

## 6.30

---

任务是熟悉 **fuel ros cmake** 搜索多机协同解决文章（便于未来解决）

今日任务：看下fuel框架 熟悉背景等等

明日任务：开始看ros，大概一周可以看完，看完找华去要简单实验

## 7.1

---

学习命令行

```
cd ..
//返回
cd ./xxw

touch test_file
//touch命令用于创建新文件或更新现有文件的时间戳。
mv test_file /home/xxw
//移动文件位置(剪切)
cp test_file /home/xxw/test_file2
//拷贝目标目录下，并且可以重新命名
rm -r
//删除
sudo
//提升权限

g++ test.cpp -o testname
//编译成可运行文件，后面接名字
./testname
//运行文件
```

## 7.2

---

完成ros安装，因为之前是版本装错了，早点来，安装好，后面接着看，因为安装要很久

好的可以使用

```
sudo apt install aptitude
```

```
sudo aptitude install python-roslib
```

来解决以来冲突问题（`python-roslib`），以及对应的版本安装对应的ros

`roscore`是启动ros

## 显示节点列表：`rosnodes list`

其中`rosout`是启动ros就默认的有的

## 查看节点信息：`rosnodes info`

`publication` 发布的话题

`subscription`订阅的话题

`services` 服务

## 显示topic列表：`rostopic list`

### 手动发布数据给topic（其中定义数据名和数据格式）

`publisher`给数据给topic，`subscribe`订阅其中的数据

我们来手动发布数据给topic来控制海龟移动，使用 `rostopic pub`

### `rosmsg`查看消息数据结构

## 使用服务通信方式：`rosservice`

通信是双向的，是有交流的，不像订阅是单向的

### 查看服务列表：`rosservice list`

### 手动通过服务来进行操作

格式：`rosservice call (参数) 服务名 “具体数据”`

## 创建工作空间和编译包功能

### What is Workspace(工作空间)

工作空间（Workspace）：存放工程开发相关文件的文件夹。类似一个IDE（例如Pycharm）新建一个工程，就是一个工作空间。包含4个文件夹：

src: 代码空间 (Source Space): 放置功能包代码  
build: 编译空间 (Build Space): 编译过程中产生的中间文件, 不用过多关注  
devel: 开发空间 (Development Space): 放置编译生成的可执行文件、库、脚本  
install: 安装空间 (Install Space): 存放可执行文件, 与上面有一定重复

## • 工作空间



工作空间 (workspace) 是一个存放工程开发相关文件的文件夹。

- **src**: 代码空间 (Source Space)
- **build**: 编译空间 (Build Space)
- **devel**: 开发空间 (Development Space)
- **install**: 安装空间 (Install Space)

```
workspace_folder/          -- WORKSPACE
src/                      -- SOURCE SPACE
CMakeLists.txt             -- The 'toplevel' CMake file
package_1/
  CMakeLists.txt
  package.xml
...
package_n/
  CMakeLists.txt
  package.xml
...
build/                     -- BUILD SPACE
  CATKIN_IGNORE
  devel/
    bin/
    etc/
    include/
    lib/
    share/
    .catkin
    env.bash
    setup.bash
    setup.sh
...
install/                  -- INSTALL SPACE (set by CMAKE_INSTALL_PREFIX)
  bin/
  etc/
  include/
  lib/
  share/
  .catkin
  env.bash
  setup.bash
  setup.sh
...
```

catkin编译系统下的工作空间结构

## 创建工作空间

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/src
catkin_init_workspace意思是将当前文件夹变为工作空间
```

一般是在src下面创建, 名字固定的

## 编译空代码的工作空间

要编译工作空间, 先要回到工作空间的根目录。

```
cd ~/catkin_ws
catkin_make //编译命令
要生成install文件的话要
catkin_make install
```

devel开发空间

build编译空间

install生成可执行文件

## 创建功能包

功能包是放置ROS源码的最小单元。

上面我们创建了一个空的工作空间，src文件夹里面没写东西，现在我们创建一个自己的功能包。

注意同一工作空间下，不允许存在同名功能包；不同工作空间下，允许存在同名功能包。

## 创建功能包

指令格式：`catkin_create_pkg <package_name> [depend1] [depend2] [depend3]`

<package\_name>为包名

[depend]为依赖，即指明编译的时候需要ROS中的其他功能包，如需要调用python、C++库，就要指明rospy、roscpp。

我们创建功能包：

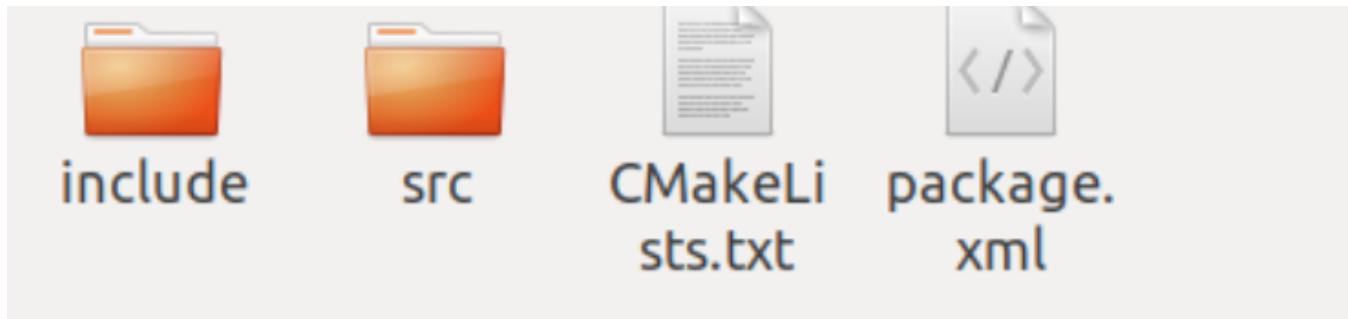
```
cd ~/catkin_ws/src  
catkin_create_pkg test_pkg std_msgs rospy roscpp
```

其中里面

**include**: 放置cpp头文件等等比如std之类的

**src**: 放置代码

**cmakelist**: 必有文件等等



## 编译新的功能包

回到工作空间根目录，再编译一下。

```
cd ~/catkin_ws  
catkin_make
```

## 设置和检查环境变量

### 设置

编译完功能包后，为了运行，先[设置环境变量](#)，以便系统找到我们的工作空间和功能包。

```
source ~/catkin_ws/devel/setup.bash
```

## 检查

```
echo $ROS_PACKAGE_PATH  
AI写代码powershell1
```

echo为打开环境变量，通过 ROS\_PACKAGE\_PATH 查找所有ros功能包的路径。

前面source了 devel/setup.bash ， 现在就会包含这个路径：

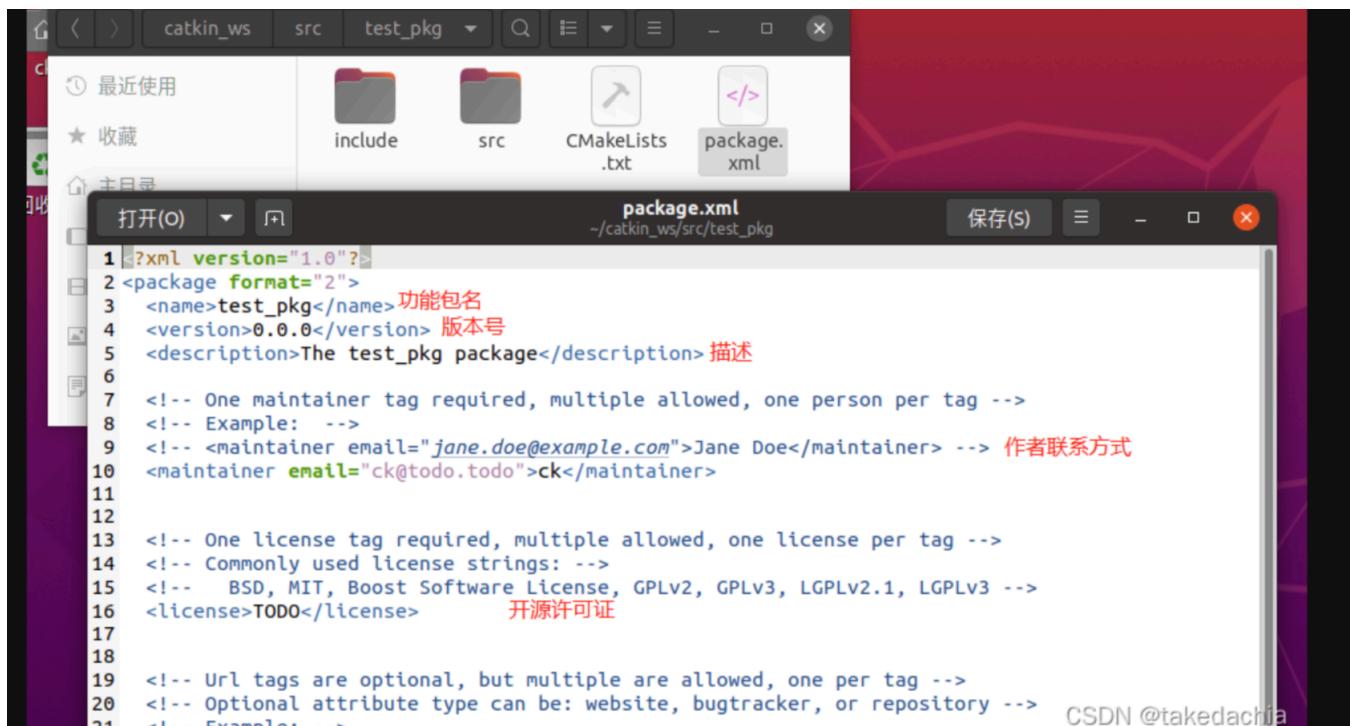
```
xxxx@xxw-vpc:~/catkin_ws$ source ~/catkin_ws/devel/setup.bash  
xxxx@xxw-vpc:~/catkin_ws$ echo $ROS_PACKAGE_PATH  
/home/xxw/catkin_ws/src:/opt/ros/melodic/share  
xxxx@xxw-vpc:~/catkin_ws$
```

## 功能包中的两个重要文件

### package.xml

使用xml语言描述功能包相关的信息：

(后面的课会用到)



The screenshot shows a code editor window displaying the `package.xml` file for a ROS package named `test_pkg`. The file content is as follows:

```
1 <?xml version="1.0"?>  
2 <package format="2">  
3   <name>test_pkg</name>功能包名  
4   <version>0.0.0</version>版本号  
5   <description>The test_pkg package</description>描述  
6  
7   <!-- One maintainer tag required, multiple allowed, one person per tag -->  
8   <!-- Example: -->  
9   <!-- <maintainer email="jane.doe@example.com">Jane Doe</maintainer> --> 作者联系方式  
10  <maintainer email="ck@todo.todo">ck</maintainer>  
11  
12  
13  <!-- One license tag required, multiple allowed, one license per tag -->  
14  <!-- Commonly used license strings: -->  
15  <!-- BSD, MIT, Boost Software License, GPLv2, GPLv3, LGPLv2.1, LGPLv3 -->  
16  <license>TODO</license>开源许可证  
17  
18  
19  <!-- Url tags are optional, but multiple are allowed, one per tag -->  
20  <!-- Optional attribute type can be: website, bugtracker, or repository -->  
21  <!-- Example: -->
```

CSDN @takedachia

```
50  <!-- <doc_depend>doxygen</doc_depend> -->  
51  <buildtool_depend>catkin</buildtool_depend>  
52  <build_depend>roscpp</build_depend>  
53  <build_depend> rospy </build_depend>依赖信息  
54  <build_depend> std_msgs</build_depend>  
55  <build_export_depend> roscpp</build_export_depend>  
56  <build_export_depend> rospy</build_export_depend>  
57  <build_export_depend> std_msgs</build_export_depend>  
58  <exec_depend> roscpp</exec_depend>  
59  <exec_depend> rospy</exec_depend>  
60  <exec_depend> std_msgs</exec_depend>
```

## CMakeLists.txt

描述功能包里的编译规则，使用CMake语法。

(后面的课会越来越多的用到)



The screenshot shows a terminal window with the following content:

```
catkin_ws | src | test_pkg | CMakeLists.txt | package.xml
CMakeLists.txt
1 cmake_minimum_required(VERSION 3.0.2)
2 project(test_pkg)
3
4 ## Compile as C++11, supported in ROS Kinetic and newer
5 # add_compile_options(-std=c++11)
6
7 ## Find catkin macros and libraries
8 ## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
9 ## is used, also find other catkin packages
10 find_package(catkin REQUIRED COMPONENTS
11   roscpp
12   rospy
13   std_msgs 比如设定依赖库
14 )
15
16 ## System dependencies are found with CMake's conventions
17 # find_package(Boost REQUIRED COMPONENTS system)
18
```

Annotations in red text are overlaid on the code:

- “描述功能包里的编译规则” is placed next to the first few lines of the file.
- “比如设定依赖库” is placed next to the line “rospy”.

## publisher编程实现

### 1、模型图

图中，我们使用ROS Master管理节点。

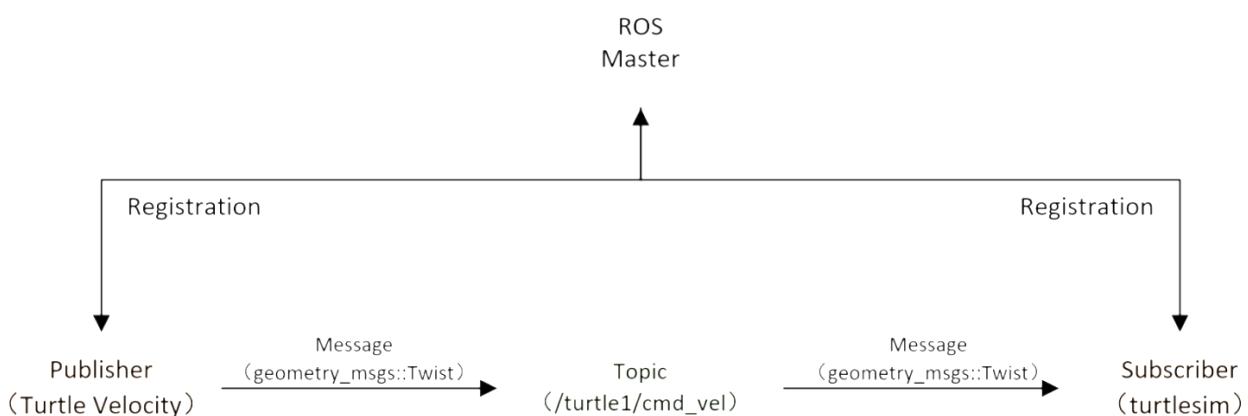
有两个主要节点：

Publisher，名为Turtle Velocity（即海龟的速度）

Subscriber，即海龟仿真器节点/turtlesim

Publisher(Turtle Velocity)，发布Message（即海龟的速度信息，以geometry\_msgs::Twist的数据结构，包括线速度和角速度），通过Topic（/turtle1/cmd\_vel）总线管道，将数据传输给Subscriber。Subscriber订阅得到的速度信息，来控制海龟发生运动。

“/turtle1/cmd\_vel”这个topic是海龟仿真器节点/turtlesim下自带的topic，可直接拿来用。



## 2、创建功能包

我们之前已经创建了工作空间了，这次我们在src文件夹创建一个新的功能包learning topic

```
cd ~/catkin_ws/src  
catkin_create_pkg learning_topic roscpp rospy std_msgs geometry_msgs turtlesim
```

## 3、创建Publisher代码（以C++为例）

关于如何实现一个Publisher：

初始化ROS节点

向ROS Master注册节点信息，包括发布的话题名和话题中的消息类型

创建消息数据

按照一定的频率循环发布消息

先以C++为例（后面会讲Python的情形），我们这次需要在src里创建C++的代码文件以输入代码。源码见下。

（源码地址：[https://github.com/guyuehome/ros\\_21\\_tutorials/blob/master/learning\\_topic/src/velocity\\_publisher.cpp](https://github.com/guyuehome/ros_21_tutorials/blob/master/learning_topic/src/velocity_publisher.cpp)）

创建cpp代码

```
touch test.cpp
```

写代码

## 4 编译代码（以C++为例）

有了代码，接下来编译。

配置Publisher代码编译规则

首先需要配置CMakeLists.txt中的编译规则：

设置需要编译的代码和生成的可执行文件

设置链接库

将下列代码拷贝至CMakeLists.txt中：（当前的功能包下面里面）

```
add_executable(velocity_publisher src/velocity_publisher.cpp)  
//指明将一个程序文件编译成什么可执行文件  
target_link_libraries(velocity_publisher ${catkin_LIBRARIES})  
//将可执行文件与ROS相关库做关联
```

```
149  
150 ## Specify libraries to link a library or executable target against  
151 # target_link_libraries(${PROJECT_NAME}_node  
152 #   ${catkin_LIBRARIES}  
153 # )  
154  
155 add_executable(velocity_publisher src/velocity_publisher.cpp) // 指明将哪一个程序文件编译成什么可执行文件  
156 target_link_libraries(velocity_publisher ${catkin_LIBRARIES}) // 将可执行文件和ROS相关的库做链接  
157  
158 #####  
159 ## Install ##  
160 #####
```

在第155行找到了括号匹配 CMake 制表符宽度：8 第155行，第62列 插入

## 执行编译

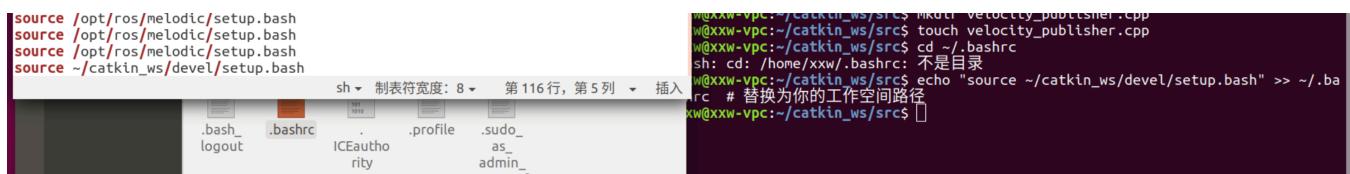
回到工作空间目录，执行编译。

```
cd ~/catkin_ws  
catkin_make
```

## source 一下 setup.bash

我们之后每次运行这个程序都需要source一下devel/setup.bash，我们不妨将 `source devel/setup.bash` 放入环境变量.bashrc中。（原因是这个每次开终端会自动执行，这样就不用每次自动加载工作环境）

.bashr一般是隐藏在的



```
source /opt/ros/melodic/setup.bash  
source /opt/ros/melodic/setup.bash  
source /opt/ros/melodic/setup.bash  
source ~/catkin_ws/devel/setup.bash  
  
w@xxw-vpc:~/catkin_ws/src$ touch velocity_publisher.cpp  
w@xxw-vpc:~/catkin_ws/src$ cd ~/.bashrc  
sh: cd: /home/xxw/.bashrc: 不是目录  
w@xxw-vpc:~/catkin_ws/src$ echo "source ~/catkin_ws/devel/setup.bash" >> ~/.ba  
sh: echo: 命令未找到  
w@xxw-vpc:~/catkin_ws/src$
```

立即生效：

```
source ~/.bashrc # 刷新当前配置
```

接着执行常规操作

## 运行

打开终端，分别运行：

```
roscore  
  
rosrun turtlesim turtlesim_node  
  
rosrun learning_topic velocity_publisher
```

## 7.3

一般流程是编写代码，在cmake中配置编译规则，回到工作空间，执行编译（catkin\_make），source一下 setup.bash，最后运行

其中cmake是为了在编译过程中能找到需要编译的文件，为catkin\_make准备，source是为了让ros命令能识别工作空间的功能包。

还有就是在这版系统中rosrun xx\_sdd

不能加.py后缀不然无法运行

# 话题消息的定义和使用

## 一、自定义话题消息

- 1、**定义msg文件**
- 2、**在package.xml中添加功能包依赖**
- 3、**在CmakeList.txt中添加编译选项**
- 4、**编译生成C++头文件或者Python库**

1、

定义的msg文件与语言无关

2、

打开package.xml文件，将下面代码拷到文件指定位置：

```
<build_depend>message_generation</build_depend>
<exec_depend>message_runtime</exec_depend>
```

build\_depend为编译依赖，这里依赖的是一个会动态产生message的功能包  
exer\_depend为执行依赖，这里依赖的是一个动态runtime运行的功能包

3、

在CMakeLists.txt中添加编译选项

因为在package.xml添加了功能包编译依赖，在CMakeList.txt里的find\_package中也要加上对应的部分；

需要将定义的Person.msg作为消息接口，针对它做编译；

需要指明编译这个消息接口需要哪些ROS已有的包；（有了这两个配置才可将定义的msg编译成不同的程序文件）

因为在package.xml添加了功能包执行依赖，在CMakeList.txt里的catkin\_package中也要加上对应的部分；

```
find_package( .... message_generation)加上编译依赖

add_message_files(FILES Person.msg)//将定义的Person.msg作为消息接口，这对做编译

generate_messages(DEPENDENCIES std_msgs)//指明编译这个消息接口需要哪些ROS包
//注意上面是generate，就是要看cmake里具体怎么说的
catkin_package( .... message_runtime)加上执行依赖
```

4、

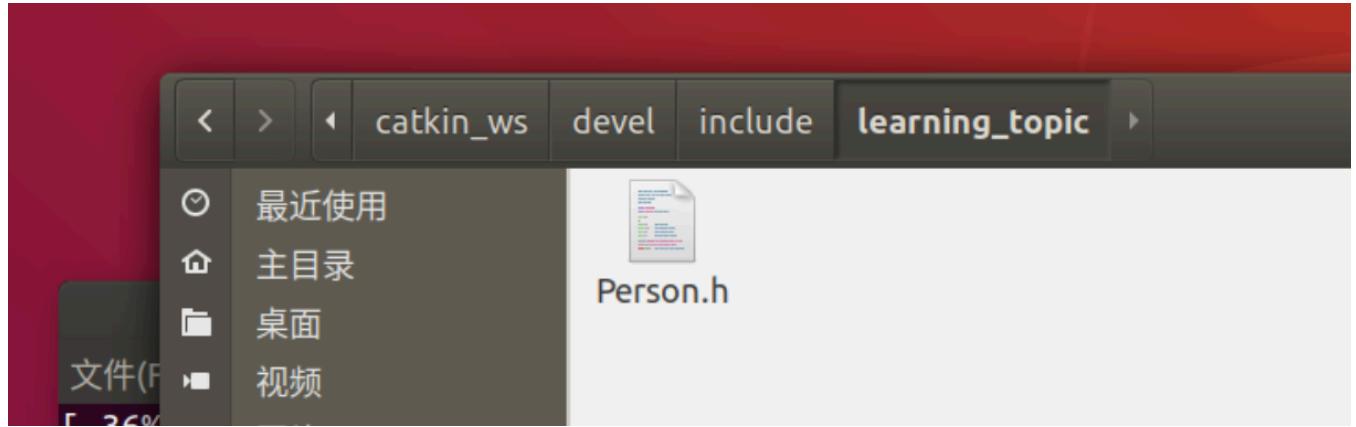
编译生成C++头文件或Python库

以上完成后，到工作空间根目录，编译：catkin\_make

编译完成后，我们可以在 devel/include/learning\_topic/ 下找到这个C++的头文件

也可以在 devel/lib/python3/dist-packages/learning\_topic/msg 下找到Python的包

接下来我们就可以通过编写程序来调用生成的.h或.py了！



## 二、创建代码并编译运行（C++）

1、先创建代码

2、编辑cmake

先配置CMakeLists.txt编译规则，复习一下规则：

- 设置需要编译的代码和生成的可执行文件；
- 设置链接库；
- **添加依赖项。**

```
add_executable(person_publisher src/person_publisher.cpp)
target_link_libraries(person_publisher ${catkin_LIBRARIES})
add_dependencies(person_publisher ${PROJECT_NAME}_generate_messages_cpp)

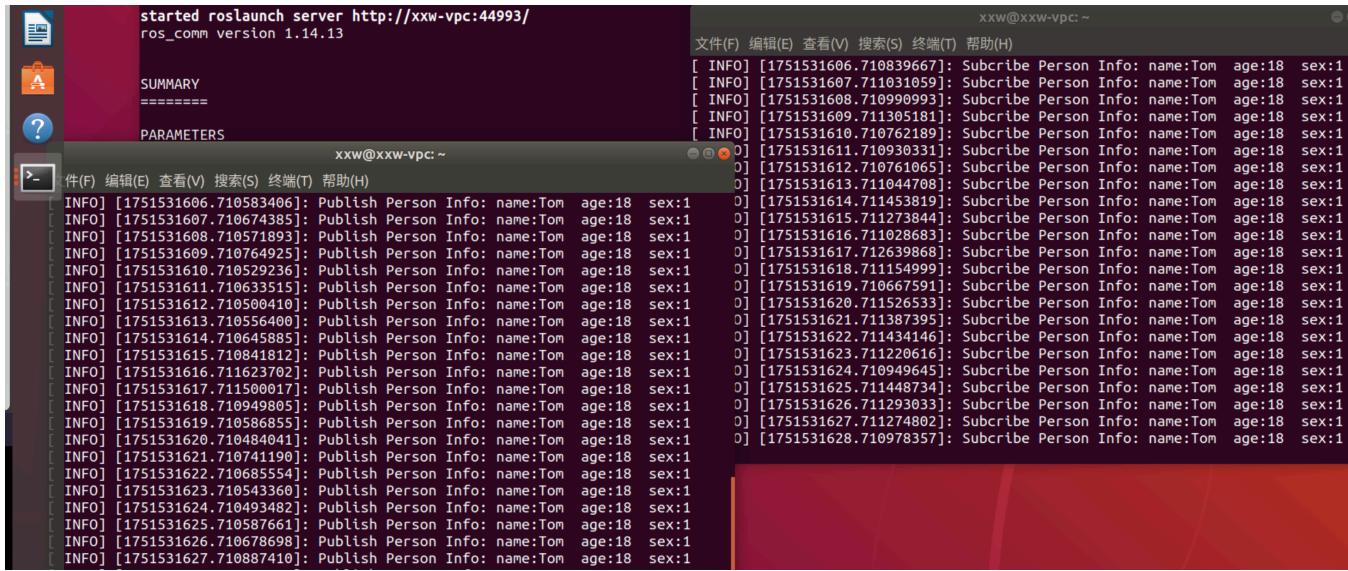
add_executable(person_subscriber src/person_subscriber.cpp)
target_link_libraries(person_subscriber ${catkin_LIBRARIES})
add_dependencies(person_subscriber ${PROJECT_NAME}_generate_messages_cpp)
```

这里新增了一个添加依赖项，因为代码涉及到动态生成，我们需要将可执行文件与动态生成的程序产生依赖关系。

（配置CMakeLists.txt编译规则，注意和C++的区别：

我们只要加上一个关于person\_publisher.py和person\_subscriber.py的catkin\_install\_python方法：只需要在catkin\_install\_python下加上scripts/person\_publisher.py就行）

接着运行就行



```
started roslaunch server http://xxw-vpc:44993/
ros_comm version 1.14.13

SUMMARY
========
PARAMETERS
        xxw@xxw-vpc: ~

件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
[ INFO] [1751531606.710583406]: Publish Person Info: name:Tom age:18 sex:1
[ INFO] [1751531607.710674385]: Publish Person Info: name:Tom age:18 sex:1
[ INFO] [1751531608.710571893]: Publish Person Info: name:Tom age:18 sex:1
[ INFO] [1751531609.710764925]: Publish Person Info: name:Tom age:18 sex:1
[ INFO] [1751531610.710529236]: Publish Person Info: name:Tom age:18 sex:1
[ INFO] [1751531611.710633515]: Publish Person Info: name:Tom age:18 sex:1
[ INFO] [1751531612.710500410]: Publish Person Info: name:Tom age:18 sex:1
[ INFO] [1751531613.710556400]: Publish Person Info: name:Tom age:18 sex:1
[ INFO] [1751531614.710645885]: Publish Person Info: name:Tom age:18 sex:1
[ INFO] [1751531615.710841812]: Publish Person Info: name:Tom age:18 sex:1
[ INFO] [1751531616.711623702]: Publish Person Info: name:Tom age:18 sex:1
[ INFO] [1751531617.711500017]: Publish Person Info: name:Tom age:18 sex:1
[ INFO] [1751531618.710949805]: Publish Person Info: name:Tom age:18 sex:1
[ INFO] [1751531619.710586855]: Publish Person Info: name:Tom age:18 sex:1
[ INFO] [1751531620.710484041]: Publish Person Info: name:Tom age:18 sex:1
[ INFO] [1751531621.710741190]: Publish Person Info: name:Tom age:18 sex:1
[ INFO] [1751531622.710685554]: Publish Person Info: name:Tom age:18 sex:1
[ INFO] [1751531623.710543360]: Publish Person Info: name:Tom age:18 sex:1
[ INFO] [1751531624.710493482]: Publish Person Info: name:Tom age:18 sex:1
[ INFO] [1751531625.710587661]: Publish Person Info: name:Tom age:18 sex:1
[ INFO] [1751531626.710678698]: Publish Person Info: name:Tom age:18 sex:1
[ INFO] [1751531627.710887410]: Publish Person Info: name:Tom age:18 sex:1

[ INFO] [1751531606.710839667]: Subscribe Person Info: name:Tom age:18 sex:1
[ INFO] [1751531607.711031059]: Subscribe Person Info: name:Tom age:18 sex:1
[ INFO] [1751531608.710990993]: Subscribe Person Info: name:Tom age:18 sex:1
[ INFO] [1751531609.711305181]: Subscribe Person Info: name:Tom age:18 sex:1
[ INFO] [1751531610.710762189]: Subscribe Person Info: name:Tom age:18 sex:1
[ INFO] [1751531611.710930331]: Subscribe Person Info: name:Tom age:18 sex:1
[ INFO] [1751531612.710761065]: Subscribe Person Info: name:Tom age:18 sex:1
[ INFO] [1751531613.711044708]: Subscribe Person Info: name:Tom age:18 sex:1
[ INFO] [1751531614.711453819]: Subscribe Person Info: name:Tom age:18 sex:1
[ INFO] [1751531615.711273841]: Subscribe Person Info: name:Tom age:18 sex:1
[ INFO] [1751531616.711028683]: Subscribe Person Info: name:Tom age:18 sex:1
[ INFO] [1751531617.712639868]: Subscribe Person Info: name:Tom age:18 sex:1
[ INFO] [1751531618.711154999]: Subscribe Person Info: name:Tom age:18 sex:1
[ INFO] [1751531619.710667591]: Subscribe Person Info: name:Tom age:18 sex:1
[ INFO] [1751531620.711526533]: Subscribe Person Info: name:Tom age:18 sex:1
[ INFO] [1751531621.711387395]: Subscribe Person Info: name:Tom age:18 sex:1
[ INFO] [1751531622.711434146]: Subscribe Person Info: name:Tom age:18 sex:1
[ INFO] [1751531623.711220616]: Subscribe Person Info: name:Tom age:18 sex:1
[ INFO] [1751531624.710949645]: Subscribe Person Info: name:Tom age:18 sex:1
[ INFO] [1751531625.711448734]: Subscribe Person Info: name:Tom age:18 sex:1
[ INFO] [1751531626.711293033]: Subscribe Person Info: name:Tom age:18 sex:1
[ INFO] [1751531627.711274802]: Subscribe Person Info: name:Tom age:18 sex:1
[ INFO] [1751531628.710978357]: Subscribe Person Info: name:Tom age:18 sex:1
```

如果我们将roscore关掉，可以看到subscriber和publisher依然在接发。roscore代表了ROS Master，它帮助 subscriber和publisher建立通信连接，连接建立后退出舞台也没什么问题了。

但是关闭ROS Master就不能管理这个连接了。同时也无法看到rqt\_graph。

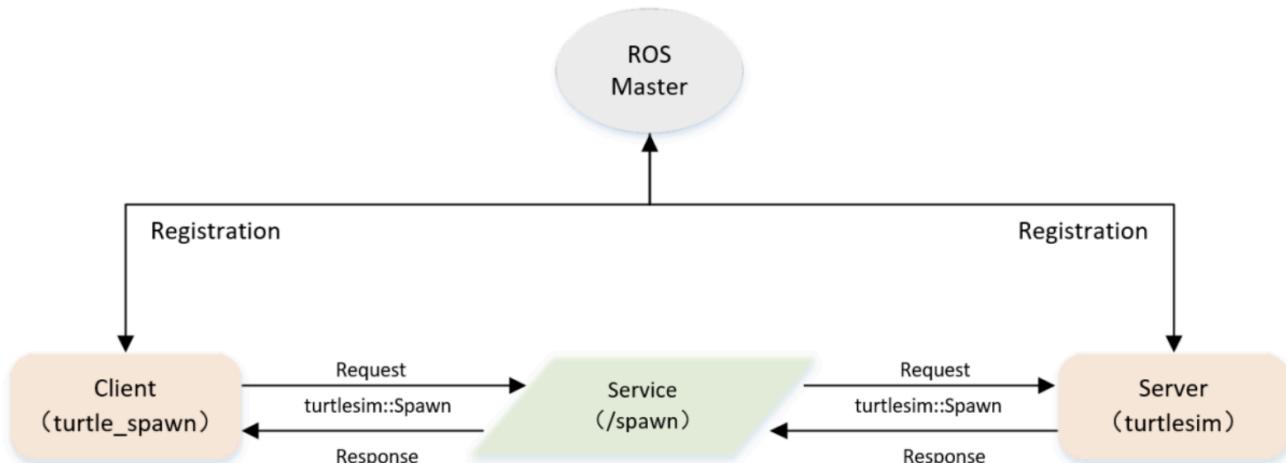
## 客户端Client的编程实现

### 1、模型图

Sever端是海龟仿真器/turtlesim，Client端是待实现的程序，其作为Response的节点，并产生Request的请求，发给 Server端。Server端收到Request请求后产生一只海龟，回馈一个Response给Client海龟产生是否成功。Service的名称为/spawn，中间传输消息的数据结构为turtlesim::Spawn。

ROS Master负责管理节点。

其中service和之前的topic是一样的，定义其名字和数据类型



### 2、创建功能包

这次我们创建新的功能包 learning\_service

```

cd ~/catkin_ws/src
catkin_create_pkg learning_service roscpp rospy std_msgs geometry_msgs turtlesim
//很重要，这个是要创建功能包的工作空间

```

### 3、创建代码并编译运行（C++）

编代码

配置编译规则

cmake就行

先配置CMakeLists.txt编译规则：

设置需要编译的代码和生成的可执行文件；

设置链接库；

```

add_executable(turtle_spawn src/turtle_spawn.cpp)
target_link_libraries(turtle_spawn ${catkin_LIBRARIES})

```

编译

## 服务端Server的编程实现

### 1 模型图

Server端本身是进行模拟海龟运动的命令端，它的实现是通过给海龟发送速度（Twist）的指令，来控制海龟运动（本身通过Topic实现）。

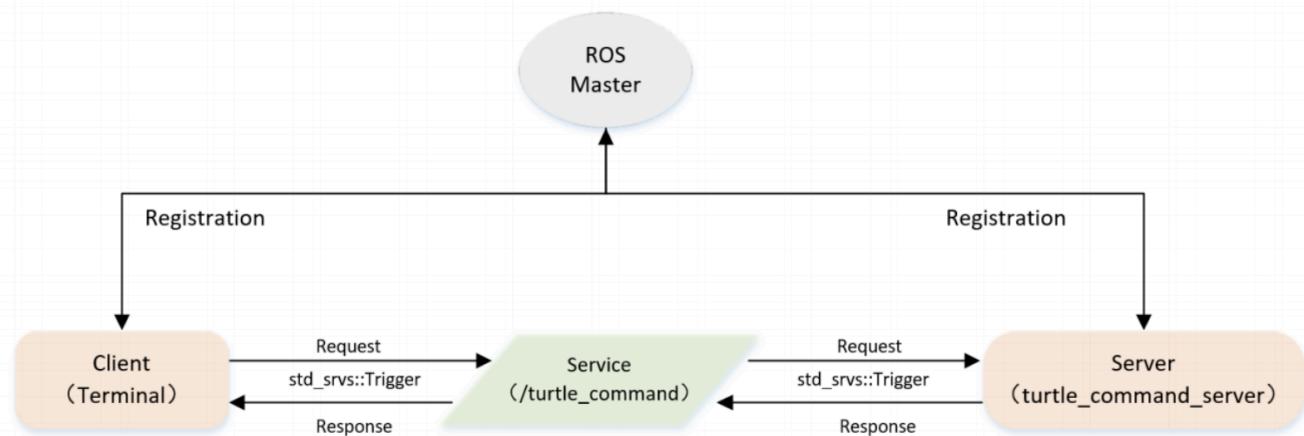
Client端相当于海龟运动的开关，其发布Request来控制Server端。

通过自定义名为 /turtle\_command 的Service实现，中间传输消息的数据类型为std\_srvs::Trigger（一种针对服务标准 std\_srvs下的数据定义）来通信。Trigger意为触发，通过Trigger信号来触发Server端的运动指令。

Server端接收这个Trigger信号后，可控制其是否要给海龟发送Twist指令，同时给Client发送Response反馈告诉它海龟的运动状态。

ROS Master负责管理节点。

所以本例既有Server端自己的Topic模式控制海龟运动，又有S/C之间的Service模式，包含两种通信模式的实现。



## 2 创建功能包

本节还是使用上节创建的 learning\_service 包来进行代码存放和编译。

## 3 创建代码并编译运行（C++）

### 如何实现一个服务器端Server

- 初始化ROS
- 创建一个Server实例
- 循环等待服务请求，进入回调函数
- 在回调函数中完成服务功能的处理，并反馈应答数据

### 创建服务端Server代码

其中在回调函数中给Client端的反馈数据res是与Trigger相对应的，我们可以查看一下Trigger的数据结构。可以使用 rossrv 指令查看service中的数据类型：

```
xxw@xxw-vpc:~/catkin_ws$ rossrv show std_srvs/Trigger
/!-----
bool success
string message
```

我们可以看到在定义srv的数据结构时，有一块三个连续破折号“---”。

这指的是破折号上方是定义Request部分，下方是定义Response部分。

在Trigger中，没有Request部分，即一个空内容的Request，这也解释了我们不需要在让海龟运动时给 /turtle\_command 传内容，直接传个空值 "{}" 就可以了（rosservice call /turtle\_command "{}"）。

这部分为下一节做铺垫，下一节将讲自定义服务数据srv。

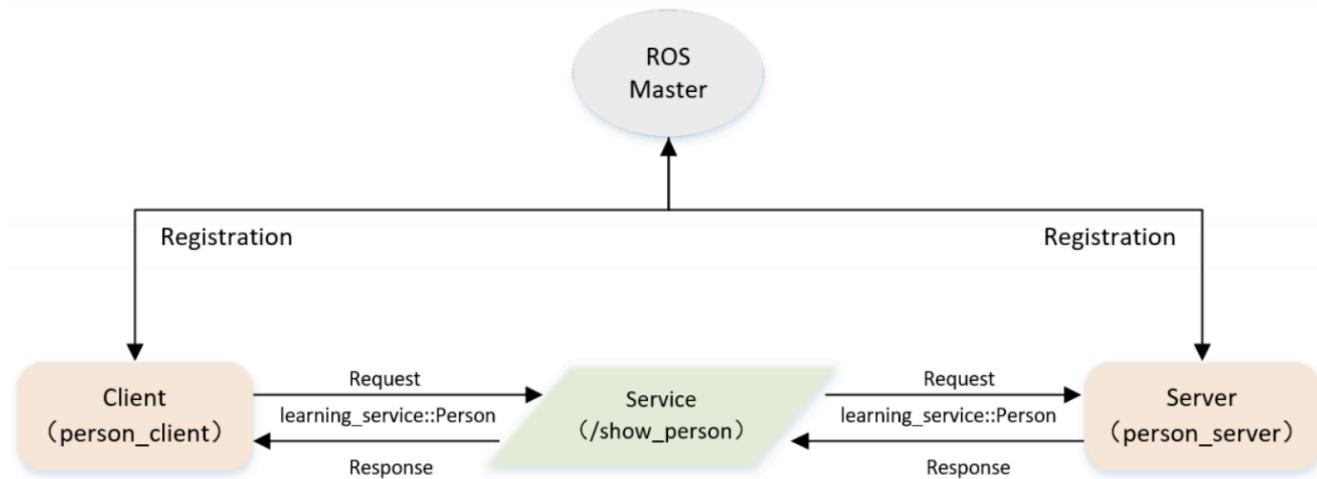
## 服务数据(srv)的定义与使用

### 1 模型图

在我们讲解了话题消息msg的定义与使用，在第8节的例子中我们曾自定义了一个消息类型“Person”以发布个人信息，Publisher发布个人信息，Subscriber接收个人信息。这个例子中，Publisher会不断地发信息，Subscriber不停地接数据，一开动就停不下来了，也是topic模式的缺陷。

本节我们使用Service模式用自定义的服务数据srv来实现，我们希望Request一次才发一次信息来显示。

如图，Client发布显示某个人的信息的Request，通过自定义的服务数据“Person”(learning::Person) 来发出去。Server端收到Request，显示这个人的具体信息，同时发Response向Client反馈显示结果。ROS Master负责管理节点。



## 2 创建功能包

本节还是使用上节创建的 learning\_service 包来进行代码存放和编译。

## 3 自定义服务数据

我们通过自定义srv文件来自定义服务数据。与之前自定义话题数据msg类似。

我们定义srv文件名为：Person.srv

1、在learning\_topic的功能包根目录下，新建文件夹 srv

并创建新并创建新文件 Person.srv，创建方法为使用 touch 命令在当前目录输入：

```
touch Person.srv
```

注意Person的"P"要大写。

2、我们把下面代码复制进Person.srv

```
string name
uint8 sex
uint8 age uint8
unknown = 0
uint8 male = 1
uint8 female = 2
---
string result
```

与之前Person.msg不同的是，多了破折号下面这个Response结果，上面的是Request内容。  
定义好srv数据接口后，就可以根据这个定义用C++或Python编译。

在package.xml中添加功能包依赖

添加动态生成程序的功能包依赖。

打开package.xml文件，将下面代码拷到文件指定位置：

```
<build_depend>message_generation</build_depend>
<exec_depend>message_runtime</exec_depend>
```

build\_depend为编译依赖，这里依赖的是一个会动态产生message的功能包  
exec\_depend为执行依赖，这里依赖的是一个动态runtime运行的功能包

## 在CMakeLists.txt中添加编译选项

为什么要添加编译选项：

1. 因为在package.xml添加了功能包编译依赖，在CMakeList.txt里的find\_package中也要加上对应的部分；
2. 需要将定义的Person.srv作为消息接口，针对它做编译；
3. 需要指明编译这个消息接口需要哪些ROS已有的包；  
有了这两个配置才可将定义的srv编译成不同的程序文件
4. 因为在package.xml添加了功能包执行依赖，在CMakeList.txt里的catkin\_package中也要加上对应的部分；

```
find_package( ..... message_generation)

add_service_files(FILES Person.srv)
generate_messages(DEPENDENCIES std_msgs)

catkin_package( ..... message_runtime)
```

## 编译生成语言相关文件

以上完成后，到工作空间根目录，编译：

```
catkin_make
```

编译完成后，我们可以在 devel/include/learning\_topic/ 下找到这个C++的头文件；  
也可以在 devel/lib/python3/dist-packages/learning\_topic/mrv 下找到Python的包。

## 4 创建代码并编译运行（C++）

### 创建代码

我们创建一个Client代码和一个Server代码，通过程序调用自己编译的py库。

接着常规操作

## 参数的使用与编程方法

### 1 概念图

在ROS Master中，存在一个参数服务器（Parameter Server），它是一个全局字典，即一个全局变量的存储空间，用来保存各个节点的配置参数。各个节点都可以对参数进行全局访问。

我们来看看参数服务器的使用方法。

## 2 创建功能包

本节建立一个新的功能包，命名为 learning\_parameter。在src下创建。

```
cd ~/catkin_ws/src  
catkin_create_pkg learning_parameter roscpp rospy std_srvs
```

## 3 参数命令行的使用(rosparam)

rosparam命令可以完成参数相关的大部分功能。

在ROS中，参数文件常以YAML文件的格式保存，形式如下：

### YAML参数文件

```
background_b: 255  
background_g: 86  
background_r: 69  
rosdistro: 'melodic'  
roslaunch:  
    uris: {host_hcx_vpc_43763: 'http://hcx-vpc:43763/'}  
rosversion: '1.14.3'  
run_id: 077058de-a38b-11e9-818b-000c29d22e4d
```

常用 rosparam 命令用法：

- 列出当前所有参数

\$ rosparam list

- 显示某个参数值

\$ rosparam get *param\_key*

- 设置某个参数值

\$ rosparam set *param\_key param\_value*

- 保存参数到文件

\$ rosparam dump *file\_name*

- 从文件读取参数

\$ rosparam load *file\_name*

- 删除参数

\$ rosparam delete *param\_key*

CSDN @takedachia

我们打开海龟仿真节点来试一下。

### 显示参数列表

```
rosparam list
```

```
xxw@xxw-vpc:~$ rosparam list
/rosdistro
/roslaunch_uris/host_xxw_vpc__41909
/rosversion
/run_id
/turtlesim/background_b
/turtlesim/background_g
/turtlesim/background_r
xxw@xxw-vpc:~$ rosparam get /turtlesim/background_b
255
xxw@xxw-vpc:~$ rosparam set /turtlesim/background_b 100
xxw@xxw-vpc:~$ rosservice call /clear "{}"
```

```
xxw@xxw-vpc:~$ █
```

观察一下这些参数，可以看到：

/turtlesim/background\_b

/turtlesim/background\_g

/turtlesim/background\_r

分别代表了小海龟的背景RGB颜色，目前是蓝色。

/rosdistro 为ros的版本代号

/roslaunch/uris/host\_ck\_vpc\_35381

/rosversion 当前ros的版本

/run\_id 进程的id号

## 显示某个参数值

例：

```
rosparam get /turtlesim/background_b
```

## 设定某个参数值

例：

```
rosparam set /turtlesim/background_b 100
```

把/turtlesim/background\_b的值改成100，再get一下可以看到已经改成了255

但是，海龟的背景颜色还没变。

我们需要发送一个clear的空内容服务请求：

```
rosservice call /clear "{}"
```

可以看到背景以及改过来了。

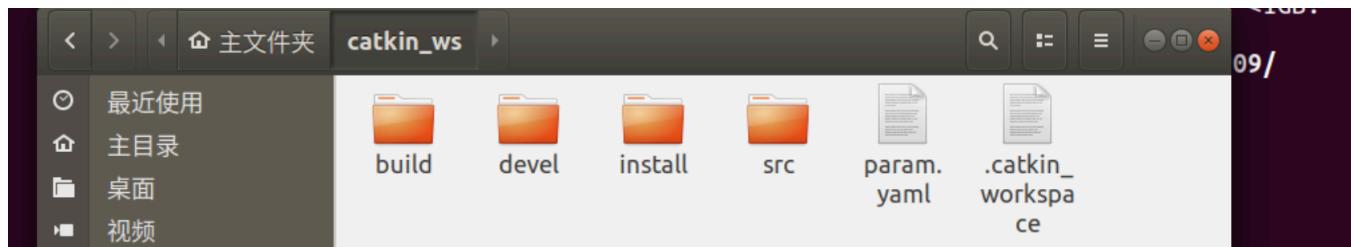
## 保存参数到文件

例：

```
rosparam dump param.yaml
```

将参数导出，保存为param.yaml文件。

默认保存位置为当前工作空间根目录下，我们可以打开看看：



## 从文件读取参数

我们可以直接在这个yaml文件里修改参数，然后导回去。

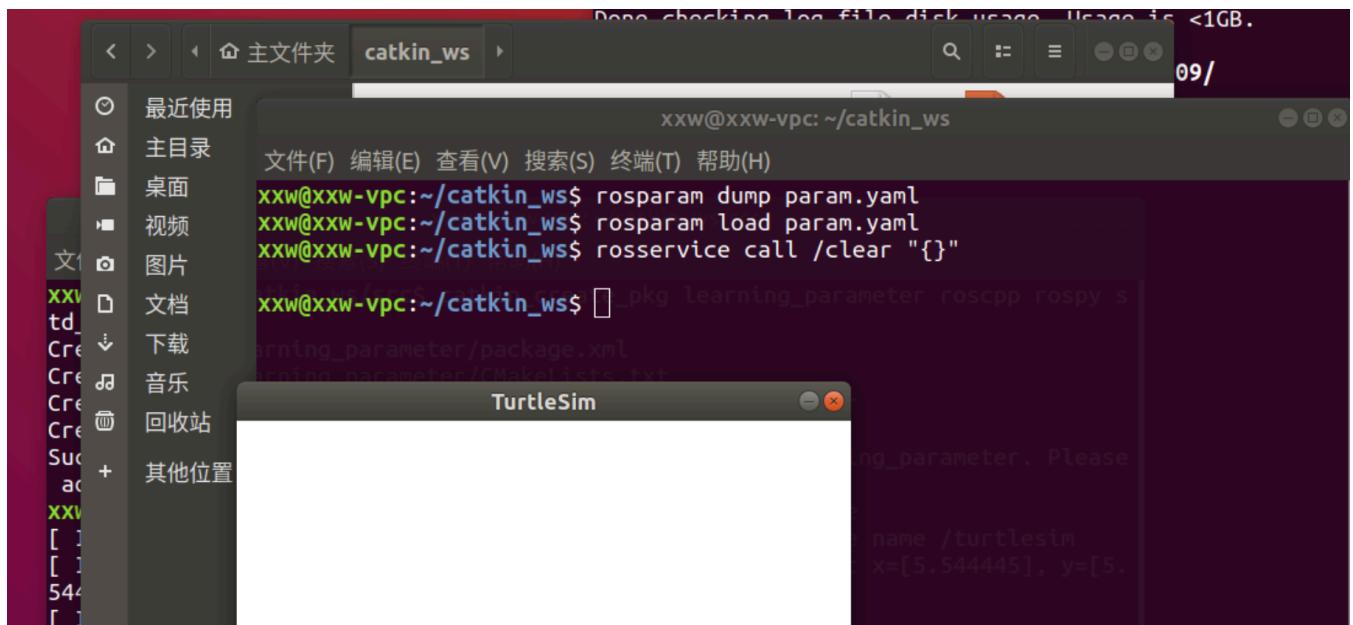
比如背景色改成 255, 255, 255 (白色)

导回去：

```
rosparam load param.yaml
```

clear一下，背景就变白色了

```
rosservice call /clear "{}"
```



## 删除参数

例：

```
rosparam delete /turtlesim/background_b
```

就可以删掉指定的参数。

删掉后可以用 `rosparam set ...` 设回来。

```
rosparam get /turtlesim/background_b
```

## 4 使用程序来使用参数 (C++)

这次我们使用程序来获取和设置参数。

如何获取/设置参数：

- `get`函数获取参数
- `set`函数设置参数

## 创建代码

# 问题：代码有问题，后面尝试解决！

## 7.4

# ROS中的坐标管理系统

看网页，这节基本是理论

[【ROS学习笔记】13.ROS中的坐标管理系统 ros world坐标系-CSDN博客](#)

## tf坐标系广播与监听的编程实现

讲解TF坐标变换的实现机制 广播与 监听的编程实现。

### 1 创建功能包

创建的 learning\_tf 包来进行代码存放和编译。

```
cd ~/catkin_ws/src  
catkin_create_pkg learning_tf roscpp rospy tf turtlesim
```

### 2 创建代码并编译运行（C++）

如何实现一个TF广播器：

- 定义TF广播器（TransformBroadcaster）
- 创建坐标变换值
- 发布坐标变换（sendTransform）

如何实现一个TF监听器：

- 定义TF监听器（TransformListener）
- 查找坐标变换（waitForTransform、lookupTransform）

## 创建代码

(源码：[https://github.com/guyuehome/ros\\_21\\_tutorials/tree/master/learning\\_tf/src](https://github.com/guyuehome/ros_21_tutorials/tree/master/learning_tf/src))

turtle\_tf\_broadcaster.cpp：根据实现的步骤，我们想要通过TF广播任意两个坐标系之间的位置关系，需要建立一个广播器，然后创建坐标的变换值，将这个变换矩阵的信息广播出去(插入TF tree)。

注意下面main函数时我们需要传入参数，参数从终端命令行输入（输入的参数包括节点名称 和 turtle\_name），下面运行部分会解释一下传入的参数的方法。这样我们从终端传参可以重复跑两遍这个C++程序分别对应turtle1和turtle2的坐标关系插入TF tree后，树会自动运算变换矩阵，后面我们就可以用监听器调用了。

turtle\_tf\_listener.cpp：根据步骤，从tf中获取任意两个坐标之间的位置关系(通过waitFor和lookup)，然后命令turtle2向turtle1以定义的速度(Twist)移动。

## 编译

先配置[CMakeLists.txt](#)编译规则：

- 设置需要编译的代码和生成的可执行文件；
- 设置链接库；

```
add_executable(turtle_tf_broadcaster src/turtle_tf_broadcaster.cpp)
target_link_libraries(turtle_tf_broadcaster ${catkin_LIBRARIES})

add_executable(turtle_tf_listener src/turtle_tf_listener.cpp)
target_link_libraries(turtle_tf_listener ${catkin_LIBRARIES})
```

然后编译：

```
cd ~/catkin_ws
catkin_make
```

## 运行

默认已经source，接着分别在每个终端运行。

```
roscore
rossrun turtlesim turtlesim_node
```

我们下面直接在命令行传入参数。

第1个参数：我们在turtle\_tf\_broadcaster.cpp定义节点时使用了"my\_tf\_broadcaster"的名字，我们使用\_\_name:=传入新的名字取代"my\_tf\_broadcaster"，这样避免名字重复（因为ROS中节点名字不能重复），这样就可以重复跑程序了。

第2个参数是turtle名称 turtle1 和 turtle2。

**这些代码要不同终端一直运行**

```
rossrun learning_tf turtle_tf_broadcaster name:=turtle1_tf_broadcaster /turtle1
rossrun learning_tf turtle_tf_broadcaster name:=turtle2_tf_broadcaster /turtle2
rossrun learning_tf turtle_tf_listener
```

上面完成后就会有一个海龟生成并跑向中间的第1只海龟。

```
rossrun turtlesim turtle_teleop_key
```

我们用键盘控制海龟，同样可以让第2只海龟追着我们跑。成后就会有一个海龟生成并跑向中间的第1只海龟。

# launch启动文件的使用方法

在之前的学习中，比如上一讲TF坐标广播和监听，启动程序非常麻烦，一共启动了6个终端窗口，并且涉及到终端向ROS的参数传递。

**launch启动文件**将解决这个问题，帮助我们快速部署、整合并启动程序。

## 1 launch文件结构

- 由XML语言写的，可实现多个节点的配置和启动。
- 不再需要打开多个终端用多个rosrun命令来启动不同的节点了
- 可自动启动ROS Master

## 2 launch文件语法

### 根元素

注：name为节点名称，会取代程序中初始化节点 init 时赋予的名字。

```
<launch>
    <node pkg="turtlesim" name="sim1" type="turtlesim_node"/>
    <node pkg="turtlesim" name="sim2" type="turtlesim_node"/>
</launch>
```

**<launch>** launch文件中的根元素采用<launch>标签定义

### 启动节点

```
<node pkg="package-name" type="executable-name" name="node-name" />
```

- <node>**
- pkg: 节点所在的功能包名称
  - type: 节点的可执行文件名称
  - name: 节点运行时的名称
  - output、respawn、required、ns、args

其他：

output: 控制某个节点node把日志信息打印到终端。

respawn: 节点奔溃后是否重启

required: 节点是否为必须节点，即改节点奔溃后须终止其他节点

ns: 自定义的命名空间，在自定义的命名空间中运行节点

args: 输入参数用

### 参数设置

param: 【在ROS参数服务器中】处理一个参数

rosparam: 【在ROS参数服务器中】处理多个参数

arg: 【不存在于ROS的参数服务器中】仅在launch文件中出现，可作为node运行时传的参数，如之前在终端输入指令时传的参数。

## 参数设置

**<param>** /  
**<rosparam>**

设置ROS系统运行中的参数，存储在参数服务器中。

`<param name="output_frame" value="odom"/>`

- name: 参数名
- value: 参数值

加载参数文件中的多个参数：

`<rosparam file="params.yaml" command="load" ns= "params" />`

**<arg>**

launch文件内部的局部变量，仅限于launch文件使用

`<arg name="arg-name" default="arg-value" />`

- name: 参数名
- value: 参数值

调用：

`<param name="foo" value="$(arg arg-name)" />`

`<node name="node" pkg="package" type="type" args="$(arg arg-name)" />`

CSDN @takedachia

更多标签参见：<http://wiki.ros.org/rosLaunch/XML>

## 3 示例

演示一些launch文件的实例。

需要先创建一个新的功能包 learning\_launch，包本身不需要添加别的依赖。

```
cd ~/catkin_ws/src  
catkin_create_pkg learning_launch
```

可以在learning\_launch下新建一个launch文件夹，来存放launch文件。

(源文件：[https://github.com/guyuehome/ROS\\_21\\_Tutorials/tree/master/learning\\_launch/launch](https://github.com/guyuehome/ROS_21_Tutorials/tree/master/learning_launch/launch))

可以把源文件中的几个launch文件拷贝到launch文件夹下。

打开launch文件后，文本编辑器的菜单下选择“查看”→“高亮模式”，选择XML可方便查看代码。

下面看一些示例。

## 7.4

Savage:

运行我给你那个镜像

Savage:

然后看fuel和apexnav

Savage:

之后就是那个双机

今日任务：运行镜像

## 7.5

运行了FUEL官方的

## 7.7

看下论文，学习apexnav

## 7.9

学习双机问题

Ubuntu20.4 nano加win11 wsl\_ubuntu

1、确定win11的ip1和nano的ip2

2、测试互通

3、 nano上(作为ROS Master)

设置ROS\_IP

```
echo "export ROS_IP=192.168.1.101" >> ~/.bashrc  
#ip应该是nano的实际ip
```

设置 ROS\_MASTER\_URI (指向自己):

```
echo "export ROS_MASTER_URI=http://192.168.1.101:11311" >> ~/.bashrc  
#ip应该是nano的实际ip
```

```
source ~/.bashrc
```

```
roscore
```

在win11上(作为ROS Client)

设置ROS\_IP

```
echo "export ROS_IP=192.168.1.101" >> ~/.bashrc  
#ip应该是win11的实际ip
```

设置 ROS\_MASTER\_URI (指向nano):

```
echo "export ROS_MASTER_URI=http://192.168.1.101:11311" >> ~/.bashrc  
#ip应该是nano的实际ip
```

```
source ~/.bashrc
```

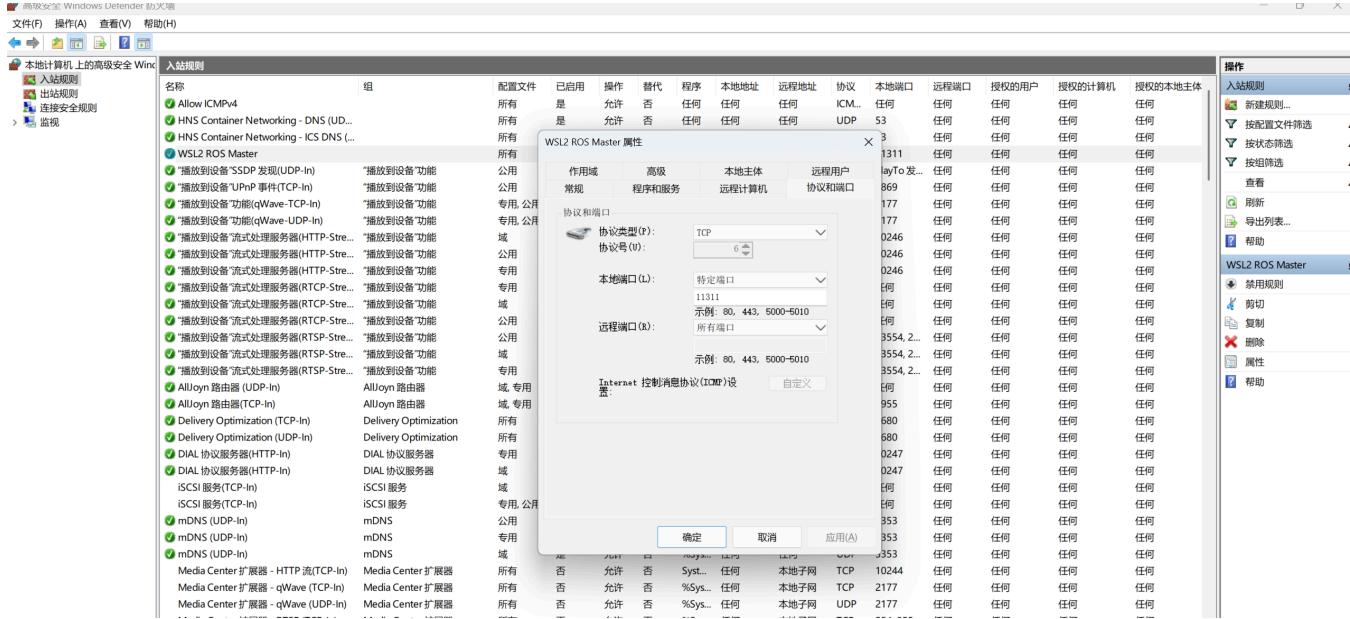
4、测试通信

```
#前置可以telnet ip加端口，这是更有效的  
#在Jetson Nano上启动一个ROS发布者节点:
```

```
rosrun rospy_tutorials talker.py
```

```
#在WSL Ubuntu上启动一个ROS订阅者节点:  
rosrun rospy_tutorials listener.py
```

## 其中重要的是需要配置防火墙问题



这两个需要在windows端配置 (可能需要管理员powershell来制作规则)