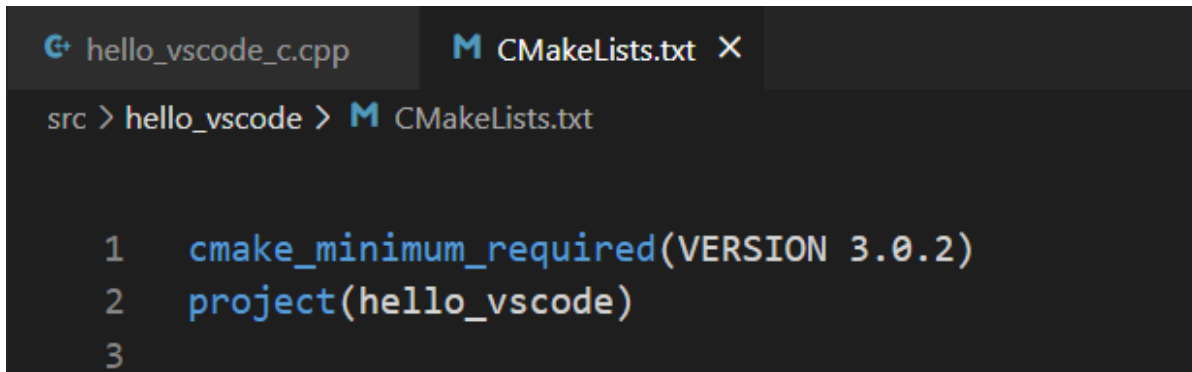
CMakeList

最终执行指令：

```
roslaunch hello_vscode hello_vscode_c
```

分解：

`hello_vscode` project name, 位于CMakeLists.txt 最上端,又成软件包名（最外层src下的文件夹）



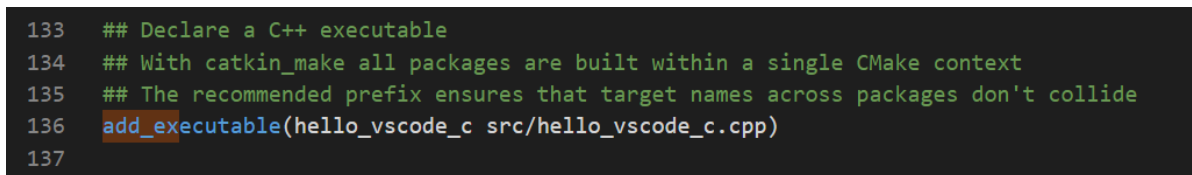
```
src > hello_vscode > CMakeLists.txt

1  cmake_minimum_required(VERSION 3.0.2)
2  project(hello_vscode)
3
```

`hello_vscode_c` 可执行目标：

对C++, 只包含文件名前缀

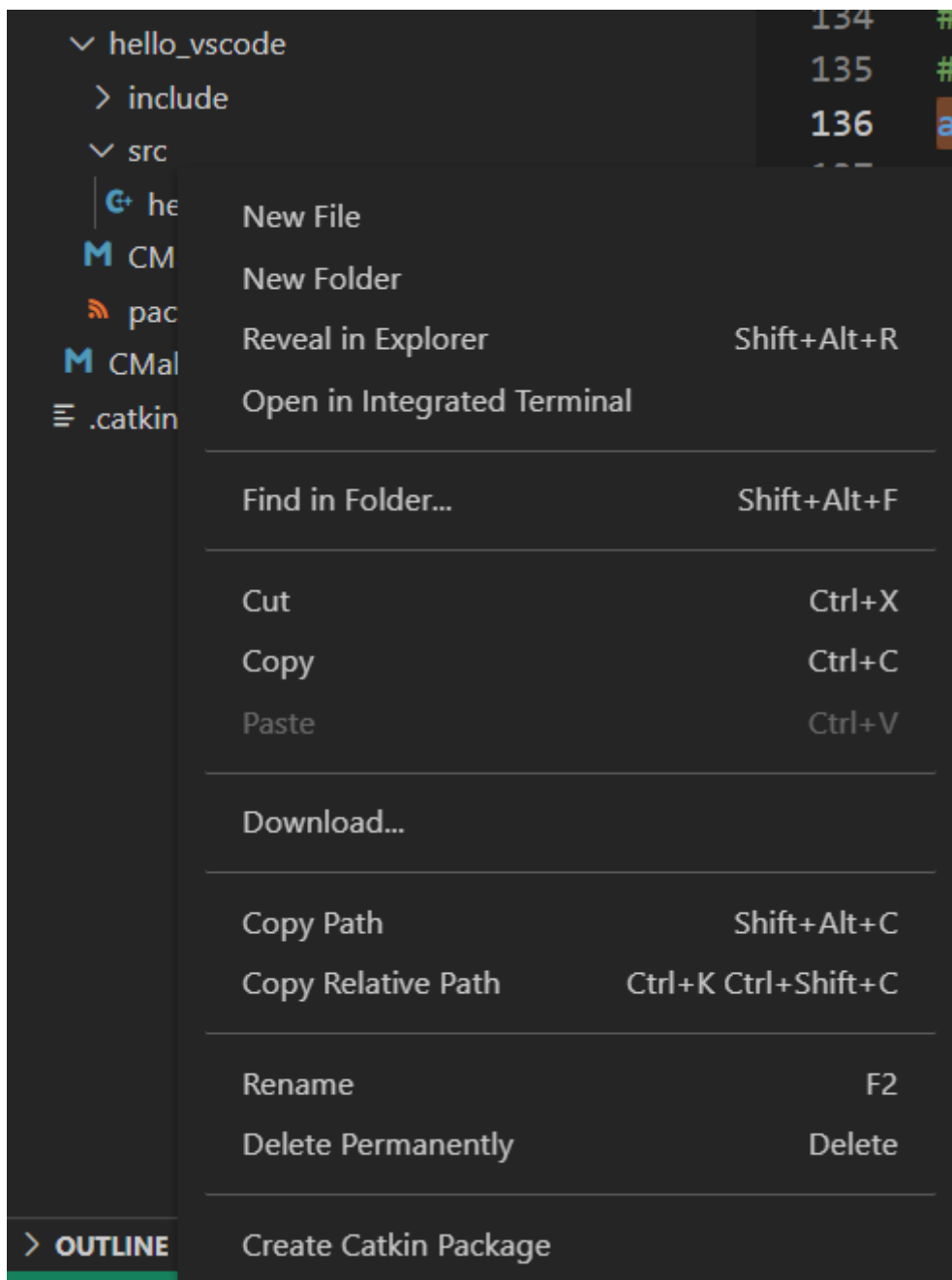
对python, 使用全称, 包含后缀



```
133  ## Declare a C++ executable
134  ## With catkin_make all packages are built within a single CMake context
135  ## The recommended prefix ensures that target names across packages don't collide
136  add_executable(hello_vscode_c src/hello_vscode_c.cpp)
137
```

add_executable(可执行目标, 源文件)

ROS插件用处：



选择src文件夹，右击，选择 Create Catkin Package 生成CMakeList.txt

roslaunch运行文件，必须在运行在项目文件下路径下，不能在子文件夹内。

c++ 运行修改CMakeList.txt:

```
add_executable(步骤3的源文件名
  src/步骤3的源文件名.cpp
)
target_link_libraries(步骤3的源文件名
  ${catkin_LIBRARIES}
)
```

python运行修改CMakeList.txt:

```
catkin_install_python(PROGRAMS scripts/自定义文件名.py
  DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
)
```

C++ 中文乱码

```
setlocale(LC_ALL, "");
```

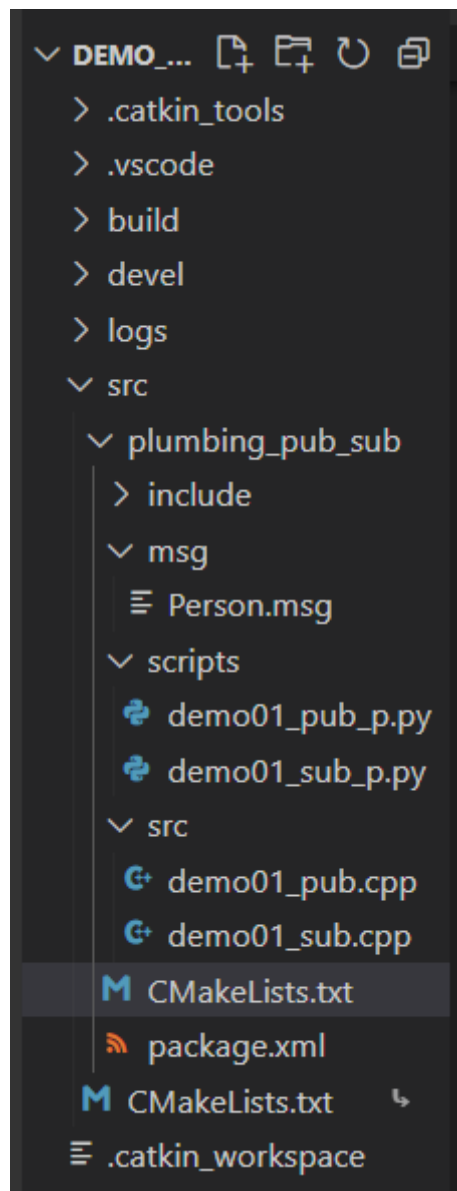
或者

```
setlocale(LC_CTYPE, "zh-CN.utf8");
```

 加入main函数中

在每次新开终端时，记得执行source，不然无法运行

CMakeList内部联系



在这样的文件列表中，我们的 **功能包** 是 plumbing_pub_sub

在执行指令中也有提示：`roslaunch plumbing_pub_sub demo01_pub_p.py`

功能包 依赖于 find_package，编译时依赖

```
10 find_package(catkin REQUIRED COMPONENTS
11     roscpp
12     rospy
13     std_msgs
14     message_generation
15 )
```

find_package 又依赖于catkin_package，运行时依赖

```
105 catkin_package(
106     # INCLUDE_DIRS include
107     # LIBRARIES plumbing_pub_sub
108     CATKIN_DEPENDS roscpp rospy std_msgs message_runtime
109     # DEPENDS system_lib
110 )
111
```

ROS通信

基础通信

C++

发布消息

```
ros::NodeHandle nh;
ros::Publisher pub = nh.advertise<std_msgs::String>("chatter",10);
//添加延时，使其完全注册topic，以免丢失
ros::Duration(3.0).sleep();
//chatter 是topic 话题

pub.publish(msg);
```

```
$ rostopic echo chatter 可查看信息
```

接受信息：

```

void doMsg(const std_msgs::String::ConstPtr &msg)
{
    ROS_INFO("我听见:%s",msg->data.c_str());
}

ros::NodeHandle nh;
//
topic, 消息列表, 回调函数
ros::Subscriber sub = nh.subscribe("chatter",10,doMsg);
// 循环接受
ros::spin();

```

python

发布消息:

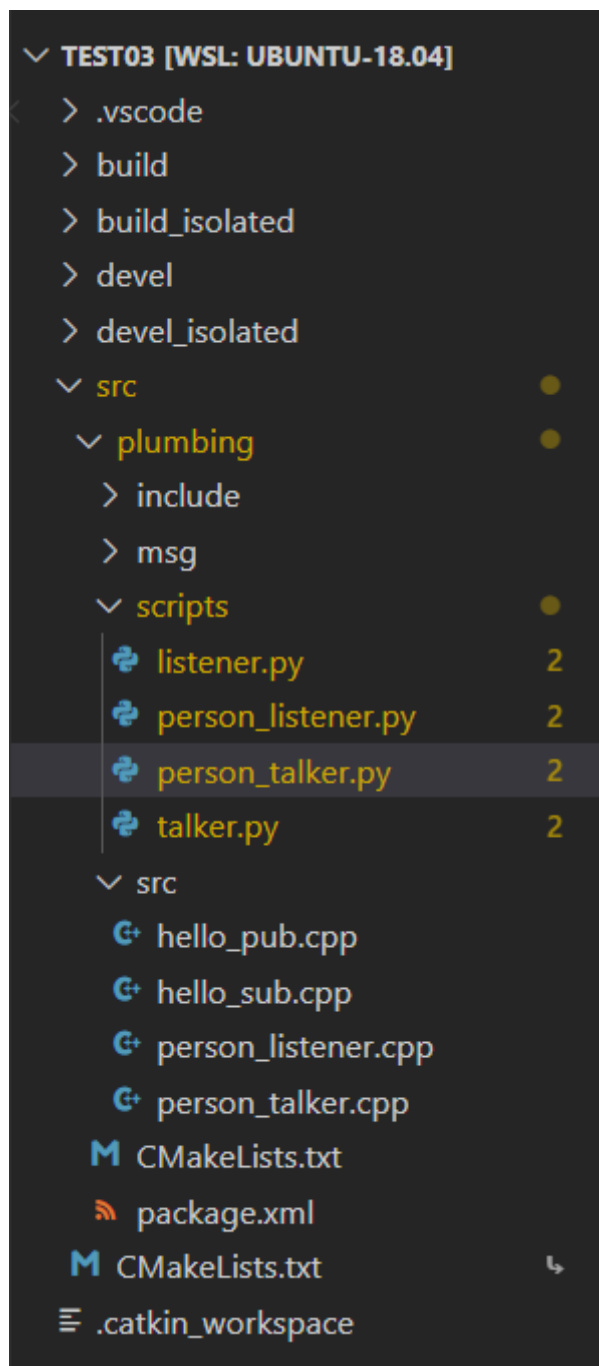
```

#!/usr/bin/env python

import rospy
from std_msgs.msg import String
if __name__ == "__main__":
    rospy.init_node("sanDai")
    # param:topic data_class, 消息队列（超出指定值，则清理旧数据）
    pub = rospy.Publisher("che",String,queue_size=10)
    # 初始化消息, 自带data属性
    msg = String()
    rate = rospy.Rate(1)
    while not rospy.is_shutdown():
        msg.data = "hello"
        pub.publish(msg)
        rate.sleep()
        rospy.loginfo("写出的数据:%s",msg.data)

```

msg 调用



添加cpp、py文件后需要修改 CMakeLists.txt

- ▶ 针对C++需要修改如下这些内容：
- ▶ 针对python，需要修改如下

参数服务器

在 roscpp 中提供了两套 API 实现参数操作

ros::NodeHandle

setParam("键", 值)

ros::param

set("键", "值")

示例: 分别设置整形、浮点、字符串、bool、列表、字典等类型参数

修改(相同的键, 不同的值)

set增加参数

```
//NodeHandle-----  
ros::NodeHandle nh;  
nh.setParam("nh_int",10); //整型  
//param-----  
ros::param::set("param_int",20);  
ros::param::set("param_double",3.14);
```

get 查询参数

```
//nh  
//1  
nh.getParam("nh_int",nh_int_value);  
//2  
nh.getParamCached("nh_int",nh_int_value);  
//3  
std::vector<std::string> param_names1;  
nh.getParamNames(param_names1);  
for (auto &&name : param_names1)  
{  
    ROS_INFO("名称解析name = %s",name.c_str());  
}  
//4  
ROS_INFO("存在 nh_int 吗? %d",nh.hasParam("nh_int"));  
//5  
std::string key;  
nh.searchParam("nh_int",key);  
ROS_INFO("搜索键:%s",key.c_str());  
// param  
//1  
ros::param::get("param_int",param_int_value);  
//2  
ros::param::getCached("param_int",param_int_value);  
//3  
std::vector<std::string> param_names2;  
ros::param::getParamNames(param_names2);  
for (auto &&name : param_names2)  
{  
    ROS_INFO("名称解析name = %s",name.c_str());  
}  
//4  
ROS_INFO("存在 param_int 吗? %d",ros::param::has("param_int"));  
//5  
std::string key;  
ros::param::search("param_int",key);  
ROS_INFO("搜索键:%s",key.c_str());
```

delete删除参数

```

ros::NodeHandle nh;
bool r1 = nh.deleteParam("nh_int");
ROS_INFO("nh 删除结果:%d",r1);

bool r2 = ros::param::del("param_int");
ROS_INFO("param 删除结果:%d",r2);

```

常用指令

roscnode

```

roscnode ping      测试到节点的连接状态
roscnode list      列出活动节点
roscnode info      打印节点信息
roscnode machine    列出指定设备上节点
roscnode kill      杀死某个节点
roscnode cleanup    清除不可连接的节点

```

rostopic

```

rostopic bw        显示主题使用的带宽
rostopic delay      显示带有 header 的主题延迟
rostopic echo       打印消息到屏幕
rostopic find       根据类型查找主题
rostopic hz         显示主题的发布频率
rostopic info       显示主题相关信息
rostopic list       显示所有活动状态下的主题
rostopic pub        将数据发布到主题
rostopic type       打印主题类型

```

```

#                pub      topic                两次TAB补齐
rostopic pub    /chatter_person  hellTopic/Person  "name: 'huluwa'
age: 8
height: 0.8"

```

```

# 每秒10次
rostopic pub -r 10 /chatter_person  hellTopic/Person "name: 'huluwa'
age: 5
height: 10"

```

rosmmsg

```

rosmmsg show      显示消息描述
rosmmsg info      显示消息信息
rosmmsg list      列出所有消息
rosmmsg md5       显示 md5 加密后的消息
rosmmsg package    显示某个功能包下的所有消息
rosmmsg packages  列出包含消息的功能包

```


rosservice

```
rosservice args 打印服务参数
rosservice call  使用提供的参数调用服务
rosservice find  按照服务类型查找服务
rosservice info  打印有关服务的信息
rosservice list  列出所有活动的服务
rosservice type  打印服务类型
rosservice uri   打印服务的 ROSRPC uri
```

rossrv

rossrv是用于显示有关ROS服务类型的信息的命令行工具，与 rosmmsg 使用语法高度雷同。

```
rossrv show 显示服务消息详情
rossrv info 显示服务消息相关信息
rossrv list  列出所有服务信息
rossrv md5   显示 md5 加密后的服务消息
rossrv package 显示某个包下所有服务消息
rossrv packages 显示包含服务消息的所有包
```

常用APIs

初始化

```
void init(int &argc, char **argv, const std::string& name, uint32_t options = 0);
```

```
// 节点可重复启动，设置ros::init_options::AnonymousName
// 在用户定义的节点名称加上随机数后缀，避免重复名称
ros::init(argc, argv, "NodeName", ros::init_options::AnonymousName)
```

话题与服务相关对象

```
/*
    latch设置为true的作用？
    以发布静态地图为例子：发布者对象的latch设置为True,每当订阅者连接时，发布者就会发布一次最新的数据，仅新连接时触发一次
*/
ros::Publisher pub = nh.advertise<std_msgs::String>("topicName",10,latch=True)

pub.publish(data)
```

回旋函数

1.spinOnce()

```
/**
 * \brief 处理一轮回调
 *
 * 一般应用场景：
 *     在循环体内，处理所有可用的回调函数
 *
 */
ROSCPP_DECL void spinOnce();
```

2.spin()

```
/**
 * \brief 进入循环处理回调
 */
ROSCPP_DECL void spin();
```

3.二者比较

相同点:二者都用于处理回调函数;

不同点:ros::spin() 是进入了循环执行回调函数，而 ros::spinOnce() 只会执行一次回调函数(没有循环)，在 ros::spin() 后的语句不会执行到，而 ros::spinOnce() 后的语句可以执行。

元功能包

实现

首先:新建一个功能包

然后:修改package.xml ,内容如下:

```
<exec_depend>被集成的功能包</exec_depend>
.....
<export>
  <metapackage />
</export>
```

最后:修改 CMakeLists.txt,内容如下:

```
cmake_minimum_required(VERSION 3.0.2)
project(demo)
find_package(catkin REQUIRED)
catkin_metapackage()
```

PS:CMakeLists.txt 中不可以有换行。

launch文件

弃用声明

- deprecated = "弃用声明"

告知用户当前 launch 文件已经弃用

```
<launch deprecated ="此文件废弃" >
</launch>
```

node

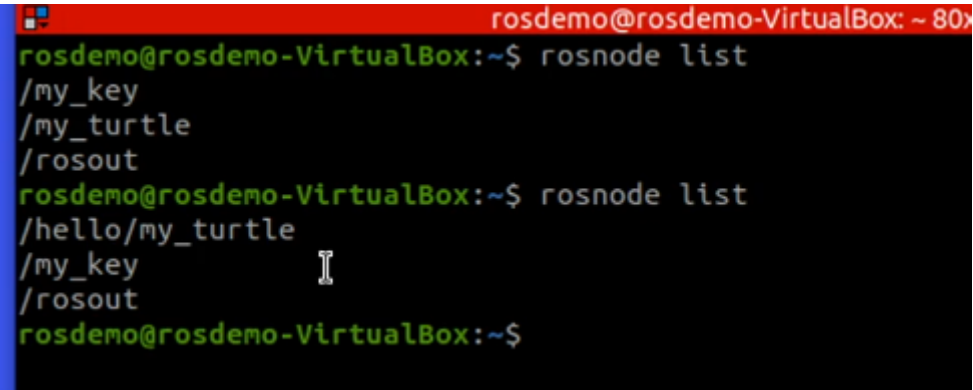
属性

- pkg="包名"
节点所属的包
- type="nodeType"
节点类型(与之相同名称的可执行文件)
- name="nodeName"
节点名称(在 ROS 网络拓扑中节点的名称)
- args="xxx xxx xxx" (可选)
将参数传递给节点
- machine="机器名"
在指定机器上启动节点
- respawn="true | false" (可选)
如果节点退出，是否自动重启

```
<node pkg="" type="" name="" respawn=" true"/ >
```

- respawn_delay=" N" (可选)
如果 respawn 为 true, 那么延迟 N 秒后启动节点
- required="true | false" (可选)
该节点是否必须，如果为 true,那么如果该节点退出，将杀死整个 roslaunch
- ns="xxx" (可选)
在指定命名空间 xxx 中启动节点

```
<node pkg="" type="" name="" ns="hello"/ >
```



```
rosdemo@rosdemo-VirtualBox: ~ 80x
rosdemo@rosdemo-VirtualBox:~$ roslaunch my_key
/my_key
/my_turtle
/rosout
rosdemo@rosdemo-VirtualBox:~$ roslaunch my_key hello
/hello/my_turtle
/my_key
/rosout
rosdemo@rosdemo-VirtualBox:~$
```

- clear_params="true | false" (可选)
在启动前，删除节点的私有空间的所有参数
- output="log | screen" (可选)
日志发送目标，可以设置为 log 日志文件，或 screen 屏幕,默认是 log

include

`include` 标签用于将另一个 xml 格式的 launch 文件导入到当前文件

属性

- `file="$(find 包名)/xxx/xxx.launch"`
要包含的文件路径
- `ns="xxx"` (可选)
在指定命名空间导入文件

```
<launch>
  <!-- <include file="$(find 功能包名)/launch/具体路径" /> -->
  <include file="$(find hello)/launch/start_turtle.launch" />
</launch>
```

remap

用于话题重命名

属性

- `from="xxx"`
原始话题名称
- `to="yyy"`
目标名称

```
<launch>
  <!-- 添加执行节点-->
  <!-- roslaunch turtlesim turtlesim_node-->
  <!-- <node pkg="hello" type="hello" name="hello" output="screen"/> -->
  <node pkg="turtlesim" type="turtlesim_node" name="turtle_GUI"
output="screen" >
  <!-- 当使用不同的话题时，通过remap修改话题将两个节点互通 -->
  <remap from="/turtle/cmd_vel" to="/cmd_vel" />
</node>
  <node pkg="turtlesim" type="turtle_teleop_key" name="turtle_key"
output="screen" />
</launch>
```

param&rosparam

`<param>` 标签主要用于在参数服务器上设置参数，参数源可以在标签中通过 `value` 指定，也可以通过外部文件加载，在 `<node>` 标签中时，相当于私有命名空间。

属性

- `name="命名空间/参数名"`
参数名称，可以包含命名空间
- `value="xxx"` (可选)
定义参数值，如果此处省略，必须指定外部文件作为参数源
- `type="str | int | double | bool | yaml"` (可选)
指定参数类型，如果未指定，roslaunch 会尝试确定参数类型，规则如下：

- 如果包含 '.' 的数字解析为浮点型，否则为整型
- "true" 和 "false" 是 bool 值(不区分大小写)
- 其他是字符串

```
<launch>
  <!-- 添加执行节点-->
  <!-- rosrun turtlesim turtlesim_node-->
  <!-- <node pkg="hello" type="hello" name="hello" output="screen"/> -->
  <param name="param_A" type="int" value="1000"/>
  <rosparam command="load" file="$(find hello)/launch/param.yaml" />
  <node pkg="turtlesim" type="turtlesim_node" name="turtle_GUI"
output="screen" >
  <!-- 当使用不同话题的节点时，通过remap修改话题将两个节点互通 -->
    <remap from="/turtle/cmd_vel" to="/cmd_vel" />
    <param name="param_B" type="double" value="1.5"/>
  </node>
  <node pkg="turtlesim" type="turtle_teleop_key" name="turtle_key"
output="screen" />
</launch>
```

注意param.yaml这里必须编码(中英文)正确，错误会标红，发生错误如下

```
RLException: error loading <rosparam> tag:
  'ascii' codec can't decode byte 0xef in position 71: ordinal not in
range(128)
XML is <rosparam command="load" file="$(find hello)/launch/param.yaml"/>
The traceback for the exception was written to the log file
```

```
<launch>
  <!-- 导出参数 -->
  <rosparam command="dump" file="$(find hello)/launch/params_out.yaml"/>
  <!-- 运行中删除参数 -->
  <rosparam command="delete" param="bg_B" />
</launch>
```

group

```

<launch>
  <!-- 添加执行节点-->
  <group ns="first">
    <node pkg="turtlesim" type="turtlesim_node" name="turtle_GUI"
output="screen" />
    <node pkg="turtlesim" type="turtle_teleop_key" name="turtle_key"
output="screen" />
  </group>
  <group ns="second">
    <node pkg="turtlesim" type="turtlesim_node" name="turtle_GUI"
output="screen" />
    <node pkg="turtlesim" type="turtle_teleop_key" name="turtle_key"
output="screen" />
  </group>
</launch>

```

```

meroke@meroke-W650KJ1-KK1:~/code/course01$ rosparam list
/first/turtle_GUI/background_b
/first/turtle_GUI/background_g
/first/turtle_GUI/background_r
/roscdistro
/roslaunch/uris/host_localhost__45573
/rosversion
/run_id
/second/turtle_GUI/background_b
/second/turtle_GUI/background_g
/second/turtle_GUI/background_r

```

arg

```

<launch>
  <arg name="car_length" default="0.55"/>
  <param name="A" value="$(arg car_length)"/>
  <param name="B" value="$(arg car_length)" />
</launch>

```

TF坐标变换

1.geometry_msgs/TransformStamped

命令行键入: `rosmmsg info geometry_msgs/TransformStamped`

std_msgs/Header header	#头信息
uint32 seq	# -- 序列号
time stamp	# -- 时间戳
string frame_id	# -- 坐标 ID
string child_frame_id	#子坐标系的 id
geometry_msgs/Transform transform	#坐标信息
geometry_msgs/Vector3 translation	#偏移量
float64 x	# -- X 方向的偏移量
float64 y	# -- Y 方向的偏移量
float64 z	# -- Z 方向上的偏移量
geometry_msgs/Quaternion rotation	#四元数
float64 x	
float64 y	
float64 z	
float64 w	

四元数用于表示坐标的相对姿态

2.geometry_msgs/PointStamped

命令行键入: `rosmmsg info geometry_msgs/PointStamped`

```
std_msgs/Header header          #头
  uint32 seq                    #|-- 序号
  time stamp                    #|-- 时间戳
  string frame_id               #|-- 所属坐标系的 id
geometry_msgs/Point point       #点坐标
  float64 x                     #|-- x y z 坐标
  float64 y
  float64 z
```

静态坐标转换

```
// pub.cpp
// 3.创建静态坐标转换广播器
tf2_ros::StaticTransformBroadcaster broadcaster;
// 4.创建坐标系信息
geometry_msgs::TransformStamped ts;
  ts.header.frame_id = "base_link"; //被参考的坐标系，基本是小车主体
//----设置子级坐标系
  ts.child_frame_id = "laser"; //雷达坐标系
  broadcaster.sendTransform(ts);

//sub.cpp
geometry_msgs::PointStamped point_laser;
point_laser.header.frame_id = "laser";
geometry_msgs::PointStamped point_base;
//将point_laser 转换成 相对base_link 的坐标
point_base = buffer.transform(point_laser, "base_link");
```

订阅sub是耗时操作，使用launch时，会一开始由于没接受到数据报错 `base_link不存在`

解决方法：

1. 延时休眠
2. 异常处理

补充1:

当坐标系之间的相对位置固定时，那么所需参数也是固定的: 父系坐标名称、子级坐标系名称、x偏移量、y偏移量、z偏移量、x 翻滚角度、y俯仰角度、z偏航角度，实现逻辑相同，参数不同，那么 ROS 系统就已经封装好了专门的节点，使用方式如下：

```
roslaunch tf2_ros static_transform_publisher x偏移量 y偏移量 z偏移量 z偏航角度 y俯仰角度
x翻滚角度 父级坐标系 子级坐标系
```

示例: `roslaunch tf2_ros static_transform_publisher 0.2 0 0.5 0 0 0 /base_link /laser`

针对角度: $1.570795 = \pi/2 =$ 逆时针90度

也建议使用该种方式直接实现静态坐标系相对信息发布。

可尝试将该指令封装进shell启动脚本或py启动脚本

补充2:

可以借助于rviz显示坐标系关系，具体操作:

- 新建窗口输入命令:rviz;
- 在启动的 rviz 中设置Fixed Frame 为 base_link;
- 点击左下的 add 按钮，在弹出的窗口中选择 TF 组件，即可显示坐标关系。

动态坐标

```
geometry_msgs::PointStamped ps;  
ps.header.frame_id = "son1"; // 以son1为坐标系原点生成新点ps  
ps.header.stamp = ros::Time::now();  
ps.point.x = 1.0;  
ps.point.y = 2.0;  
ps.point.z = 3.0;
```

```
// son2为父， son1为子，生成一个子son1 相对 父亲son2 的点  
geometry_msgs::TransformStamped tfs =  
buffer.lookupTransform("son2", "son1", ros::Time(0));  
ROS_INFO("Son1 相对于 Son2 的坐标关系:父坐标系  
ID=%s", tfs.header.frame_id.c_str());  
ROS_INFO("Son1 相对于 Son2 的坐标关系:子坐标系  
ID=%s", tfs.child_frame_id.c_str());  
ROS_INFO("Son1 相对于 Son2 的坐标关系:x=%.2f,y=%.2f,z=%.2f",  
tfs.transform.translation.x,  
tfs.transform.translation.y,  
tfs.transform.translation.z  
);
```

```
# point_laser 原坐标是turtle， 现在转换成相对world的坐标点  
point_base = buffer.transform(point_laser, "world");
```

坐标查询

```
roslaunch tf2_tools view_frames.y #生成坐标关系pdf  
evince frame.pdf # 查看pdf
```


系统仿真

URDF 文件是一个标准的 XML 文件，在 ROS 中预定义了一系列的标签用于描述机器人模型，机器人模型可能较为复杂，但是 ROS 的 URDF 中机器人的组成却是较为简单，可以主要简化为两部分:连杆(link 标签) 与 关节(joint 标签)，接下来我们就通过案例了解一下 URDF 中的不同标签：

- robot 根标签，类似于 launch 文件中的 launch 标签
- link 连杆标签
- joint 关节标签
- gazebo 集成 gazebo 需要使用的标签

关于 gazebo 标签，后期在使用 gazebo 仿真时，才需要使用到，用于配置仿真环境所需参数，比如：机器人材料属性、gazebo 插件等，但是该标签不是机器人模型必须的，只有在仿真时才需设置

- visual ---> 描述外观(对应的数据是可视的)
 - geometry 设置连杆的形状
 - 标签1: box(盒状)
 - 属性: size=长(x) 宽(y) 高(z)
 - 标签2: cylinder(圆柱)
 - 属性: radius=半径 length=高度
 - 标签3: sphere(球体)
 - 属性: radius=半径
 - 标签4: mesh(为连杆添加皮肤)
 - 属性: filename=资源路径(格式: **package:////文件**)
 - origin 设置偏移量与倾斜弧度
 - 属性1: xyz=x 偏移 y 偏航 z 偏移
 - 属性2: rpy=x 翻滚 y 俯仰 z 偏航 (单位是弧度)
 - material 设置材料属性(颜色)
 - 属性: name
 - 标签: color
 - 属性: rgba=红绿蓝权重值与透明度 (每个权重值以及透明度取值[0,1])
- collision ---> 连杆的碰撞属性
- Inertial ---> 连杆的惯性矩阵

```
<robot name="mycar">
  <link name="base_link">
    <visual>
      <geometry>
        <box size="0.3 0.2 0.1" />
        <!-- <cylinder radius="0.5" length="2"/>
        <sphere radius="1" />
        <mesh filename="package://vr/meshes//.stl" -->
      </geometry>
      <!-- xyz 平移量   rpy 对应轴上旋转角度 1.57=90度，左右旋转取决于z -->
      <origin xyz="3 0 0" rpy="0 0 1.57"/>
    </visual>
  </link>
</robot>
```

xacro

注意！！：xacro文件内不可以存在中文注释，否则会编码报错

属性

```
<xacro:property name="xxx" value="yyy" />
```

宏

类似于函数实现，提高代码复用率，优化代码结构，提高安全性

宏定义

```
<xacro:macro name="宏名称" params="参数列表(多参数之间使用空格分隔)">

    .....

    参数调用格式：${参数名}

</xacro:macro>
```

宏调用

```
<xacro:宏名称 参数1=xxx 参数2=xxx/>
```

示例：

```
<robot name="mycar" xmlns:xacro="http://wiki.ros.org/xacro">
  <!-- 宏定义 -->
  <xacro:macro name="getSum" params="num1 num2">
    <result value="${num1 + num2}"/>
  </xacro:macro>
  <!-- 宏调用 -->
  <xacro:getSum num1="1" num2="2"/>
</robot>
```

```
meroke@meroke-W650KJ1-KK1:~/code/course03/src/vr/urdf/xacro$ rosrunc xacro xacro macro.urdf.xacro
<?xml version="1.0" encoding="utf-8"?>
<!-- ===== -->
<!-- | This document was autogenerated by xacro from macro.urdf.xacro | -->
<!-- | EDITING THIS FILE BY HAND IS NOT RECOMMENDED | -->
<!-- ===== -->
<robot name="mycar">
  <result value="3"/>
</robot>
```

文件包含

机器人由多部件组成，不同部件可能封装为单独的 xacro 文件，最后再将不同的文件集成，组合为完整机器人，可以使用文件包含实现

文件包含

```
<robot name="xxx" xmlns:xacro="http://wiki.ros.org/xacro">
  <xacro:include filename="my_base.xacro" />
  <xacro:include filename="my_camera.xacro" />
  <xacro:include filename="my_laser.xacro" />
  ....
</robot>
```

launch启动

textfile

```
<param name="robot_description" textfile="$(find vr)/urdf/xacro/all.urdf" />
```

```
<launch>
  <!-- 参数服务器载入urdf -->
  <param name="robot_description" textfile="$(find vr)/urdf/xacro/all.urdf" />
  <node pkg="rviz" type="rviz" name="rviz" args="-d $(find
vr)/config/ros_car.rviz"/>
  <node pkg="joint_state_publisher" type="joint_state_publisher"
name="joint_state_publisher" />
  <node pkg="robot_state_publisher" type="robot_state_publisher"
name="robot_state_publisher" />

</launch>
```

command

```
<param name="robot_description" command="$(find xacro)/xacro $(find
vr)/urdf/xacro/all.urdf.xacro" />
```

```
<launch>
  <!-- 参数服务器载入urdf -->
  <param name="robot_description" command="$(find xacro)/xacro $(find
vr)/urdf/xacro/all.urdf.xacro" />
  <node pkg="rviz" type="rviz" name="rviz" args="-d $(find
vr)/config/ros_car.rviz"/>
  <node pkg="joint_state_publisher" type="joint_state_publisher"
name="joint_state_publisher" />
  <node pkg="robot_state_publisher" type="robot_state_publisher"
name="robot_state_publisher" />

</launch>
```

arbotix运动仿真

yaml

```
# 该文件是控制器配置,一个机器人模型可能有多个控制器,比如: 底盘、机械臂、夹持器(机械手)....
# 因此, 根 name 是 controller
controllers: {
  # 单控制器设置
  base_controller: {
    #类型: 差速控制器
    type: diff_controller,
```

```

    #参考坐标
    base_frame_id: base_footprint,
    #两个轮子之间的间距
    base_width: 0.2,
    #控制频率
    ticks_meter: 2000,
    #PID控制参数, 使机器人车轮快速达到预期速度
    Kp: 12,
    Kd: 12,
    Ki: 0,
    Ko: 50,
    #加速限制
    accel_limit: 1.0
  }
}

```

launch启动文件

```

<node name="arbotix" pkg="arbotix_python" type="arbotix_driver" output="screen">
  <rosparam file="$(find my_urdf05_rviz)/config/hello.yaml" command="load" />
  <param name="sim" value="true" />
</node>

```

雷达仿真

新建在gazebo文件夹下的laser.xacro, 注意将仿真和模型配合起来时, 仿真的reference是针对模型的link名的

```

<robot name="my_sensors" xmlns:xacro="http://wiki.ros.org/xacro">

  <!-- 雷达 -->
  <!-- 这里的reference 是对应前面编写的雷达的link名-->
  <gazebo reference="laser">
    <sensor type="ray" name="rplidar">
      <pose>0 0 0 0 0 0</pose>
      <visualize>true</visualize>
      <update_rate>5.5</update_rate> <!-- 每秒5.5次-->
      <ray>
        <scan>
          <horizontal>
            <samples>360</samples> <!-- 旋转一周发射360条射线, 或说是采样360个点-->
            <resolution>1</resolution> <!-- 分辨率, 每N条射线测距一条-->
            <min_angle>-3</min_angle> <!-- 一周rad=6.28 , 这里是角度范围-3到3-->
            <max_angle>3</max_angle>
          </horizontal>
        </scan>
        <range>
          <min>0.10</min> <!-- 障碍物 必须距离在0.1- 30 m 之间才能被采样-->
          <max>30.0</max>
          <resolution>0.01</resolution> <!-- 0.01m精度-->
        </range>
        <noise>

```

```

        <type>gaussian</type>  <!-- 高斯噪音 模拟测距抖动 -->
        <mean>0.0</mean>
        <stddev>0.01</stddev>
    </noise>
</ray>
<plugin name="gazebo_rplidar" filename="libgazebo_ros_laser.so">
    <topicName>/scan</topicName>
    <frameName>laser</frameName>
</plugin>
</sensor>
</gazebo>

</robot>

```

摄像头仿真

```

<robot name="my_sensors" xmlns:xacro="http://wiki.ros.org/xacro">
  <gazebo reference="camera">
    <sensor type="camera" name="camera_node">
      <update_rate>30.0</update_rate>
      <camera name="head">
        <horizontal_fov>1.3962634</horizontal_fov>
        <image>
          <width>1280</width>
          <height>720</height>
          <format>R8G8B8</format>
        </image>
        <clip>
          <near>0.02</near>
          <far>300</far>
        </clip>
        <noise>
          <type>gaussian</type>
          <mean>0.0</mean>
          <stddev>0.007</stddev>
        </noise>
      </camera>
      <plugin name="gazebo_camera" filename="libgazebo_ros_camera.so">
        <alwaysOn>true</alwaysOn>
        <updateRate>0.0</updateRate>
        <cameraName>/camera</cameraName> <!-- image topic -->
        <imageTopicName>image_raw</imageTopicName><!-- image topic -->
        <cameraInfoTopicName>camera_info</cameraInfoTopicName>
        <frameName>camera</frameName>
        <hackBaseline>0.07</hackBaseline>
        <distortionK1>0.0</distortionK1>
        <distortionK2>0.0</distortionK2>
        <distortionK3>0.0</distortionK3>
        <distortionT1>0.0</distortionT1>
        <distortionT2>0.0</distortionT2>
      </plugin>
    </sensor>
  </gazebo>
</robot>

```

深度摄像头仿真

```
<robot name="my_sensors" xmlns:xacro="http://wiki.ros.org/xacro">
  <gazebo reference="support"> <!-- 这里将摄像头支架设置成了深度摄像头 -->
    <sensor type="depth" name="camera">
      <always_on>true</always_on>
      <update_rate>20.0</update_rate>
      <camera>
        <horizontal_fov>${60.0*PI/180.0}</horizontal_fov>
        <image>
          <format>R8G8B8</format>
          <width>640</width>
          <height>480</height>
        </image>
        <clip>
          <near>0.05</near>
          <far>8.0</far>
        </clip>
      </camera>
      <plugin name="kinect_camera_controller"
filename="libgazebo_ros_openni_kinect.so">
        <cameraName>camera</cameraName>
        <alwaysOn>true</alwaysOn>
        <updateRate>10</updateRate>
        <imageTopicName>rgb/image_raw</imageTopicName>
        <depthImageTopicName>depth/image_raw</depthImageTopicName>
        <pointCloudTopicName>depth/points</pointCloudTopicName>
        <cameraInfoTopicName>rgb/camera_info</cameraInfoTopicName>

        <depthImageCameraInfoTopicName>depth/camera_info</depthImageCameraInfoTopicName>
      </plugin>
      <!-- 这里还对应坐标系，由于图像和点云坐标系不同，需要手动发布坐标变换-->
      <frameName>support</frameName><!-- 这里将摄像头支架设置成了深度摄像头 -->
      <baseline>0.1</baseline>
      <distortion_k1>0.0</distortion_k1>
      <distortion_k2>0.0</distortion_k2>
      <distortion_k3>0.0</distortion_k3>
      <distortion_t1>0.0</distortion_t1>
      <distortion_t2>0.0</distortion_t2>
      <pointCloudCutoff>0.4</pointCloudCutoff>
    </sensor>
  </gazebo>
</robot>
```

导航

保存地图

将正在运行的rviz地图保存

```
<launch>
  <arg name="filename" value="$(find mycar_nav)/map/nav" />
  <node name="map_save" pkg="map_server" type="map_saver" args="-f $(arg
filename)" />
</launch>
```

读取地图

```
<launch>
  <!-- 设置地图的配置文件 -->
  <arg name="map" default="nav.yaml" />
  <!-- 运行地图服务器，并且加载设置的地图-->
  <node name="map_server" pkg="map_server" type="map_server" args="$(find
mycar_nav)/map/$(arg map)"/>
</launch>
```

注意: yaml中不可以有中文字符

```
# nav.yaml
image: /home/meroke/code/course03/src/nav_ma/map/nav.pgm
# 分辨率 m/像素
resolution: 0.0500000
# 相对x y 偏航角度
origin: [-50.000000, -50.000000, 0.000000]
# 黑白取反
negate: 0
# 坐标向素值 > occupied_thresh, 视作障碍物
#           < free_thresh,    视作无物
occupied_thresh: 0.65
free_thresh: 0.196
```

经验教训

QS1:rviz地图与gazebo不同步

详细描述:

文件位置 `course03`

启动urdf_gazebo/launch/union.launch (模型启动文件 和 gazebo环境) , 然后再启动 nav_ma/launch/nav.launch (rviz建图) , 出现rviz中 no map receive , 地图无显示, 且无报错。

正常应该是rviz同步显示gazebo地图中被雷达检测到的部分。

问题解决:

发现是更改urdf_gazebo/xacro/my_base.urdf.xacro中的小车底盘时，添加了车轮，从二轮转为四轮，并修改了之前的轮子名称。但没有修改urdf_gazebo/xacro/gazebo/move.xacro小车控制器对应的joint，导致运行出错，节点未正常发布，修改后即可。

QS2:仿真运动需要保证rviz与gazebo地图相同

详细描述：

文件位置 `course03`

仿真运动是指用gazebo环境模拟真实环境，然后通过运动控制模拟导航效果。

因此需要保证rviz的地图与gazebo相同。

一般 gazebo地图 --> rviz 地图

流程：

1. 启动urdf_gazebo/launch/union.launch（模型启动文件 和 gazebo环境）
2. 然后再启动nav_ma/launch/nav.launch（rviz建图），rviz地图与同步gazebo。但此时的rviz只有雷达检测到的部分地图，无法从一开始获取全部地图信息。
3. 因此需要手动移动小车，运行 `roslaunch teleop_twist_keyboard teleop_twist_keyboard.py`，控制小车前进，通过雷达扫描各处更新地图，最后获取完整地图。
4. 启动 nav_ma/launch/map_save.launch，即可直接保存完整地图为nav.yaml

QS3:运动时出现Extrapolation Error

问题描述：使用ted_local_planner_params.yaml后，进行仿真运动时出现如下报错。

```
[ WARN] [1644983417.231789388, 219.046000000]: Could not transform the global plan to the
frame of the controller
[ERROR] [1644983417.555488794, 219.196000000]: Extrapolation Error: Lookup would require
extrapolation into the future. Requested time 219.189000000 but the latest data is at ti
me 219.152000000, when looking up transform from frame [odom] to frame [map]

[ERROR] [1644983417.555615862, 219.196000000]: Global Frame: odom Plan Frame size 43: map

[ WARN] [1644983417.555661931, 219.196000000]: Could not transform the global plan to the
frame of the controller
[ERROR] [1644983417.903786566, 219.346000000]: Extrapolation Error: Lookup would require
extrapolation into the future. Requested time 219.339000000 but the latest data is at ti
me 219.334000000, when looking up transform from frame [odom] to frame [map]

[ERROR] [1644983417.904139899, 219.346000000]: Global Frame: odom Plan Frame size 42: map

[ WARN] [1644983417.904272883, 219.346000000]: Could not transform the global plan to the
frame of the controller
^C[rviz-6] killing on exit
```

解决方法：

此处是local_costmap_params.yaml中 global_frame 需要设置为 map，而不是 odom

引用：<https://answers.ros.org/question/304537/could-not-transform-the-global-plan-to-the-frame-of-the-controller/>