



R.O.S

Robotic Operating System

1st Edition

Mục lục

ROS::BASIC.....	3
Điều kiện cài tiên quyết.....	3
Hệ thống quản lý file của ROS.....	3
Biểu đồ cấp bậc tính toán của ROS	4
msg/srv files	6
Tạo custom messages.....	6
Tạo thông báo tùy chỉnh	9
Launch files.....	11
Tạo Launch file.....	11

1

ROS::BASIC

Điều kiện cài tiên quyết

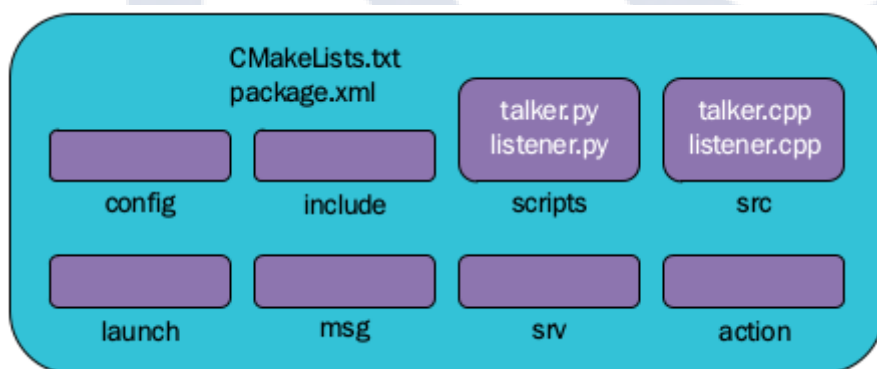


- [Cài Ubuntu 16.04](#)
- [Cài ROS trên Ubuntu](#)



Hệ thống quản lý file của ROS

- ROS Packages:



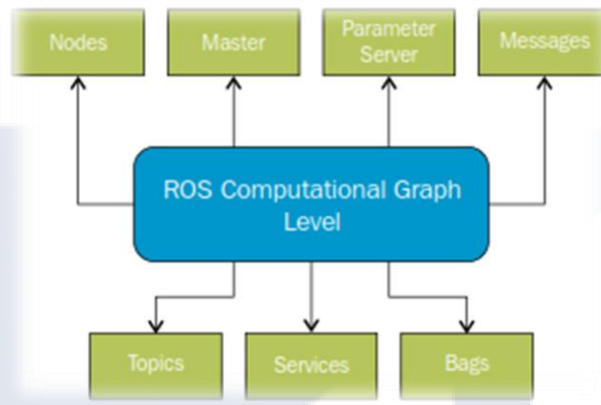
Hình 1 Nội dung của một package ROS

Gồm những folder sau: **config**, **include**, **scripts**, **src**, **launch**, **msg**, **srv**, **action** và những file: **package.xml**, **CMakeLists.txt**.

- **config**: các file cấu hình của package
- **include**: các header và các thư viện (library)
- **scripts**: các file python (.py)

- **src**: các file C++ (.cpp)
- **launch**: các launch file để chạy 1 hoặc nhiều node
- **msg**: các file message tự tạo (.msg)
- **srv**: các file service (.srv)
- **action**: các file action (*actionlib)
- **package.xml**: file chứa các khai xuất của package (package này có những gì và dùng những gì)
- **CMakeLists.txt**: CMake file của package (để hướng dẫn máy tính tạo các folder, executables, dependencies, target libraries)

Biểu đồ cấp bậc tính toán của ROS



Hình 2 Biểu đồ cấp bậc tính toán

- **ROS Node:**
 - Là các process mà thi hành các tính toán trong dùng thư viện của **ROS** (C++/Python).
- **ROS Message:**
 - **ROS Node** có thể **publish** các dữ liệu.
 - Loại dữ liệu được gửi sẽ được khai ở trong **ROS Message**.
 - Các **message** thường có 2 phần: **field** và **constant**.
 - **Field** gồm **field type** (kiểu dữ liệu: **string, int8, float, time...**) và **field name** (tên trường).
 - **Constant** gồm các giá trị ko đổi.
 - **ROS Message** còn có các **header** dùng để cho người lập trình biết được mốc thời điểm gửi **message**, người gửi hay thứ tự gửi của **message**.
- **ROS Topic:**
 - Là một kiểu trao đổi thông tin giữa hai node theo kiểu một chiều.
 - ROS Topic thường dùng **TCP/IP (TCPROS)**, nhưng cũng có thể dùng **UDP/IP (UDPROS)** cho các trao đổi mà cần tốc độ chuyển giao cao,

thời điểm thông tin được chuyển giao không cần quá chính xác.

UDPROS thường đc dùng cho teleoperation (dung lượng và băng thông thấp).

- **ROS Service:**

- Một loại thông tin hai chiều (kiểu **request** ↔ **respond**) giữa các node. **Node client** sẽ gửi **request** và đợi **respond** của **node server** còn lại. Giao tiếp giữa hai node vẫn dùng **ROS Message**.
- Thường được dùng cho **camera** hoặc **cảm biến 3D** (dung lượng và băng truyền cao)

- **ROS Bags:**

- Dùng để giữ lại dữ liệu **ROS Message** từ các **topic** và các **service**. Các **Bag** có thể ghi lại, chơi lại và sửa lại các dữ liệu. Thường dùng cho việc data logging.

- **ROS Master:**

- Giống một hệ thống **DNS**, khi bất kì node nào khởi động, nó sẽ tìm đến **ROS Master** và đăng kí tên của mình.
- Khi node thay đổi, **ROS Master** sẽ có lệnh call-back đến node đó và ghi lại sự thay đổi.
- Khi node **publish** một **topic**, dữ liệu của topic được đẩy đến **ROS Master** và **ROS Master** sẽ kiểm tra xem còn node nào khác **subscribe** đến **topic** đó không. Nếu có, **ROS Master** sẽ chia sẻ dữ liệu ở topic đó cho node đã **subscribe**.

- **ROS Parameter:**

- Công cụ dùng để ghi nhớ dữ liệu mà các node có thể truy cập vào, đọc và chỉnh sửa.
- Thường được dùng cho các dữ liệu như các tham số **P, I, D** trong một vòng điều khiển **PID**.

2

msg/srv files

Trong phần này, chúng ta sẽ:

- Tạo các custom messages.
- Tạo các service definitions.

Các custom messages được lưu trong file **.msg** và service definitions được lưu trong file **.srv**. Những định nghĩa này thông báo cho ROS về loại dữ liệu và tên của dữ liệu được truyền từ **node ROS**. Khi một thông báo tùy chỉnh được thêm vào, ROS sẽ chuyển đổi các định nghĩa thành code **C++** tương đương.

Tạo custom messages

Chúng ta có thể bắt đầu với các custom message. Custom message phải được viết trong file **.msg** và phải được lưu trong thư mục **msg** trong gói (package).

Chúng ta sẽ tạo một file message có tên là **demo_msg.msg** với định nghĩa sau:

```
string greeting
int32 number
```

Đến bây giờ, chúng ta chỉ làm việc với các message chuẩn (std_msgs). Bây giờ, chúng ta đã tạo message riêng của mình và có thể xem cách sử dụng chúng trong code.

Bước đầu tiên là chỉnh sửa file **package.xml** của gói hiện tại và bỏ ghi chú (##):

```
<build_depend>message_generation</build_depend> int32 number
```

Và:

```
<exec_depend>message_runtime</exec_depend>.
```

Chỉnh sửa **CMakeLists.txt** hiện tại và thêm dòng **message_generation** như sau:

```
find_package(catkin REQUIRED COMPONENTS
  roscpp
  rospy std_msgs
  actionlib
  actionlib_msgs
  message_generation
)
```

Bỏ ghi chú dòng sau và thêm file custom message:

```
add_message_files(
  FILES
  demo_msg.msg
)
## Generate added messages and services with any
dependencies listed here
generate_messages(
  DEPENDENCIES
  Std_msgs
  actionlib_msgs
)
```

Sau các bước này, chúng ta có thể biên dịch và xây dựng gói:

```
$ cd ~/catkin_ws/  
$ catkin_make
```

Để kiểm tra xem message có được xây dựng đúng cách hay không, chúng ta có thể sử dụng lệnh **rosmmsg**:

```
$ rosmmsg show mastering_ros_demo_pkg/demo_msg
```

Nếu nội dung được hiển thị bởi lệnh và định nghĩa là như nhau, quy trình là chính xác.

Nếu chúng ta muốn kiểm tra custom message, chúng ta có thể xây dựng một **publisher** và một **subscriber** bằng cách sử dụng loại thông báo tùy chỉnh có tên là **demo_msg_publisher.cpp** và **demo_msg_subscriber.cpp**. Điều hướng đến thư mục **mastering_ros_demo_pkg/src** để xem những code này.

Chúng ta có thể kiểm tra message bằng cách thêm các dòng code sau trong **CMakeLists.txt**:

```
add_executable(demo_msg_publishersrc/demo_msg_publisher.cpp)  
add_executable(demo_msg_subscribersrc/demo_msg_subscriber.cpp)  
  
add_dependencies(demo_msg_publisher  
mastering_ros_demo_pkg_generate_messages_cpp)  
add_dependencies(demo_msg_subscriber  
mastering_ros_demo_pkg_generate_messages_cpp)  
  
target_link_libraries(demo_msg_publisher ${catkin_LIBRARIES})  
target_link_libraries(demo_msg_subscriber ${catkin_LIBRARIES})
```

Xây dựng gói bằng cách sử dụng **catkin_make** và kiểm tra node bằng các lệnh sau.

```
$ roscore  
$ rosrn mastering_ros_demo_pkg demo_msg_publisher
```

Node **publisher** xuất bản một chuỗi cùng với một số nguyên và node **subscriber** đăng ký chủ đề và in các giá trị. Đầu ra và đồ thị được hiển thị như sau.


```

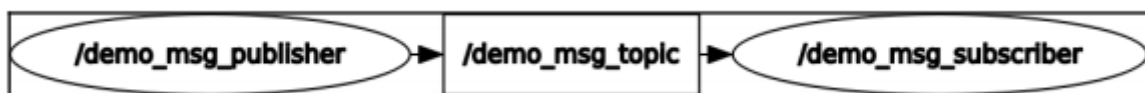
jcat@robot:~$ roslaunch mastering_ros_demo_pkg demo_msg_publisher
INFO [1500276387.166778705]: 0
INFO [1500276387.166861438]: hello world
INFO [1500276387.267694471]: 1
INFO [1500276387.267855187]: hello world
INFO [1500276387.368803935]: 2
INFO [1500276387.368898128]: hello world
INFO [1500276387.466853659]: 3
INFO [1500276387.466933039]: hello world

jcat@robot:~$ roslaunch mastering_ros_demo_pkg demo_msg_subscriber
INFO [1500276387.467496520]: Received greeting [hello world ]
INFO [1500276387.467579254]: Received [3]
INFO [1500276387.567331442]: Received greeting [hello world ]
INFO [1500276387.567382312]: Received [4]
INFO [1500276387.668345874]: Received greeting [hello world ]
INFO [1500276387.668564167]: Received [5]
INFO [1500276387.768672445]: Received greeting [hello world ]
INFO [1500276387.768753221]: Received [6]

```

Hình 3 Màn hình console khi chạy 2 node publish và subscribe song song

Chủ đề mà các node giao tiếp được gọi là `/demo_msg_topic`. Đây là biểu đồ của hai node:



Hình 4 Cách thức giao tiếp của 2 node với nhau

Tạo thông báo tùy chỉnh

Tiếp theo, chúng ta có thể thêm các file `srv` vào gói. Tạo một thư mục mới có tên là `srv` trong thư mục gói hiện tại và thêm một file `srv` có tên là `demo_srv.srv`. Định nghĩa của file này như sau:

```

string in
---
string out

```

Ở đây, cả **Request** và **Response** là các chuỗi.

Trong bước tiếp theo, chúng ta cần bỏ ghi chú của các dòng sau trong **package.xml**:

```

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

```

Bổ sung thêm `message_runtime` vào **catkin_package()** ở trong file **CMakeLists.txt**:

```
catkin_package(CATKIN_DEPENDS
    roscpp
    rospy
    std_msgs
    actionlib
    actionlib_
    msgs
    message_runtime
)
```

Chúng ta cần phải làm theo quy trình tương tự trong việc tạo ra các service như chúng ta đã làm cho thông báo **ROS (ROS message)**. Ngoài ra, chúng ta cần bổ thêm các phần ghi chú, như được hiển thị ở đây:

```
## Generate services in the 'srv' folder
add_service_files(
  FILES
  demo_srv.srv
)
```

Sau khi thực hiện những thay đổi này, chúng ta có thể xây dựng gói bằng cách sử dụng **catkin_make**. Chúng ta có thể kiểm chứng quy trình khi sử dụng lệnh sau:

```
$ rossrv show mastering_ros_demo_pkg/demo_srv $ rosrn
```

Nếu chúng ta thấy cùng một nội dung như đã xác định trong file **demo_srv.srv** thì có nghĩa rằng tất cả service đang hoạt động bình thường.

3

Launch files

Các launch file (tệp khởi chạy) trong **ROS** là một tính năng rất hữu ích để khởi chạy nhiều hơn một **node**. Trong các ví dụ trước, chúng ta đã thấy tối đa hai **node ROS**, nhưng hãy tưởng tượng một trường hợp mà phải khởi chạy 10 hoặc 20 node cho một robot. Sẽ rất khó nếu chúng ta chạy từng node một trong terminal. Thay vào đó, chúng ta có thể viết tất cả các node bên trong một file **XML** được gọi là các **launch file**. Chúng ta có thể phân tích file này và khởi chạy các node bằng cách sử dụng lệnh gọi là **\$roslaunch**.

Lệnh **\$roslaunch** sẽ tự khởi động **ROS Master** và máy chủ tham số. Về bản chất, sẽ không cần phải khởi động lệnh **\$roscore** và các node tự tạo. Nếu chúng ta khởi chạy **launch file**, tất cả các thao tác sẽ được thực hiện trong một lệnh duy nhất.

Tạo Launch file

Hãy bắt đầu tạo các launch files. Chuyển sang thư mục gói (**mastering_ros_demo_pkg**) và tạo một launch file mới có tên là **demo_topic.launch** để khởi chạy hai node ROS đang xuất bản (publish) và đăng ký (subscribe) một số nguyên. Chúng ta giữ các **launch file** trong thư mục **launch**, nằm bên trong gói:

```
$ roscd mastering_ros_demo_pkg
$ mkdir launch
$ cd launch
$ gedit demo_topic.launch
$ rosrn mastering_ros_demo_pkg demo_msg_subscriber
```

Paste nội dung sau vào file `demo_topic.launch`:

```
<launch>
  <node name="publisher_node" pkg="mastering_ros_demo_pkg"
type="demo_topic_publisher" output="screen"/>

  <node name="subscriber_node" pkg="mastering_ros_demo_pkg"
type="demo_topic_subscriber" output="screen"/>
</launch>
```

Hãy thảo luận những gì có trong code. Các thẻ `<launch></launch>` là phần gốc trong **launch file**. Tất cả các định nghĩa sẽ nằm trong các thẻ này.

Thẻ `<node>` chỉ định node mong muốn để khởi chạy:

```
<node name="publisher_node" pkg="mastering_ros_demo_pkg"
type="demo_topic_publisher" output="screen"/>
```

Thẻ **name** bên trong `<node>` cho biết tên của node, **pkg** là tên của gói và **type** là tên của **executable** mà chúng ta sẽ khởi chạy.

Sau khi tạo launch file `demo_topic.launch`, chúng ta có thể khởi chạy nó bằng cách sử dụng lệnh sau:

```
$ roslaunch mastering_ros_demo_pkg demo_topic.launch
```

Đây là đầu ra chúng ta nhận được nếu khởi chạy thành công:

```
started roslaunch server http://robot:34091/
SUMMARY
*****
PARAMETERS
 * /rostdistro: kinetic
 * /rosversion: 1.12.7
NODES
 /
  publisher_node (mastering_ros_demo_pkg/demo_topic_publisher)
  subscriber_node (mastering_ros_demo_pkg/demo_topic_subscriber)
auto-starting new master
process[master]: started with pid [10348]
ROS_MASTER_URI=http://localhost:11311
```

Hình 5 Màn hình terminal khi có lệnh `roslaunch`

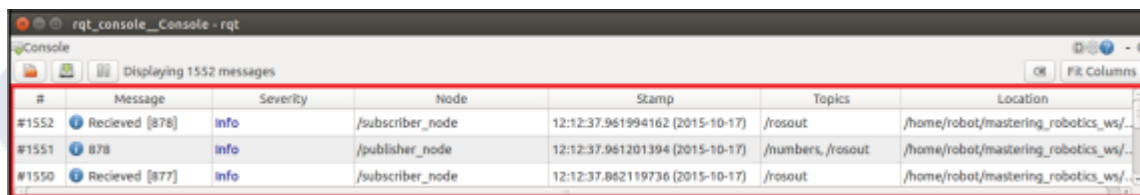
Chúng ta có thể kiểm tra danh sách các node bằng cách sử dụng:

```
$ rosnode list
```

Chúng ta cũng có thể xem các thông điệp tường trình và **debug** các **node** bằng cách sử dụng một công cụ **GUI** có tên là **rqt_console**:

```
$ rqt_console
```

Chúng ta có thể thấy các bản ghi được tạo ra bởi hai node trong công cụ này, như được hiển thị ở đây:



The screenshot shows the rqt_console window with a table of log messages. The table has columns for #, Message, Severity, Node, Stamp, Topics, and Location. Three messages are visible, all with 'Info' severity.

#	Message	Severity	Node	Stamp	Topics	Location
#1552	Recieved [878]	Info	/subscriber_node	12:12:37.961994162 (2015-10-17)	/rosout	/home/robot/mastering_robotics_ws/...
#1551	878	Info	/publisher_node	12:12:37.961201394 (2015-10-17)	/numbers, /rosout	/home/robot/mastering_robotics_ws/...
#1550	Recieved [877]	Info	/subscriber_node	12:12:37.862119736 (2015-10-17)	/rosout	/home/robot/mastering_robotics_ws/...