DynamicX控制组、视觉组、rmua组联合培训
ROS导论 —— 第一课

广东工业大学DynamicX 机器人队

广东工业大学 Guangdong University of Technology 智能决策与协同控制研究所
广东工业大学 Guangdong University of Technology 实验教学部
UnitreeRobotics 杭州宇树科技有限公司

课程简介

第一课

- ROS 的构造与哲学
- ROS的master,nodes,和topics
- 命令行命令
- Catkin工作空间与构建系统
- Launch文件
- 仿真软件——Gazebo

第二课

- ROS 软件包结构
- 用Clion开发ROS程序
- ROS C++客户端库（roscpp）
- ROS 的subscriber与publisher
- ROS的参数服务器
- Rviz 数据可视化

第三课

- TF 坐标系转换系统
- rqt用户界面
- 机器人模型（URDF）
- 仿真描述（SDF）

第四课

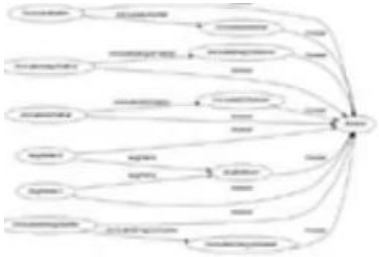
- ROS services
- ROS actions(actionlib)
- ROS time
- ROS bags

第一课概要

- ROS 的构造与哲学
- ROS的master,nodes,和topics
- 命令行命令
- Catkin工作空间与构建系统
- Launch文件
- 仿真软件——Gazebo

ROS 是什么？

ROS = Robot Operating System



Plumbing

- 进程管理
- 进程通信
- 设备驱动

Tools

- 仿真
- 可视化
- 图形用户界面
- 数据记录

Capabilities

- 控制
- 规划
- 感知
- 映射
- 操作

Ecosystem

- 被组织好的软件包
- 大量文档及教程

ROS的历史

- 最早在2007年发展于斯坦福人工智能实验室
- 从2013年起由OSRF管理
- 在今天被用在许多机器人，高校与企业里
- 已成为机器人编程的事实规范

ROS的哲学

- 点对点

  分立的程序通过预先定义的接口（例如ROS messages,services）进行通信。
- 分散式

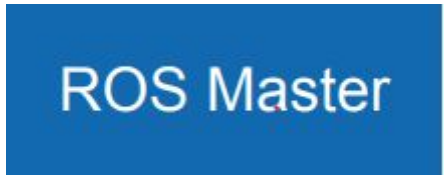
  多个程序可运行在多台电脑上并通过网络进行通信。
- 可用多种语言编写

  ROS的模块可以用多种语言编写，前提是存在这种语言的客户端库（C++,Python，Java等）
- 轻量级

  独立的库被ROS层包装着。
- 免费且开源

  大多数ROS软件包是开源的并且可免费使用。

ROS Master



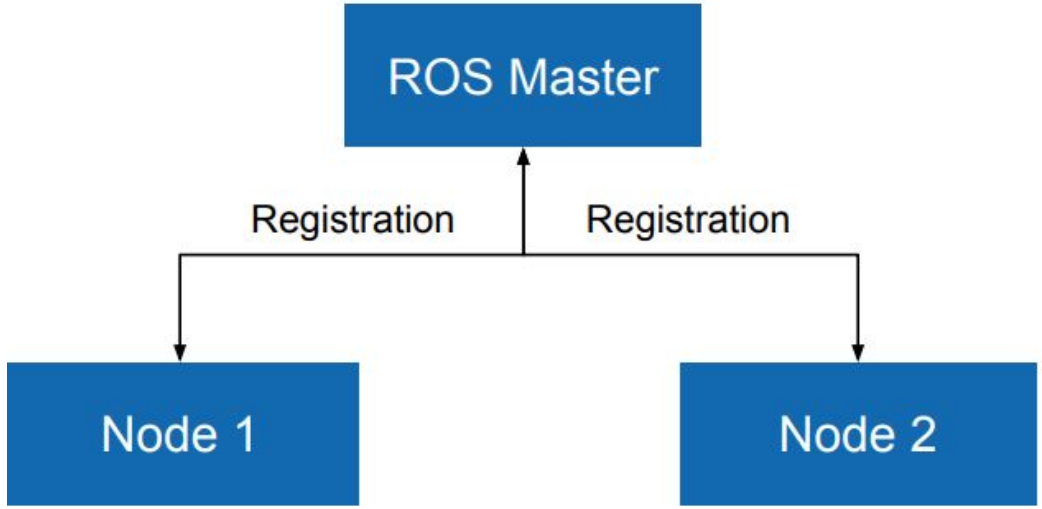
- 管理node之间的通信
- 每个node都需要在启动时向master注册

启动一个master

```
> roscore
```

ROS Node（下译为节点）

- 单用途的可执行程序
- 单独编译、执行和管理
- 以软件包的形式组织起来



运行一个节点

```
> rosrun package_name node_name
```

查看活跃的节点

```
> rosnode list
```

查看一个节点的信息

```
> rosnode info node_name
```

ROS Topic（下译为话题）

- 节点间可通过topic进行通信
  - 节点可以发布或订阅一个topic
  - 通常有一个发布者和若干个订阅者
- topic是message流的名字

列出活跃的topic

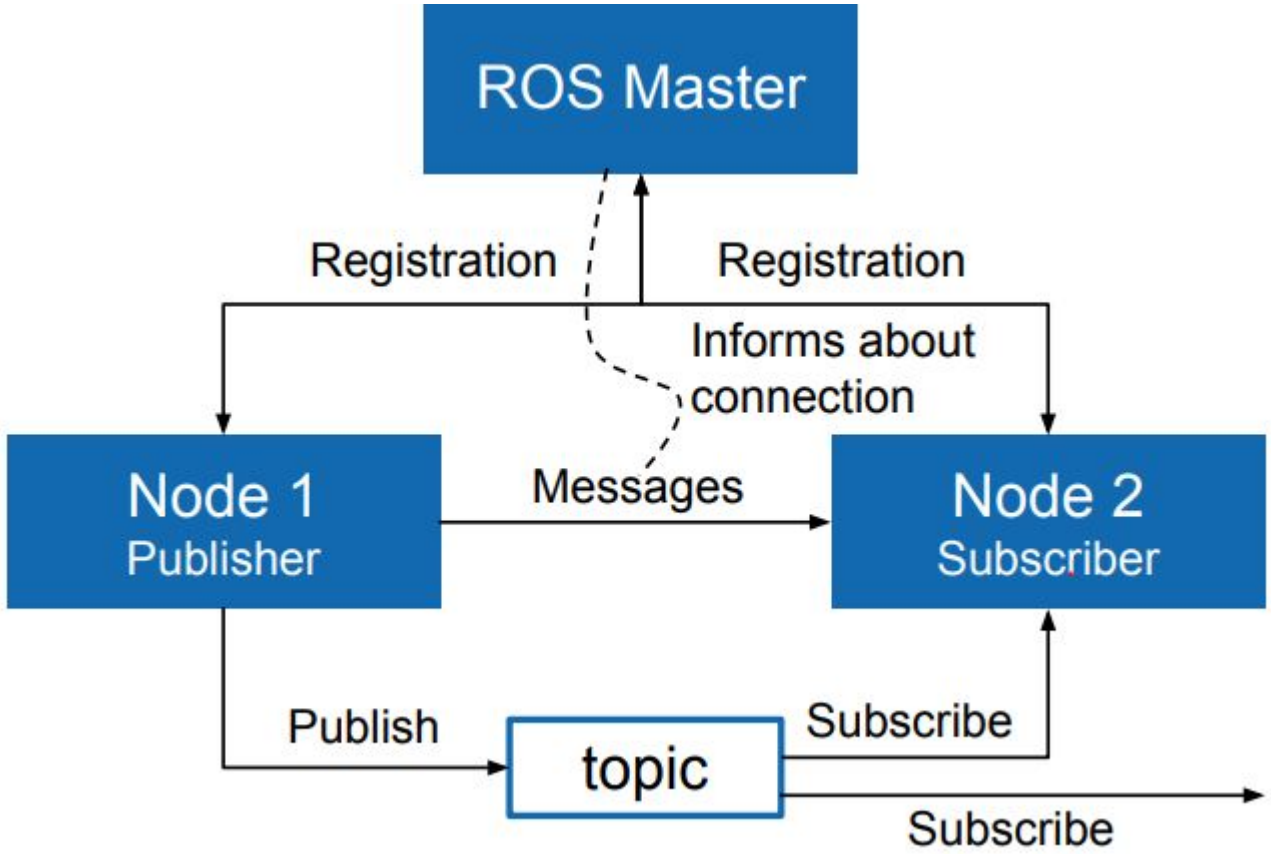
```
> rostopic list
```

将topic上的消息打印出来

```
> rostopic echo /topic
```

显示topic的信息

```
> rostopic info /topic
```



ROS Message

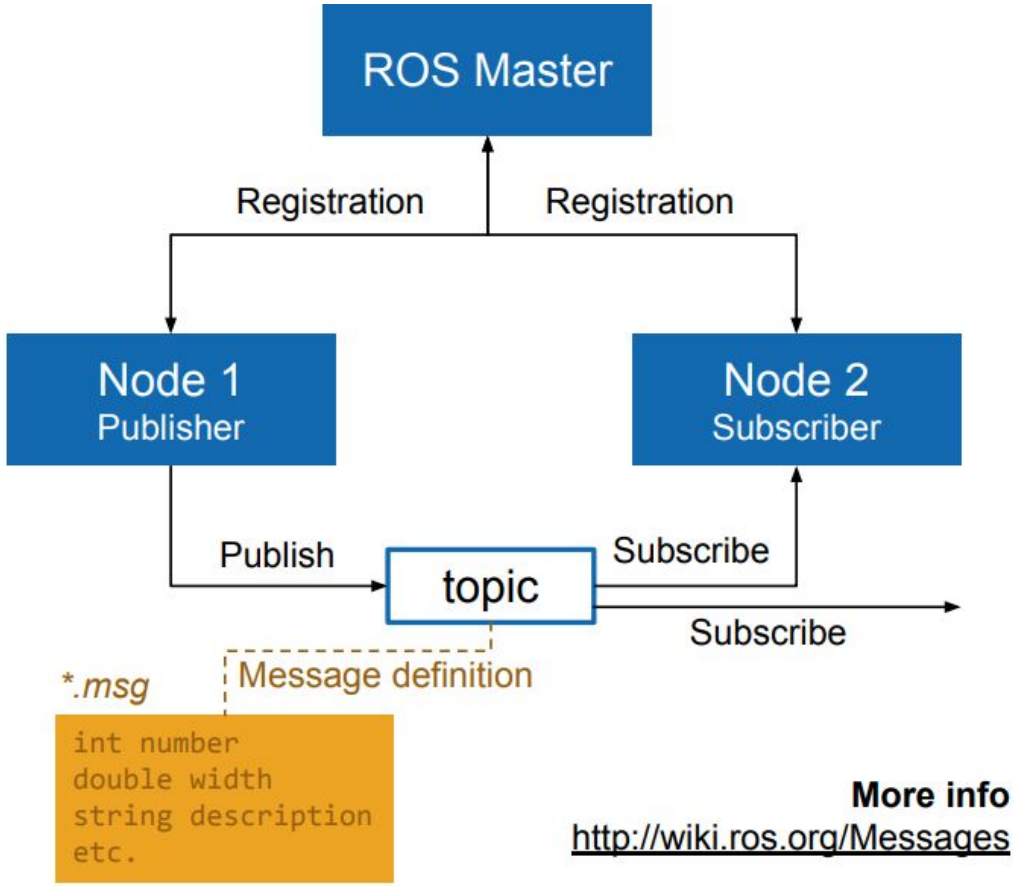
- 是定义topic类型的数据结构
- 可由int，float，bool，string等数据类型组成
- 在后缀为msg的文件中定义

查看topic类型

```
> rostopic type /topic
```

发布message到topic上

```
> rostopic pub /topic type data
```



ROS Message

位姿时间戳消息示例

geometry_msgs/Point.msg

```
float64 x
float64 y
float64 z
```

sensor_msgs/Image.msg

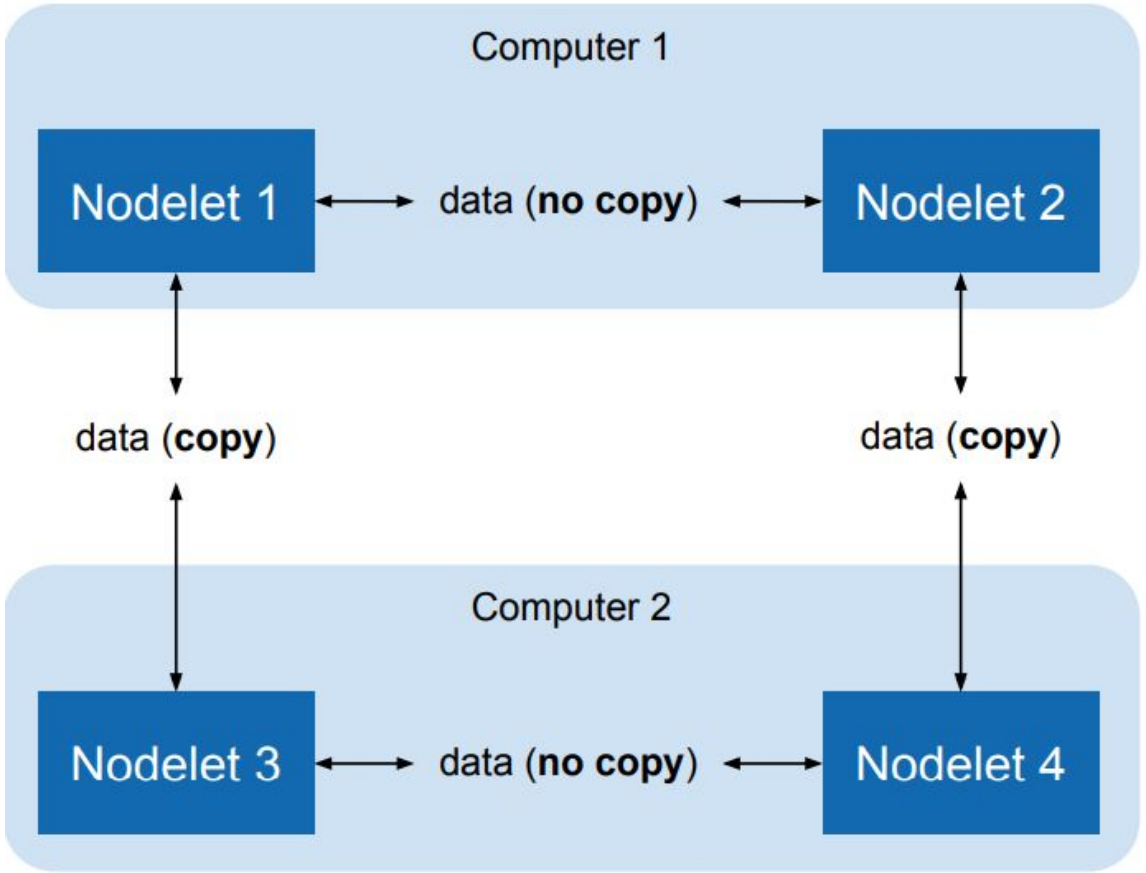
```
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
uint32 height
uint32 width
string encoding
uint8 is_bigendian
uint32 step
uint8[] data
```

geometry_msgs/PoseStamped.msg

```
std_msgs/Header header
 uint32 seq
 time stamp
 string frame_id
geometry_msgs/Pose pose
  geometry_msgs/Point position
    float64 x
    float64 y
    float64 z
  geometry_msgs/Quaternion orientation
    float64 x
    float64 y
    float64 z
    float64 w
```

ROS Nodelet

- 与ROS的node是相同的概念
- 当多个nodelet运行在同一台机器上时，相比node减少通信开销
- 优先尝试使用node
- nodelet实现上更加复杂



示例

第一个终端——启动一个roscore

- 使用以下命令启动roscore

```
> roscore
```

```
student@ubuntu:~/catkin_ws$ roscore
... logging to /home/student/.ros/log/6c1852aa-e961-11e6-8543-000c297bd368/ros
launch-ubuntu-6696.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ubuntu:34089/
ros_comm version 1.11.20


SUMMARY
========

PARAMETERS
 * /rosdistro: indigo
 * /rosversion: 1.11.20

NODES

auto-starting new master
process[master]: started with pid [6708]
ROS_MASTER_URI=http://ubuntu:11311/

setting /run_id to 6c1852aa-e961-11e6-8543-000c297bd368
process[rosout-1]: started with pid [6721]
started core service [/rosout]
```

示例

第二个终端——启动一个talker node

- 用以下指令启动一个talker node

```
> rosrun roscpp_tutorials talker
```

```
student@ubuntu:~/catkin_ws$ rosrun roscpp_tutorials talker
[ INFO] [1486051708.424661519]: hello world 0
[ INFO] [1486051708.525227845]: hello world 1
[ INFO] [1486051708.624747612]: hello world 2
[ INFO] [1486051708.724826782]: hello world 3
[ INFO] [1486051708.825928577]: hello world 4
[ INFO] [1486051708.925379775]: hello world 5
[ INFO] [1486051709.024971132]: hello world 6
[ INFO] [1486051709.125450960]: hello world 7
[ INFO] [1486051709.225272747]: hello world 8
[ INFO] [1486051709.325389210]: hello world 9
```

示例

第三个终端——分析talker node

列出活跃的node

```
> rosnode list
```

```
student@ubuntu:~/catkin_ws$ rosnode list
/rosout
/talker
```

查看talker node的有关信息

```
> rosnode info /talker
```

```
student@ubuntu:~/catkin_ws$ rosnode info /talker
--------------------------------------------------------------------------------
Node [/talker]
Publications:
 * /chatter [std_msgs/String]
 * /rosout [rosgraph_msgs/Log]

Subscriptions: Nonc

Services:
 * /talker/get_loggers
 * /talker/set_logger_level
```

查看chatter topic的信息

```
> rostopic info /chatter
```

```
student@ubuntu:~/catkin_ws$ rostopic info /chatter
Type: std_msgs/String

Publishers:
 * /talker (http://ubuntu:39173/)

Subscribers: None
```

示例

第三个终端——分析chatter topic

检查chatter topic的消息类型

```
> rostopic type /chatter
```

```
student@ubuntu:~/catkin_ws$ rostopic type /chatter
std_msgs/String
```

查看发布在topic上的消息

```
> rostopic echo /chatter
```

```
student@ubuntu:~/catkin_ws$ rostopic echo /chatter
data: hello world 11874
---
data: hello world 11875
---
data: hello world 11876
```

分析消息发布频率

```
> rostopic hz /chatter
```

```
student@ubuntu:~/catkin_ws$ rostopic hz /chatter
subscribed to [/chatter]
average rate: 9.991
        min: 0.099s max: 0.101s std dev: 0.00076s window: 10
average rate: 9.996
        min: 0.099s max: 0.101s std dev: 0.00069s window: 20
```

示例

第四个终端——启动一个listener node

运行一个listener node

```
> rosrun roscpp_tutorials listener
```

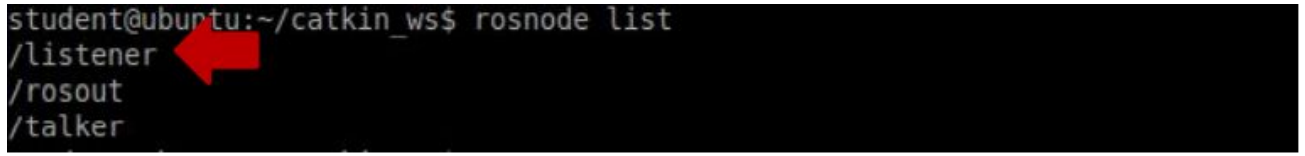
```
student@ubuntu:~/catkin_ws$ rosrun roscpp_tutorials listener
[ INFO] [1486053802.204104598]: I heard: [hello world 19548]
[ INFO] [1486053802.304538827]: I heard: [hello world 19549]
[ INFO] [1486053802.403853395]: I heard: [hello world 19550]
[ INFO] [1486053802.504438133]: I heard: [hello world 19551]
[ INFO] [1486053802.604297608]: I heard: [hello world 19552]
```

示例

第三个终端——分析

查看这个新的listener node

```
> rosnode list
```



展示多个node通过chatter topic建立的连接

```
> rostopic info /chatter
```

示例

第三个终端——通过终端发布消息

在运行talker node的终端按下ctrl+c可以关掉这个node

用命令行发布你自己的消息

```
> rostopic pub /chatter std_msgs/String
“data: ‘Hello DynamicX’”
```

查看listener的输出

```
student@ubuntu:~/catkin_ws$ rostopic pub /chatter std_msgs/String "data: 'ETH
Zurich ROS Course'"
publishing and latching message. Press ctrl-C to terminate
```

```
[ INFO] [1486054667.604322265]: I heard: [hello world 28202]
[ INFO] [1486054667.704264199]: I heard: [hello world 28203]
[ INFO] [1486054667.804389058]: I heard: [hello world 28204]
[ INFO] [1486054707.646404558]: I heard: [ETH Zurich ROS Course]
```

ROS工作空间环境

加载默认工作空间

```
> source /opt/ros/noetic/setup.bash
```

覆盖你的工作空间

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

检查你的工作空间

```
> echo $ROS_PACKAGE_PATH
```

Catkin 构建系统

- catkin 是ROS的构建系统用来生成可执行文件，库，和接口
- 用catkin build代替catkin_make（需安装catkin-tools）
- 在你的工作空间下，使用以下指令构建一个软件包

```
> catkin build package_name
```

catkin 构建系统

caktin工作空间包含以下文件夹

这个文件夹存放着源代码，是你可以克隆，创建，编辑你想要构建的软件包的源代码的地方

build文件夹是调用Cmake构建src文件夹中的软件包的地方。缓存信息和中间文件会保存到这里

devel文件夹存放编译好的目标（可执行程序，库）

catkin 构建系统

可以用以下指令检查catkin工作空间的设置

```
> catkin config
```

可用以下指令设置CMake构建类型为Release(或Debug)

```
> catkin build –cmake-args
–DCMAKE_BUILD_TYPE=Release
```

```
student@ubuntu:~/catkin_ws$ catkin config
--------------------------------------------------------------------------------
Profile:                     default
Extending:             [env] /opt/ros/indigo:/home/student/catkin_ws/devel
Workspace:                   /home/student/catkin_ws
--------------------------------------------------------------------------------
Source Space:       [exists] /home/student/catkin_ws/src
Log Space:          [exists] /home/student/catkin_ws/logs
Build Space:        [exists] /home/student/catkin_ws/build
Devel Space:        [exists] /home/student/catkin_ws/devel
Install Space:      [unused] /home/student/catkin_ws/install
DESTDIR:            [unused] None
--------------------------------------------------------------------------------
Devel Space Layout:          linked
Install Space Layout:        None
--------------------------------------------------------------------------------
Additional CMake Args:       -GEclipse CDT4 - Unix Makefiles -DCMAKE_CXX_COMP
ILER_ARG1=-std=c++11 -DCMAKE_BUILD_TYPE=Release
Additional Make Args:        None
Additional catkin Make Args: None
Internal Make Job Server:    True
Cache Job Environments:      False
--------------------------------------------------------------------------------
Whitelisted Packages:        None
Blacklisted Packages:        None
--------------------------------------------------------------------------------
Workspace configuration appears valid.
--------------------------------------------------------------------------------
```

Already setup in the provided installation.

示例

打开一个终端并跳转到你的工作空间下

```
> cd ~/catkin_ws/src
```

克隆git仓库

```
> git clone https://github.com/leggedrobotics/
ros_best_practices.git
```

示例

跳转到你的工作空间下

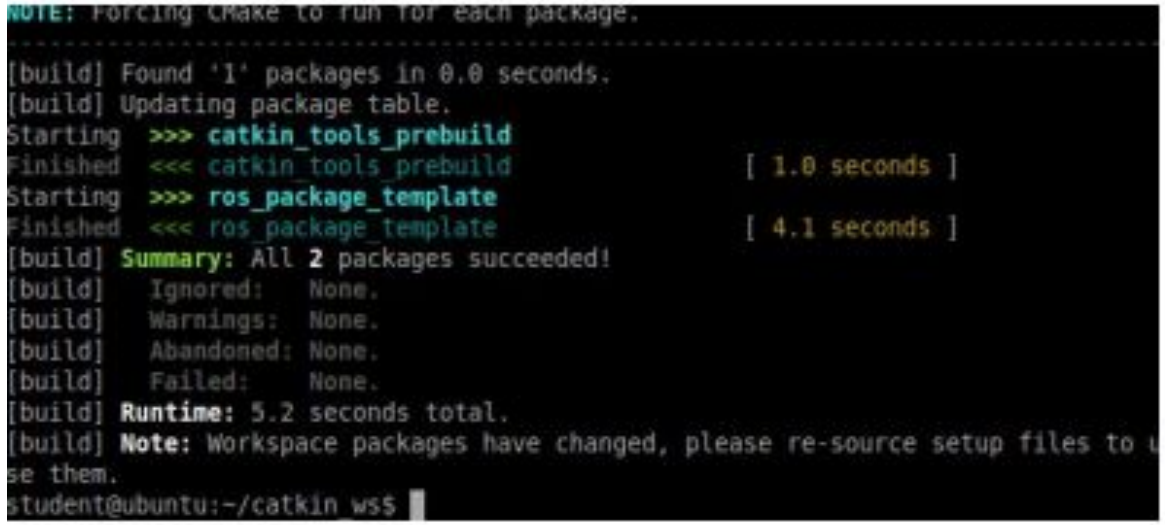
```
> cd ~/catkin_ws
```

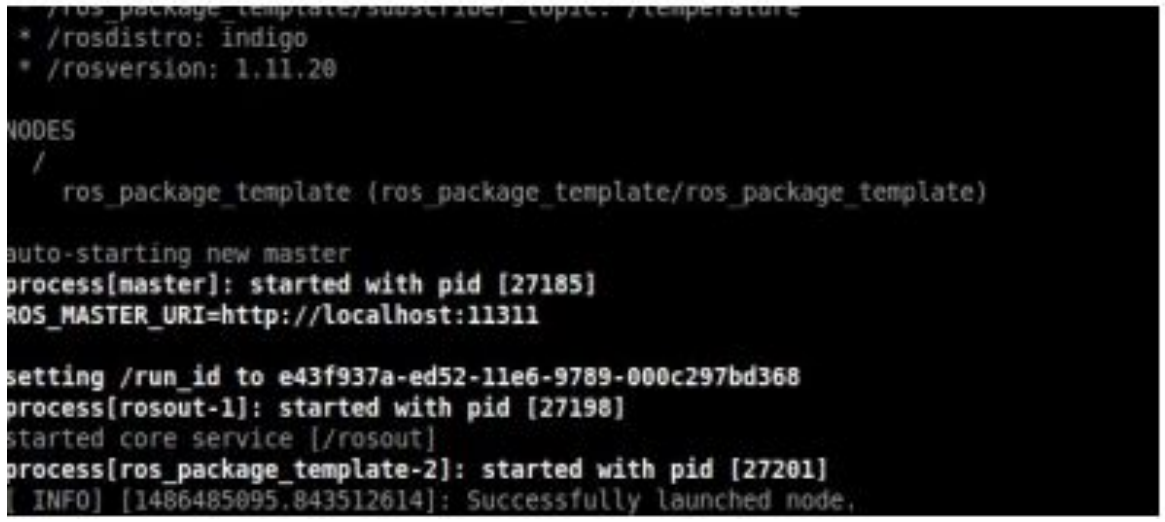
构建软件包

```
> catkin build ros_package_template
```

启动这个软件包里的node

```
> roslaunch ros_package_template
ros_package_template.launch
```





ROS Launch

- launch 是一个启动多个节点的工具（同时设置参数）
- 写在后缀为launch的文件
- 如果roscore没有启动，launch会自动启动roscore

启动一个launch文件

```
> roslaunch package_name file_name.launch
```

ROS Launch

文件结构

Attention when copy & pasting code from the internet

talker_listener.launch

```
<launch>
    <node name="listener" pkg="roscpp_tutorials" type="listener" output="screen"/>
    <node name="talker" pkg="roscpp_tutorials" type="talker" output="screen"/>
</launch>
```

Notice the syntax difference for self-closing tags: <tag></tag> and <tag/>

- launch: launch文件的根元素
- node：每个<node>标签定义了一个要启动的节点
- name：node的名字
- pkg：node所在的软件包
- type：node的类型，它必须和一个可执行文件的名字相同
- output：定义在哪里输出日志信息（screen：终端，log：日志文件）

ROS Launch

参数

- 用<arg>标签来创建一个可复用的launch文件，此标签可像一个参数一样发挥作用(即默认选项)

```
<arg name="arg_name" default="default_value"/>
```

- 在launch文件中使用这些参数

```
$(arg arg_name)
```

- 启动时，可以这种方式设置参数

```
> roslaunch launch_file.launch arg_name:=value
```

range_world.launch (simplified)

```
<?xml version="1.0"?>
<launch>
  <arg name="use_sim_time" default="true"/>
  <arg name="world" default="gazebo_ros_range"/>
  <arg name="debug" default="false"/>
  <arg name="physics" default="ode"/>

  <group if="$(arg use_sim_time)">
    <param name="/use_sim_time" value="true" />
  </group>

  <include file="$(find gazebo_ros)
                                /launch/empty_world.launch">
    <arg name="world_name" value="$(find gazebo_plugins)/
                          test/test_worlds/$(arg world).world"/>
    <arg name="debug" value="$(arg debug)"/>
    <arg name="physics" value="$(arg physics)"/>
  </include>
</launch>
```

ROS Launch

包含其他Launch文件

- 用<include>标签包含其他launch文件来组织大型项目

```
<include file="file_path"/>
```

- 查找其他软件包的系统路径

```
$(find package_name)
```

- 替换包含文件里的参数

```
<arg name="arg_name"
value="value"/>
```

range_world.launch (simplified)

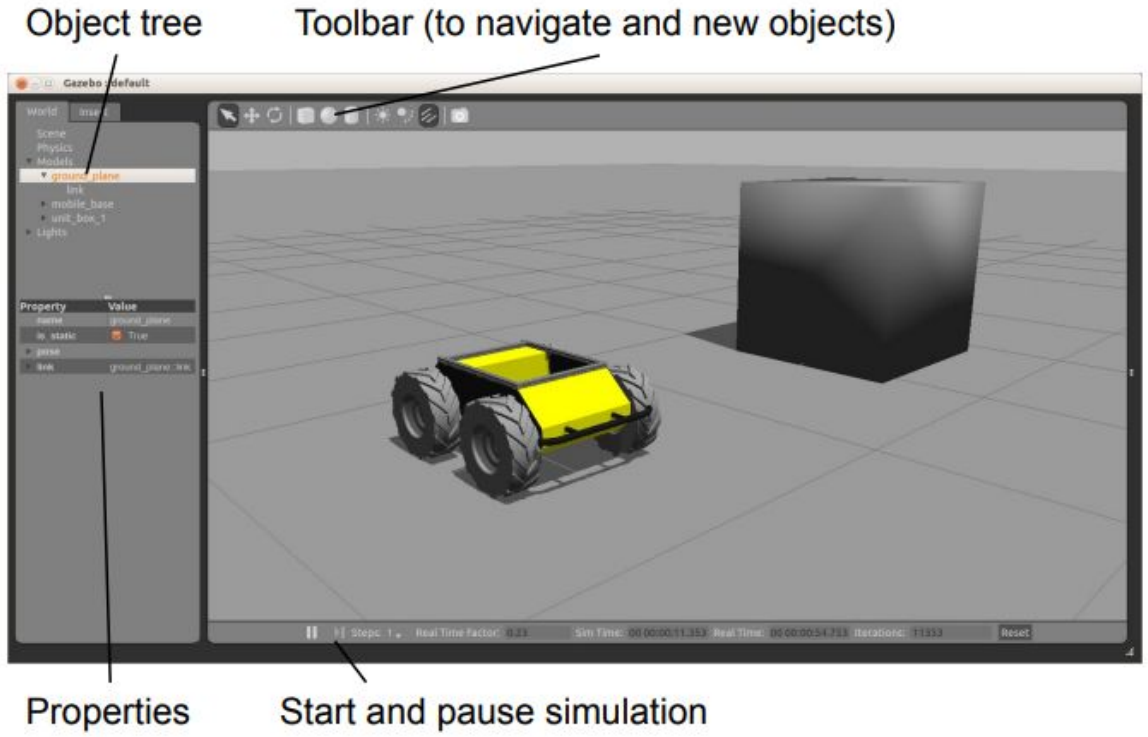
```
<?xml version="1.0"?>
<launch>
  <arg name="use_sim_time" default="true"/>
  <arg name="world" default="gazebo_ros_range"/>
  <arg name="debug" default="false"/>
  <arg name="physics" default="ode"/>

  <group if="$(arg use_sim_time)">
    <param name="/use_sim_time" value="true" />
  </group>

  <include file="$(find gazebo_ros)
                          /launch/empty_world.launch">
    <arg name="world_name" value="$(find gazebo_plugins)/
                    test/test_worlds/$(arg world).world"/>
    <arg name="debug" value="$(arg debug)"/>
    <arg name="physics" value="$(arg physics)"/>
  </include>
</launch>
```

Gazebo仿真

- 可模拟三维刚体动力学
- 模拟各种传感器，包括噪音
- 3D 可视化和用户交互
- 包含许多机器人和环境的数据库
- 提供了一个ROS接口
- 可通过插件拓展功能



更多资料

- **ROS WIKI**

http://wiki.ros.org/

- **安装**

http://wiki.ros.org/ROS/Installation

- **教程**

http://wiki.ros.org/ROS/Tutorials

- **可用软件包**

http://www.ros.org/browse/

- **ROS备忘单**

https://www.clearpathrobotics.com/ros-robot-operating-system-cheat-sheet/
https://kapeli.com/cheat_sheets/ROS.docset/Contents/Resources/Documents/index

- **ROS Best Practices**

https://github.com/leggedrobotics/ros_best_practices/wiki

- **ROS软件包示例**

https://github.com/leggedrobotics/ros_best_practices/tree/master/ros_package_template