**中国大学慕课**

**实训文档**

**ROS机器人开发技术**

**实训：通讯架构（一）**

|  |  |
| --- | --- |
| **姓名:** | **组号:** |
| **实训负责人:** | **日期:** |

# 背景知识：

# 节点之间最重要的通信机制就是基于发布（publish）/订阅（subscrib）模型的消息（message）通信。每一个消息都是一种严格的数据结构，支持标准数据类型（整型、浮点型、布尔型等），也支持嵌套结构和数组，还可以根据需求由开发者自主定义。

# topic在ROS中使用最为频繁。在ROS中有两个节点，一个是发布者（publisher），一个是订阅者（Subscriber），两个节点分别发布、订阅同一个topic，启动顺序没有强制要求，此处假设publisher先启动，可大致分成以下七个步骤来分析建立通信的详细过程。

# 1.publisher注册

# publisher启动，向ROS Master注册发布者的信息，包含所发布消息的话题名，ROS Master会将节点的注册信息加入到注册列表中。

# 2.subscriber注册

# subscriber启动，同样向ROS Master注册订阅者的消息，包含需要订阅的话题名。

# 3.ROS Master进行信息匹配

# Master根据subscriber的订阅信息从注册列表中查找，如果没有找到匹配的发布者，则等待发布者的加入；如果找到匹配的发布者的信息，则向subscriber发送publisher的地址信息。

# 4.subscriber发送连接请求

# subscriber接收到Master发回的subscriber的地址信息，尝试向publisher发送连接请求，传输订阅的话题名、消息类型以及通信协议。

# 5.publisher确认连接请求

# publisher接收到连接请求后，继续向subscriber确认连接信息，其中包含了自身的TCP地址信息。

# 6.subscriber尝试与publisher建立

# publisher接收到确认信息后，使用TCP尝试与subscriber建立网络连接。

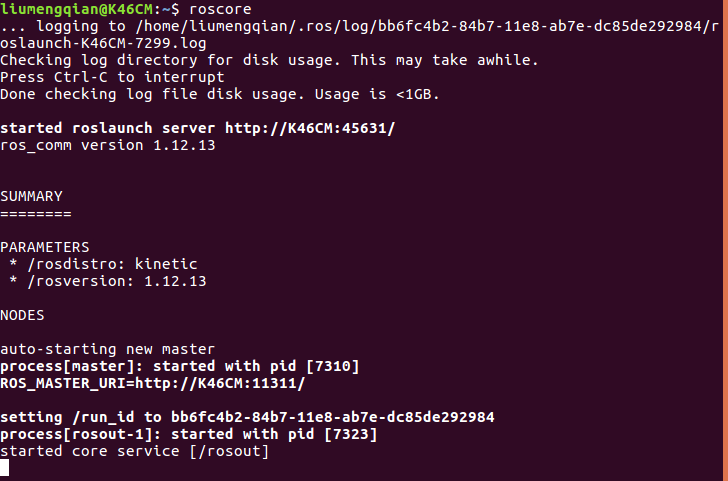
# 7.publisher向subscriber发送数据

# 成功建立连接后，publisher开始向subscriber发送话题消息数据。

# 任务1：同时打开两个turtlesim程序，使乌龟做相同的运动

# 子任务1：运行ROS的节点管理器——ROS Master(3min)

输出：



相关知识：

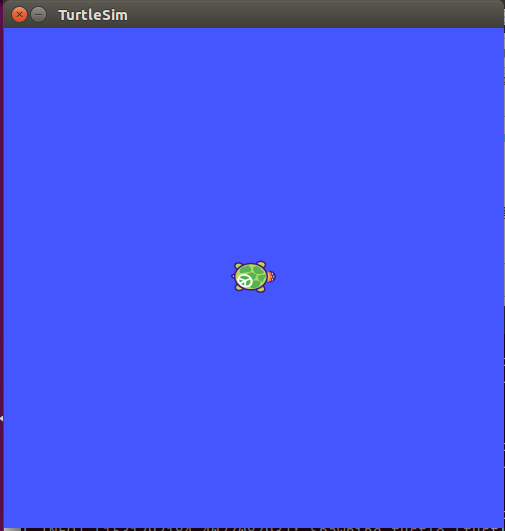
roscore启动时，同时启动的还有rosout和parameter server,其中rosout是负责日志输出的一个节点，其作用是告知用户当前系统的状态，包括输出系统的error、warning等等，并且将log记录于日志文件中，parameter server即是参数服务器，它并不是一个节点，而是存储参数配置的一个服务器，后文我们会单独介绍。每一次我们运行ROS的节点前，都需要把master启动起来，这样才能够让节点启动和注册。

**参考代码**

$ roscore

# 子任务2：打开turtlesim\_node节点(3min)

输出：



相关知识：

启动master之后，节点管理器就开始按照系统的安排协调进行启动具体的节点。节点就是一个进程，只不过在ROS中它被赋予了专用的名字里——node。在第二章我们介绍了ROS的文件系统，我们知道一个package中存放着可执行文件，可执行文件是静态的，当系统执行这些可执行文件，将这些文件加载到内存中，它就成为了动态的node。

ROS节点通常是可执行程序，用户可以在文件系统中通过cd命令手动切换到各个位置并启动节点。但是在一个很大的文件系统中，因为节点可能放在层级结构的文件系统中很深的位置，进入长的路径中会比较麻烦。因此ROS提供了一个命令行程序rosrun来寻找程序包中的可执行程序并向这个程序传递任何参数。

具体用法如下：

$ rosrun pkg\_name node\_name

rosrun允许你使用包名直接运行一个包内的节点而不需要知道这个包的路径。rosrun寻找package下名为node\_name的可执行程序，将可选参数传入。

到目前为止，ROS一共发布了11个版本，每一个版本都伴随着一个小乌龟吉祥物。下面我们就可以通过小乌龟例程开始认识ROS。在turtlesim功能包中，核心是节点turtlesim\_node，功能包提供一个可视化的乌龟仿真器，可以实现很多ROS基础功能的测试。

每一个ROS功能包都是一个独立的功能，其中可能包含一个或多个节点，这些功能对外使用topic, service, parameter作为接口。其他开发者在使用这个功能包时，可以不用关注内部的代码实现，只需要知道这些接口的类型和作用，就可以集成到自己的系统中。

下面我们通过这个简单的例程逐渐认识节点间的topic通讯方式。

如果没有安装turtlesim功能包，可以通过以下命令安装：

$ sudo apt-get install ros-kinetic-turtlesim

现在我们可以运行turtlesim包中的turtlesim\_node。

之前已经运行了roscore，不要关闭，打开一个新的终端，输入：

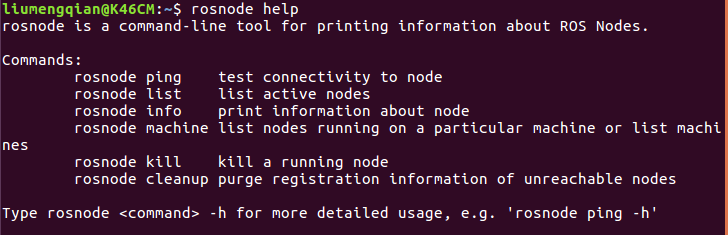
$ rosrun turtlesim turtlesim\_node

**参考代码**

$ rosrun turtlesim turtlesim\_node

# 子任务3：重新配置节点名称(13min)

输出：



相关知识：

如果忘了具体的命令，可以通过help来查看。

**参考代码**

$ rosnode help

# 查看node列表(2min)

输出：



相关知识：

rosout是用来收集和记录节点调试输出信息的，roscore命令后和Master一起启动，因此总是运行的。

**参考代码**

$ rosnode list

# 使用remapping argument改变节点名称(3min)

输出：



相关知识：

ROS可以通过命令行重新配置名称，关闭turtlesim窗口停止运行节点（或者回到rosrun turtlesim终端并使用Ctrl+c）。这一次使用remapping argument改变节点名称

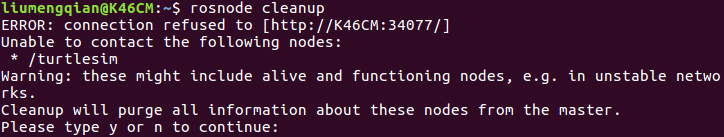
**参考代码**

$ rosrun turtlesim turtlesim\_node \_\_name:=my\_turtle

$ rosnode list

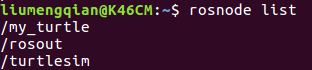
# 通过rosnode cleanup清除无用的节点(2min)

输出：



相关知识：

有人会出现这样的显示：



这是因为使用了Ctrl+c停止节点而非关闭重开窗口，可以通过rosnode cleanup清除无用的节点。

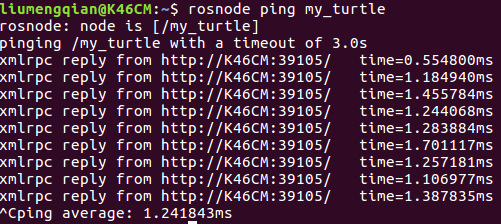
**参考代码**

$ rosnode cleanup

$ rosnode list

# 使用ping指令来测试这个节点(2min)

输出：



相关知识：

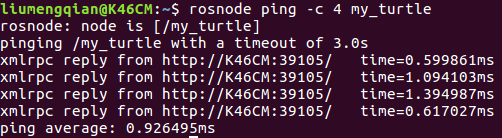
|  |  |
| --- | --- |
| rosnode ping nodename | 测试连接节点 |

**参考代码**

$ rosnode ping my\_turtle

# 使用指定次数(4次）的ping指令(2min)

输出：



相关知识：

|  |  |
| --- | --- |
| rosnode ping nodename | 测试连接节点 |
| ping -a/--all | 测试连接所有节点 |
| ping -c COUNT | 测试COUNT次（--all和-c不能同时使用） |
| rosnode cleanup | 清除不可到达节点的注册信息 |

**参考代码**

$ rosnode ping -c 4 my\_turtle

# 关闭节点，查看关闭节点后的节点列表(2min)

输出：



相关知识：

rosnode kill node\_name

结束某个或多个node

**参考代码**

$ rosnode kill /turtlesimrosnode kill /turtlesim

$ rosnode list

# 子任务10：建立launch文件(13min)

# 在catkin\_ws工作空间中建立一个package：beginner(3min)

输出：

相关知识：

**参考代码**

$ cd ~/catkin\_ws/src #改变目录

$ catkin\_create\_pkg beginner std\_msgs rospy roscpp #创建包，后面三个是依赖

$ cd ~/catkin\_ws #改变目录

$ catkin\_make #开始编译

然后开始创建launch文件：

先转到beginner package目录下：

输入：

$ roscd beginner

然后创建一个launch文件夹：

$ mkdir launch

$ cd launch

# 建立launch文件(6min)

输出：

相关知识：

<launch>

#表明这是一个launch文件

<group ns="turtlesim1">

<node pkg="turtlesim" name="sim" type="turtlesim\_node"/>

</group>

<group ns="turtlesim2">

<node pkg="turtlesim" name="sim" type="turtlesim\_node"/>

</group>

#这部分创建了两个节点分组，并以namespace标签来区分，其中一个命名为trutlesim1，一个命名为trutlesim2，两个组里面都使用相同的turtlesim节点并命名为’sim’，这样可以使我们同时启动两个turtlesim模拟器而不产生命名冲突。每个<node> 标签中包含了声明ROS图中节点的名字属性，该节点所在的包名，以及节点的类型，也就是可执行程序的文件名。

其中pkg定义节点所在的功能包的名称，type定义节点的可执行文件名称，name定义节点运行的名称，这三个是最常用的属性。

<node pkg="turtlesim" name="mimic" type="mimic">

<remap from="input" to="turtlesim1/turtle1"/>

<remap from="output" to="turtlesim2/turtle1"/>

</node>

#这里我们启动模仿节点，并将所有topic的输入输出分别重命名为trutlesim1和trutlesim2，这样就会使trutlesim2模仿trutlesim1。我们后面会看到这样设定的作用。

</launch>

#标志launch文件结束。

**参考代码**

<launch>

<group ns="turtlesim1">

<node pkg="turtlesim" name="sim" type="turtlesim\_node"/>

</group>

<group ns="turtlesim2">

<node pkg="turtlesim" name="sim" type="turtlesim\_node"/>

</group>

<node pkg="turtlesim" name="mimic" type="mimic">

<remap from="input" to="turtlesim1/turtle1"/>

<remap from="output" to="turtlesim2/turtle1"/>

</node>

</launch>

# 通过roslaunch命令来启动launch文件,然后在一个新终端中使用rostopic命令发送速度设定消息(2min)

输出：



相关知识：

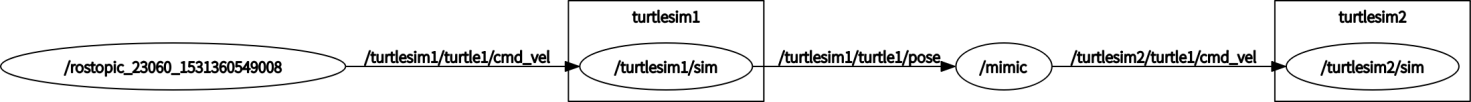
**参考代码**

$ roslaunch beginner turtlemimic.launch

$ rostopic pub /turtlesim1/turtle1/cmd\_vel geometry\_msgs/Twist -r 1 -- ‘[2.0, 0.0, 0.0]’ ‘[0.0, 0.0, -1.8]’

# 使用rqt\_graph(2min)

输出：



相关知识：

通过ROS图可以直观形象地看到各个节点之间的关系。

**参考代码**

$ rqt\_graph

# 任务2：创建msg文件(10min)

输出：

int64 num

相关知识：

**参考代码**

$ cd ~/catkin\_ws/src/beginner

$ mkdir msg

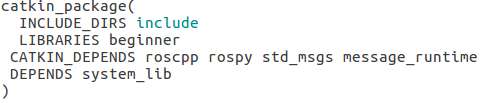
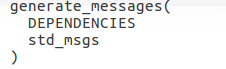
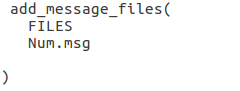
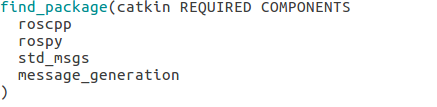
$ echo “int64 num” > msg/Num.msg

查看package.xml，确保它包含以下两条语句：

<build\_depend>message\_generation</build\_depend>

<exec\_depend>message\_runtime</exec\_depend>

CMakeLists.txt的配置:



# 子任务1：查看系统上所有的消息(2min)

输出：

相关知识：

rostopic list 列出当前所有的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

rostopic bw topic\_name 查看某个topic的带宽

rostopic hz topic\_name 查看某个topic的频率

rostopic find topic\_type 查找某个类型的topic

rostopic type topic\_name 查看某个topic的类型(msg)

**参考代码**

$ rosmsg list

# 子任务2：选择turtle\_actionlib/Velocity这个消息查看内容(2min)

输出：



相关知识：

rostopic list 列出当前所有的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

rostopic bw topic\_name 查看某个topic的带宽

rostopic hz topic\_name 查看某个topic的频率

rostopic find topic\_type 查找某个类型的topic

rostopic type topic\_name 查看某个topic的类型(msg)

**参考代码**

$rosmsg show turtle\_actionlib/Velocity

# 子任务3（思考题1）：查看刚刚创建的消息(2min)

输出：

int64 num

相关知识：

rostopic list 列出当前所有的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

rostopic bw topic\_name 查看某个topic的带宽

rostopic hz topic\_name 查看某个topic的频率

rostopic find topic\_type 查找某个类型的topic

rostopic type topic\_name 查看某个topic的类型(msg)

**参考代码**

$ rosmsg show beginner/Num #beginner：消息所在的package Num：消息名

# 子任务4（思考题2）：如果不知道Num消息所在的package，我们怎么找到这个msg？(2min)

输出：



相关知识：

ros提供给了我们一个更简便的方法，同样是使用rosmsg show命令，直接输入消息名，不加package名，也能找到我们想要的消息。

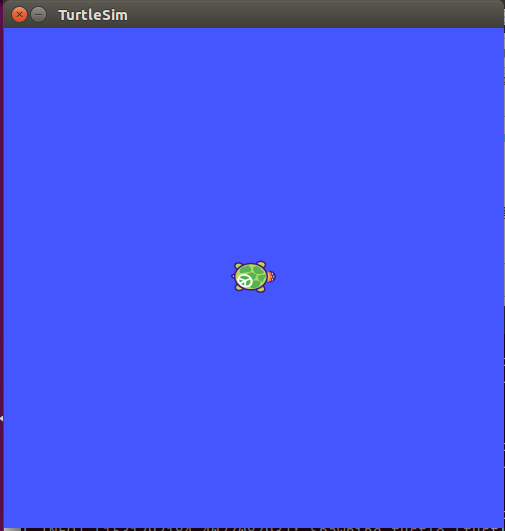
**参考代码**

$ rosmsg show Num

# 任务3：使用键盘控制小乌龟运动(4min)

# 子任务1：打开小乌龟节点(2min)

输出：



相关知识：

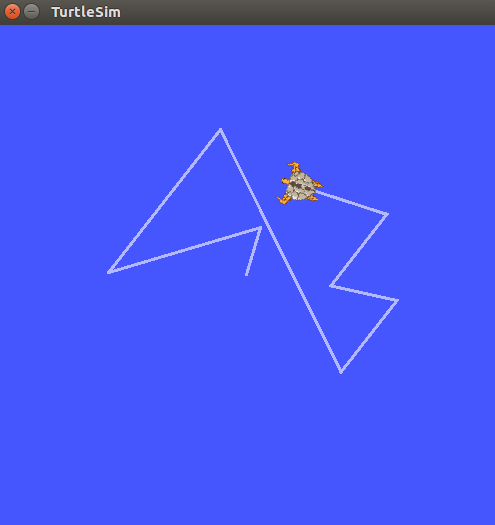
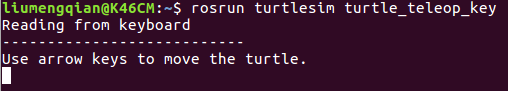
**参考代码**

$ roscore

$ rosrun turtlesim turtlesim\_node

# 子任务2：用方向键控制小乌龟行动(2min)

输出：



相关知识：

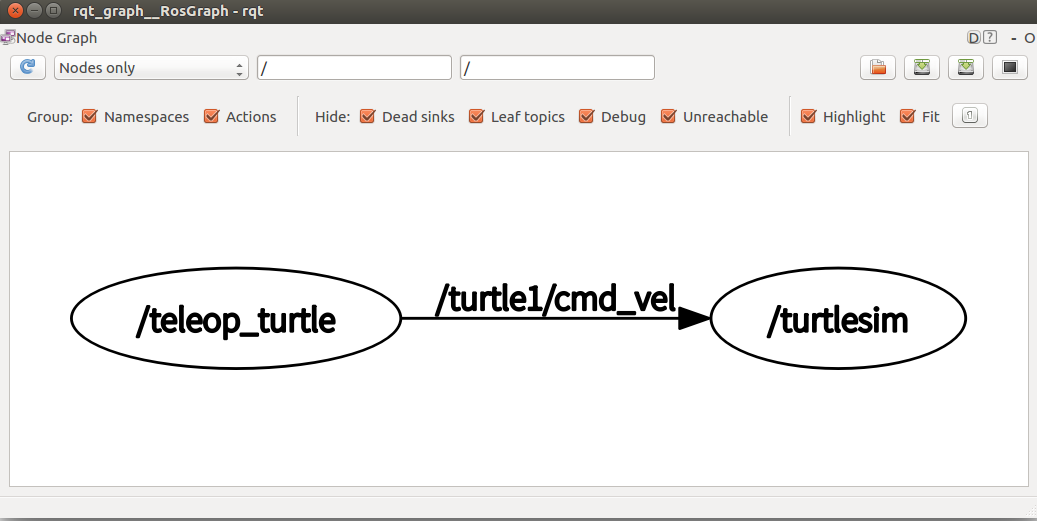
要注意需要选中turtle\_teleop\_key所在的终端窗口以确保按键输入能够被捕获。

**参考代码**

$ rosrun turtlesim turtle\_teleop\_key

# 子任务3：显示当前运行的节点和话题的ros图(2min)

输出：



相关知识：

如果将鼠标放在/turtle1/cmd\_vel上方，相应的ROS节点（蓝色和绿色）和话题（红色）就会高亮显示。意味着turtlesim\_node和turtle\_teleop\_key节点正通过一个名为/turtle1/cmd\_vel的话题来互相通信。

**参考代码**

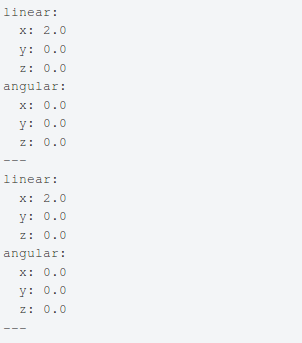
$ rosrun rqt\_graph rqt\_graph

或者

$ rqt\_graph

# 子任务4：显示topic中的消息(3min)

输出：



相关知识：

rostopic list 列出当前所有的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

rostopic bw topic\_name 查看某个topic的带宽

rostopic hz topic\_name 查看某个topic的频率

rostopic find topic\_type 查找某个类型的topic

rostopic type topic\_name 查看某个topic的类型(msg)

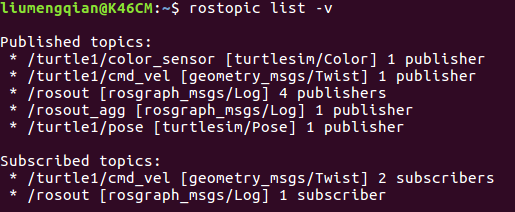
**参考代码**

$ rostopic echo /turtle1/cmd\_vel

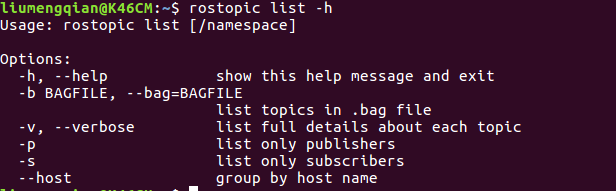
# 任务4：rostopic 练习

# 子任务1：列出当前所有订阅和发布的话题：显示出有关所发布和订阅的话题及其类型的详细信息(2min)

输出：



相关知识：

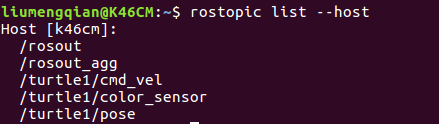


**参考代码**

$rostopic list -v

# 子任务2：列出当前所有订阅和发布的话题：以hostname为标准分组(2min)

输出：



相关知识：

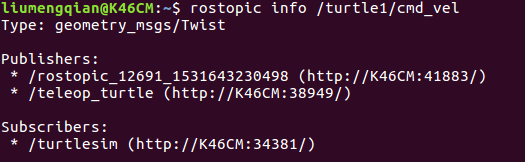
--host：以hostname为标准分组（环境变量）

**参考代码**

$rostopic list --host

# 子任务3：查看/turtle1/cmd\_vel这个topic的信息(2min)

输出：



相关知识：

rostopic list 列出当前所有的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

rostopic bw topic\_name 查看某个topic的带宽

rostopic hz topic\_name 查看某个topic的频率

rostopic find topic\_type 查找某个类型的topic

rostopic type topic\_name 查看某个topic的类型(msg)

**参考代码**

$ rostopic info /turtle1/cmd\_vel

# 子任务4：只想要查看某个topic的type信息(2min)

输出：



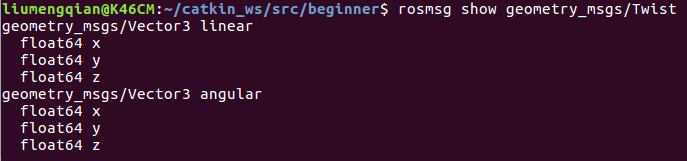
相关知识：

**参考代码**

$ rostopic type /cmd\_vel

# 子任务5：查看消息的详细情况，**显示**geometry\_msgs/Twist**信息**(2min)

输出：



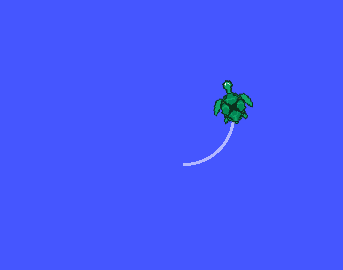
相关知识：

**参考代码**

$ rosmsg show geometry\_msgs/Twist

# 子任务6：把数据发布到当前某个正在广播的话题上，使乌龟运动一个弧度(3min)

输出：



相关知识：

用法：

rostopic pub [topic] [msg\_type] [args]

-1：表示只发布一个命令，然后马上退出

/turtle1/cmd\_vel：消息所发布到的话题的名称

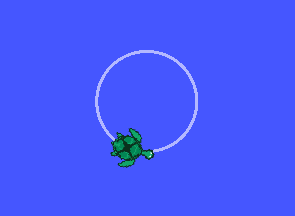
--：告诉命令选项解析器，接下来的参数部分都不是命令选项。

**参考代码**

$ rostopic pub -1 /turtle1/cmd\_vel geometry\_msgs/Twist -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'

# 子任务7：乌龟持续运动(3min)

输出：



相关知识：

-r：发布一个稳定的命令流，使乌龟不停运动。

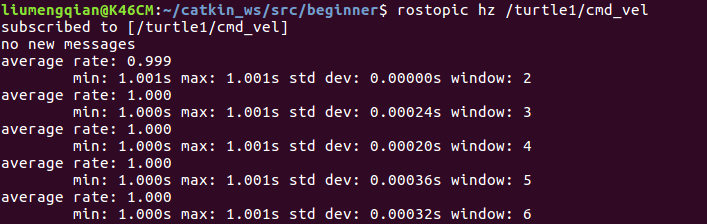
1：代表1hz。

**参考代码**

$ rostopic pub /turtle1/cmd\_vel geometry\_msgs/Twist -r 1 -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'

# 子任务8：查看turtlesim\_node发布/turtle1/cmd\_vel的频率(2min)

输出：



相关知识：

rostopic hz

用来查看数据发布的频率。

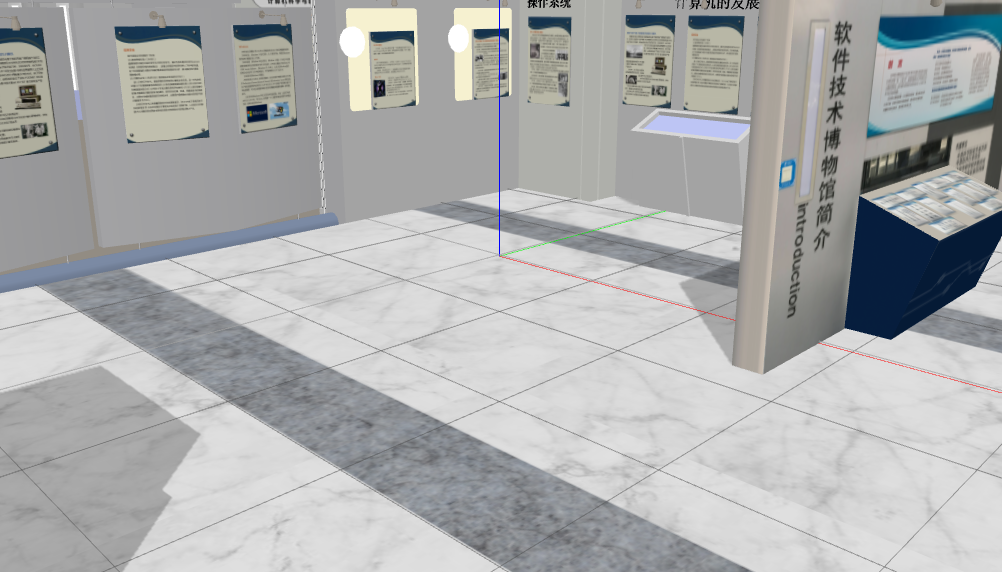
**参考代码**

$ rostopic hz /turtle1/cmd\_vel

# 任务5：仿真环境下使用topic控制机器人运动，并且观察相机采集到的图像

# 子任务1.打开仿真环境，启动模拟场景和机器人(2min)

输出：



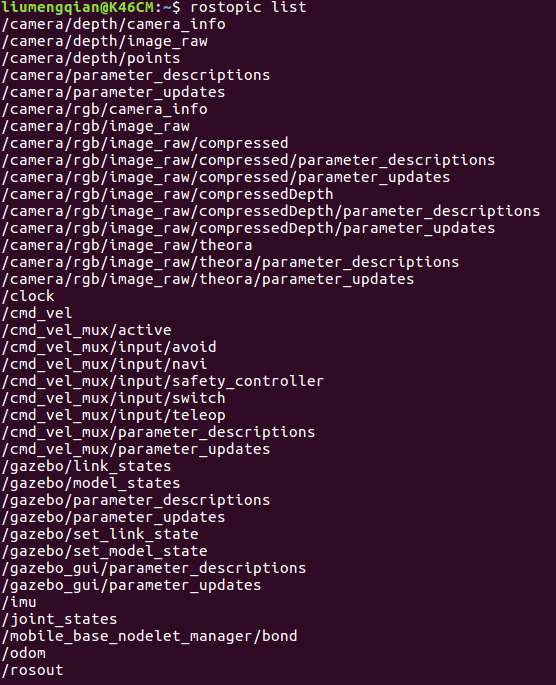
相关知识：

**参考代码**

$roslaunch robot\_sim\_demo robot\_spawn.launch

# 子任务2：查看当前模拟器中存在的topic(2min)

输出：



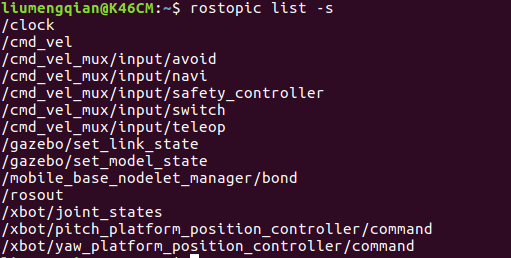
相关知识：

**参考代码**

$ rostopic list

# 子任务2.1：查看全部订阅者(2min)

输出：



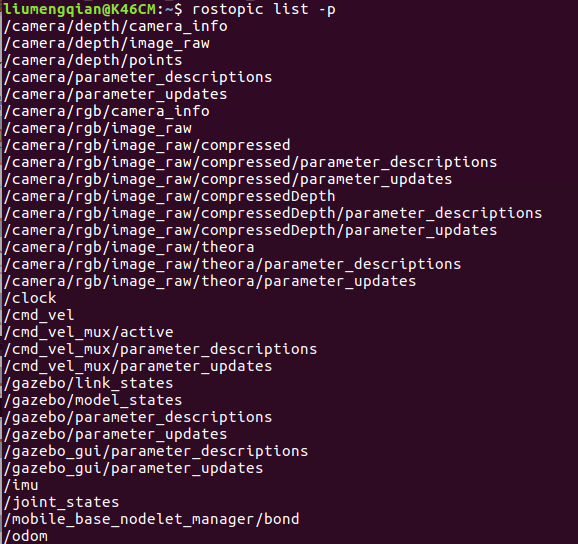
相关知识：

**参考代码**

$ rostopic list -s

# 子任务2.2：查看全部发布者(2min)

输出：



相关知识：

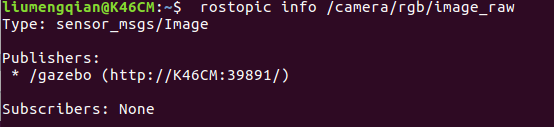
**参考代码**

$ rostopic list -p

# 子任务3：查看

# topic /camera/rgb/image\_raw (2min)

输出：



相关知识：

**参考代码**

$ rostopic info /camera/rgb/image\_raw

# 子任务3.1：

# 查看topic /camera/rgb/image\_raw的类型(2min)

输出：



相关知识：

**参考代码**

$ rostopic type /camera/rgb/image\_raw

# 子任务3.2：寻找所有sensor\_msgs/Image类型的topic(3min)

输出：



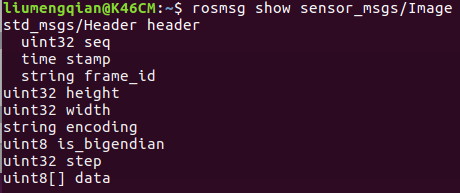
相关知识：

**参考代码**

$ rostopic find sensor\_msgs/Image

# 子任务4：查看/sensor\_msgs/Image类型的信息(2min)

输出：



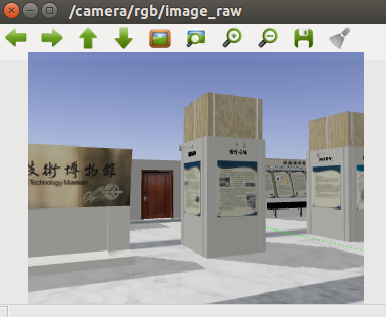
相关知识：

**参考代码**

$ rosmsg show sensor\_msgs/Image

# 子任务5：指定image\_view接收这个消息(5min)

输出：



相关知识：

尝试一下输入：

$ rosrun image\_view image\_view

发现不行，因此我们按照它的指引改变我们的命令：

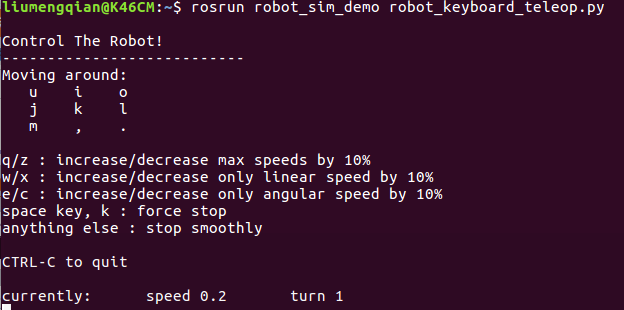
$ rosrun image\_view image\_view image:=<image topic> [transport]

**参考代码**

$ rosrun image\_view image\_view image:=/camera/rgb/image\_raw

# 子任务6：控制机器人的运动(2min)

输出：



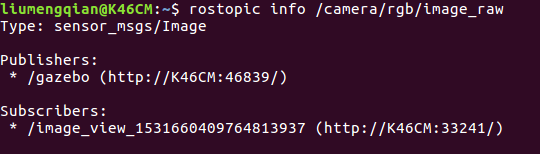
相关知识：

**参考代码**

$ rosrun robot\_sim\_demo robot\_keyboard\_teleop.py

# 子任务7：看/camera/rgb/image\_raw这个topic的信息(2min)

输出：



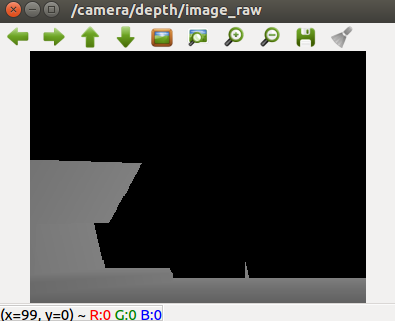
相关知识：

**参考代码**

$ rostopic info /camera/rgb/image\_raw

# 子任务8：显示深度相机的图像(3min)

输出：



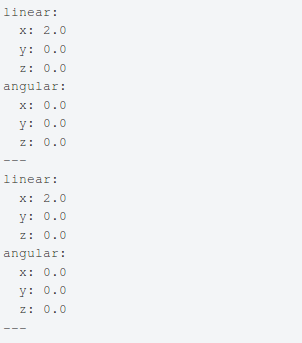
相关知识：

**参考代码**

$ rosrun image\_view image\_view image:=/camera/depth/image\_raw

# 子任务9：查看速度指令topic的内容：/cmd\_vel(2min)

输出：



相关知识：

**参考代码**

$ rostopic echo /cmd\_vel