# ROS 로봇프로그래밍

-ROS1 완독하기 챌린지-

김예나

# Ch7. ROS 기본 프로그래밍

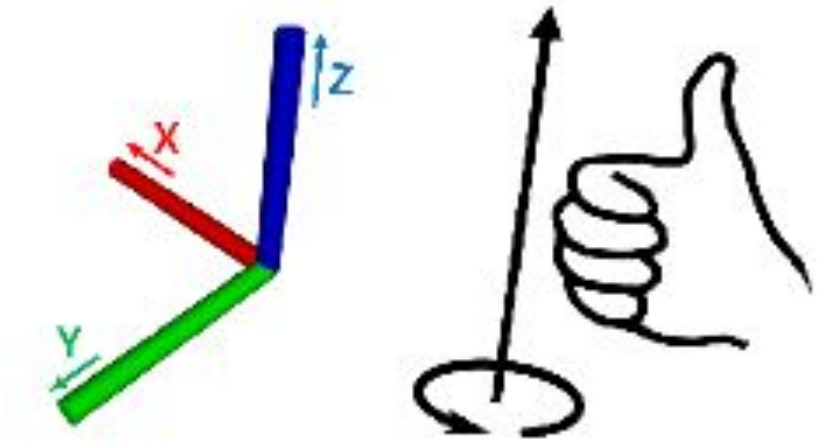
# 7.1 ROS 프로그래밍 전에 알아둬야 할 사항

1) 표준단위 : SI

| Quantity | Unit |
|---|---|
| length | meter |
| mass | kilogram |
| time | second |
| current | ampere |

| Quantity | Unit |
|---|---|
| angle | radian |
| frequency | hertz |
| force | newton |
| power | watt |
| voltage | volt |
| temperature | celsius |

2) 좌표 표현 방식 : 오른손, 시계반대방향




ROS커뮤니티에서 유저들이 제안하는 규칙, 새로운 기능, 관리 방법 등을 담은 표준문서(REP)에 명시

⇒ http://www.ros.org/reps/rep-0103.html

# 7.1 ROS 프로그래밍 전에 알아둬야 할 사항

## 3) 프로그래밍 규칙

언어별로 위키에 상세히 설명되어 있음

⇒ http://wiki.ros.org/CppStyleGuide

⇒ http://wiki.ros.org/PyStyleGuide

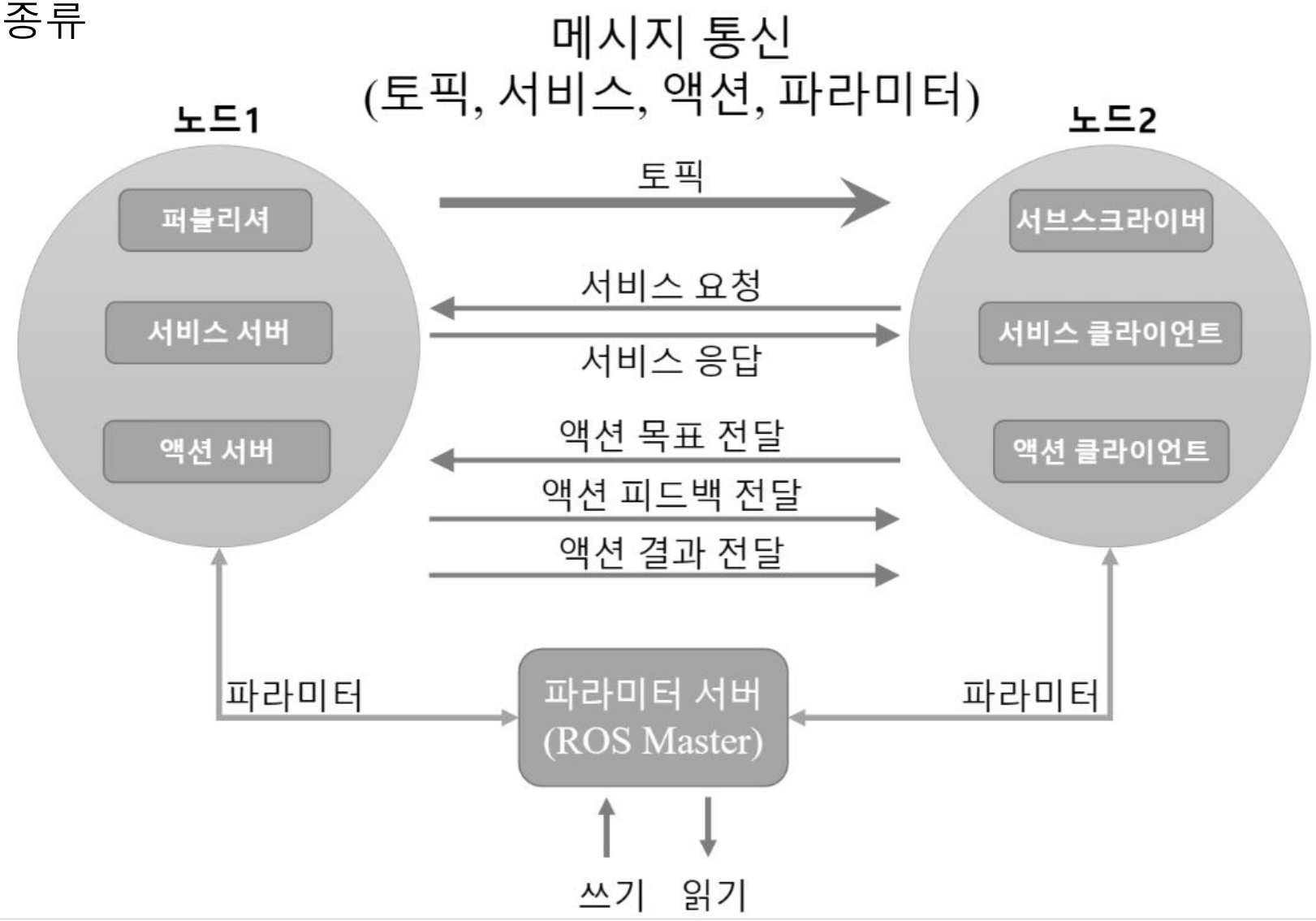
⇒ http://wiki.ros.org/ROS/Patterns

/Convetions#Naming_ROS_Resources

| 대상 | 명명 규칙 | 예제 |
|---|---|---|
| 패키지 | under_scored | first_ros_package |
| 토픽, 서비스 | under_scored | raw_image |
| 파일 | under_scored | turtlebot3_fake.cpp |
| 네임스페이스 | under_scored | ros_awesome_package |
| 변수 | under_scored | string table_name; |
| 타입 | CamelCased | typedef int32_t PropertiesNumber; |
| 클래스 | CamelCased | class UrlTable |
| 구조체 | CamelCased | struct UrlTableProperties |
| 열거형 | CamelCased | enum ChoiceNumber |
| 함수 | CamelCased | addTableEntry(); |
| 메소드 | CamelCased | void setNumEntries(int32_t num_entries) |
| 상수 | ALL_CAPITALS | const uint8_t DAYS_IN_A_WEEK = 7; |
| 매크로 | ALL_CAPITALS | #define PI_ROUNDED 3.0 |

# 7.1 ROS 프로그래밍 전에 알아둬야 할 사항

4) 메시지 통신 종류



# 7.2 토픽: 퍼블리셔와 서브스크라이버 노드 작성 및 실행

1) 패키지 생성

```
$cd ~/catkin_ws/src
$catkin_create_pkg  ros_tutorials_topic  message_generation  std_msgs  roscpp
```

// 패키지 생성 명령어 패키지 이름 의존성패키지 목록

```
$cd ros_tutorials_topic
$ls
```

⇒ include(헤더 파일 폴더)

src(소스 코드 폴더)

CMakeLists.txt(빌드 설정 파일)

package.xml(패키지 설정 파일)

# 7.2 토픽: 퍼블리셔와 서브스크라이버 노드 작성 및 실행

2) 패키지 설정 파일(package.xml)수정

```
$gedit package.xml
```

```
<?xml version="1.0"?>
<package format="2">
  <name>ros_tutorials_topic</name>
  <version>0.1.0</version>
  <description>ROS turtorial package to learn the topic</description>
  <license>Apache 2.0</license>
  <author email="pyo@robotis.com">Yoonseok Pyo</author>
  <maintainer email="pyo@robotis.com">Yoonseok Pyo</maintainer>
  <url type="website">http://www.robotis.com</url>
  <url type="repository">https://github.com/ROBOTIS-GIT/ros_tutorials.git</url>
  <url type="bugtracker">https://github.com/ROBOTIS-GIT/ros_tutorials/issues</url >
  <buildtool_depend>catkin</buildtool_depend>
  <depend>roscpp</depend>
  <depend>std_msgs</depend>
  <depend>message_generation</depend>
  <export></export >
</package>
```

# 7.2 토픽: 퍼블리셔와 서브스크라이버 노드 작성 및 실행

## 3) 빌드 설정 파일(CMakeLists.txt)수정

```
$gedit CMakeLists.txt
```

```
cmake_minimum_required(VERSION 2.8.3)
project(ros_tutorials_topic)

## 캐킨 빌드를 할 때 요구되는 구성요소 패키지이다.
## 의존성 패키지로 message_generation, std_msgs, roscpp이며 이 패키지들이 존재하지 않으면 빌드 도중에 에러가 난다.
find_package(catkin REQUIRED COMPONENTS message_generation std_msgs roscpp)

## 메시지 선언: MsgTutorial.msg
add_message_files(FILES MsgTutorial.msg)

## 의존하는 메시지를 설정하는 옵션이다.
## std_msgs가 설치되어 있지 않다면 빌드 도중에 에러가 난다.
generate_messages(DEPENDENCIES std_msgs)

## 캐킨 패키지 옵션으로 라이브러리, 캐킨 빌드 의존성, 시스템 의존 패키지를 기술한다.
catkin_package(
  LIBRARIES ros_tutorials_topic
  CATKIN_DEPENDS std_msgs roscpp
)
```

# 7.2 토픽: 퍼블리셔와 서브스크라이버 노드 작성 및 실행

3) 빌드 설정 파일(CMakeLists.txt)수정

```
## 인클루드 디렉터리를 설정한다.
include_directories(${catkin_INCLUDE_DIRS})

## topic_publisher 노드에 대한 빌드 옵션이다.
## 실행 파일, 타깃 링크 라이브러리, 추가 의존성 등을 설정한다.
add_executable(topic_publisher src/topic_publisher.cpp)
add_dependencies(topic_publisher ${${PROJECT_NAME}_EXPORTED_TARGETS}
${catkin_EXPORTED_TARGETS})
target_link_libraries(topic_publisher ${catkin_LIBRARIES})

## topic_subscriber 노드에 대한 빌드 옵션이다.
add_executable(topic_subscriber src/topic_subscriber.cpp)
add_dependencies(topic_subscriber ${${PROJECT_NAME}_EXPORTED_TARGETS}
${catkin_EXPORTED_TARGETS})
target_link_libraries(topic_subscriber ${catkin_LIBRARIES})
```

# 7.2 토픽: 퍼블리셔와 서브스크라이버 노드 작성 및 실행

4) 메시지 파일 작성

```
## 메시지 선언: MsgTutorial.msg
add_message_files(FILES MsgTutorial.msg)
```

CMakeLists.txt파일 수정할 때, 이번 노드에서 사용할 메시지인 MsgTutorial.msg를 빌드할 때 포함하라는 옵션

```
$roscd ros_tutorials_topic      → 패키지 폴더로 이동
$mkdir msg                          → 패키지에 msg 메시지 폴더 생성
$cd msg                         → 생성한 msg 폴더로 이동
$gedit MsgTutorial.msg              → MsgTutorial.msg 파일 신규 작성 및 내용 수정
```

```
time stamp
int32 data
```

# 7.2 토픽: 퍼블리셔와 서브스크라이버 노드 작성 및 실행

## 5) 퍼블리셔 노드 작성

```
## topic_publisher 노드에 대한 빌드 옵션이다.
## 실행 파일, 타깃 링크 라이브러리, 추가 의존성 등을 설정한다.
add_executable(topic_publisher src/topic_publisher.cpp)
```

CMakeLists.txt파일 수정할 때, 다음과 같은 실행 파일 생성하라는 옵션 설정함

```
$roscd ros_tutorials_topic/src        → 패키지 소스 폴더 src로 이동
$gedit topic_publisher.cpp            → 소스 파일 신규 작성 및 내용 수정
```

```
#include "ros/ros.h"                                  // ROS 기본 헤더파일
#include "ros_tutorials_topic/MsgTutorial.h"// MsgTutorial 메시지 파일 헤더(빌드 후 자동 생성됨)

int main(int argc, char **argv)                       // 노드 메인 함수
{
 ros::init(argc, argv, "topic_publisher");            // 노드명 초기화
 ros::NodeHandle nh;                                  // ROS 시스템과 통신을 위한 노드 핸들 선언
```

# 7.2 토픽: 퍼블리셔와 서브스크라이버 노드 작성 및 실행

5) 퍼블리셔 노드 작성

```
// 퍼블리셔 선언, ros_tutorials_topic 패키지의 MsgTutorial 메시지 파일을 이용한
// 퍼블리셔 ros_tutorial_pub 를 작성한다. 토픽명은 "ros_tutorial_msg" 이며,
// 퍼블리셔 큐(queue) 사이즈를 100개로 설정한다는 것이다
ros::Publisher ros_tutorial_pub = nh.advertise<ros_tutorials_topic::MsgTutorial>("ros_tutorial_msg", 100);

// 루프 주기를 설정한다. "10" 이라는 것은 10Hz를 말하는 것으로 0.1초 간격으로 반복된다
ros::Rate loop_rate(10);

// MsgTutorial 메시지 파일 형식으로 msg 라는 메시지를 선언
ros_tutorials_topic::MsgTutorial msg;

// 메시지에 사용될 변수 선언
int count = 0;
```

# 7.2 토픽: 퍼블리셔와 서브스크라이버 노드 작성 및 실행

5) 퍼블리셔 노드 작성

```
while (ros::ok())
{
  msg.stamp = ros::Time::now();          // 현재 시간을 msg의 하위 stamp 메시지에 담는다
  msg.data  = count;                     // count라는 변수 값을 msg의 하위 data 메시지에 담는다

  ROS_INFO("send msg = %d", msg.stamp.sec);          // stamp.sec 메시지를 표시한다
  ROS_INFO("send msg = %d", msg.stamp.nsec);         // stamp.nsec 메시지를 표시한다
  ROS_INFO("send msg = %d", msg.data);               // data 메시지를 표시한다

  ros_tutorial_pub.publish(msg);                     // 메시지를 발행한다

  loop_rate.sleep();                                 // 위에서 정한 루프 주기에 따라 슬립에 들어간다

  ++count;                                           // count 변수 1씩 증가
}

return 0;
}
```

# 7.2 토픽: 퍼블리셔와 서브스크라이버 노드 작성 및 실행

## 6) 서브스크라이버 노드 작성

```
## topic_subscriber 노드에 대한 빌드 옵션이다.
add_executable(topic_subscriber src/topic_subscriber.cpp)
```

CMakeLists.txt파일 수정할 때, 다음과 같은 실행 파일 생성하라는 옵션 설정함

```
$roscd ros_tutorials_topic/src        → 패키지 소스 폴더 src로 이동
$gedit topic_subscriber.cpp             → 소스 파일 신규 작성 및 내용 수정
```

```
#include "ros/ros.h"                                    // ROS 기본 헤더파일
#include "ros_tutorials_topic/MsgTutorial.h"            // MsgTutorial 메시지 파일 헤더 (빌드 후 자동 생성됨)
// 메시지 콜백 함수로써, 밑에서 설정한 ros_tutorial_msg라는 이름의 토픽
// 메시지를 수신하였을 때 동작하는 함수이다
// 입력 메시지로는 ros_tutorials_topic 패키지의 MsgTutorial 메시지를 받도록 되어있다
void msgCallback(const ros_tutorials_topic::MsgTutorial::ConstPtr& msg)
{
  ROS_INFO("recieve msg = %d", msg->stamp.sec);         // stamp.sec 메시지를 표시한다
  ROS_INFO("recieve msg = %d", msg->stamp.nsec);        // stamp.nsec 메시지를 표시한다
  ROS_INFO("recieve msg = %d", msg->data);              // data 메시지를 표시한다
}
```

# 7.2 토픽: 퍼블리셔와 서브스크라이버 노드 작성 및 실행

6) 서브스크라이버 노드 작성

```
int main(int argc, char **argv)                    // 노드 메인 함수
{
  ros::init(argc, argv, "topic_subscriber");       // 노드명 초기화

  ros::NodeHandle nh;                              // ROS 시스템과 통신을 위한 노드 핸들 선언

  // 서브스크라이버 선언, ros_tutorials_topic 패키지의 MsgTutorial 메시지 파일을 이용한
  // 서브스크라이버 ros_tutorial_sub 를 작성한다. 토픽명은 "ros_tutorial_msg" 이며,
  // 서브스크라이버 큐(queue) 사이즈를 100개로 설정한다는 것이다
  ros::Subscriber ros_tutorial_sub = nh.subscribe("ros_tutorial_msg", 100, msgCallback);

  // 콜백함수 호출을 위한 함수로써, 메시지가 수신되기를 대기,
  // 수신되었을 경우 콜백함수를 실행한다
  ros::spin();

  return 0;
}
```

# 7.2 토픽: 퍼블리셔와 서브스크라이버 노드 작성 및 실행

7) 노드 빌드

```
$cd ~/catkin_ws     → catkin 폴더로 이동
$catkin_make        → catkin 빌드 실행
```

~/catkin_ws의
/build 폴더에 캐킨빌드에 사용된 설정 내용 저장,
/devel 폴더에 실행파일 및 메시지 헤더파일 저장

8) 퍼블리셔 실행

```
$roscore
$rosrun ros_tutorials_topic topic_publisher
```

topic_publisher에서 퍼블리시하는 토픽 받아보기!
$rostopic list
→ 결과: /ros_tutorial_msg /rosout /rosout_agg
$rostopic echo /ros_tutorial_msg

9) 서브스크라이버 실행

```
$rosrun ros_tutorials_topic
topic_subscriber
```

10) rqt_graph

# 7.2 토픽: 퍼블리셔와 서브스크라이버 노드 작성 및 실행

**참고! git clone 사용하여 바로 적용하는 방법**

```
$cd ~/catkin_ws/src
$git clone https://github.com/ROBOTIS-GIT/ros_tutorials.git
$cd ~/cakin_ws
$catkin_make
```

```
$rosrun ros_tutorials_topic topic_publisher
```

```
$rosrun ros_tutorials_topic topic_subscriber
```

# 7.3 서비스: 서비스 서버와 클라이언트 노드 작성 및 실행

1) 패키지 생성

```
$cd ~/catkin_ws/src
$catkin_create_pkg ros_tutorials_service message_generation std_msgs roscpp
```

2) 패키지 설정 파일(package.xml)수정

```
$gedit package.xml
```

3) 빌드 설정 파일(CMakeLists.txt)수정

```
$gedit CMakeLists.txt
```

# 7.3 서비스: 서비스 서버와 클라이언트 노드 작성 및 실행

## 4) 서비스 파일 작성

```
$roscd ros_tutorials_service        → 패키지 폴더로 이동
$mkdir srv                          → 패키지에 srv 메시지 폴더 생성
$cd srv                             → 생성한 srv 폴더로 이동
$gedit SrvTutorial.srv              → SrvTutorial.srv 파일 신규 작성 및 내용 수정
```

```
int64 a
int64 b
---
int64 result
```

# 7.3 서비스: 서비스 서버와 클라이언트 노드 작성 및 실행

## 5) 서비스 서버 노드 작성

```
$roscd ros_tutorials_service/src
$gedit service_server.cpp
```

```
#include "ros/ros.h"                                    // ROS 기본 헤더 파일
#include "ros_tutorials_service/SrvTutorial.h"  // SrvTutorial 서비스 파일 헤더 (빌드후 자동 생성됨)

// 서비스 요청이 있을 경우, 아래의 처리를 수행한다
// 서비스 요청은 req, 서비스 응답은 res로 설정하였다
bool calculation(ros_tutorials_service::SrvTutorial::Request &req,
                 ros_tutorials_service::SrvTutorial::Response &res)
{
  // 서비스 요청시 받은 a와 b 값을 더하여 서비스 응답 값에 저장한다
  res.result = req.a + req.b;

  // 서비스 요청에 사용된 a, b 값의 표시 및 서비스 응답에 해당되는 result 값을 출력한다
  ROS_INFO("request: x=%ld, y=%ld", (long int)req.a, (long int)req.b);
  ROS_INFO("sending back response: %ld", (long int)res.result);

  return true;
}
```

# 7.3 서비스: 서비스 서버와 클라이언트 노드 작성 및 실행

## 5) 서비스 서버 노드 작성

```
int main(int argc, char **argv)                    // 노드 메인 함수
{
  ros::init(argc, argv, "service_server");         // 노드명 초기화
  ros::NodeHandle nh;                              // 노드 핸들 선언

  // 서비스 서버 선언, ros_tutorials_service 패키지의 SrvTutorial 서비스 파일을 이용한
  // 서비스 서버 ros_tutorials_service_server를 선언한다
  // 서비스명은 ros_tutorial_srv이며 서비스 요청이 있을 때,
  // calculation라는 함수를 실행하라는 설정이다
  ros::ServiceServer ros_tutorials_service_server = nh.advertiseService("ros_tutorial_srv", calculation);

  ROS_INFO("ready srv server!");

  ros::spin();   // 서비스 요청을 대기한다

  return 0;
}
```

# 7.3 서비스: 서비스 서버와 클라이언트 노드 작성 및 실행

## 6) 서비스 클라이언트 노드 작성

```
$roscd ros_tutorials_service/src
$gedit service_client.cpp
```

```
#include "ros/ros.h"                                    // ROS 기본 헤더 파일
#include "ros_tutorials_service/SrvTutorial.h" // SrvTutorial 서비스 파일 헤더 (빌드후 자동 생성됨)
#include <cstdlib>                                      // atoll 함수 사용을 위한 라이브러리
int main(int argc, char **argv)                 // 노드 메인 함수
{
 ros::init(argc, argv, "service_client");       // 노드명 초기화

 if (argc != 3)                                 // 입력값 오류 처리
 {
  ROS_INFO("cmd : rosrun ros_tutorials_service service_client arg0 arg1");
  ROS_INFO("arg0: double number, arg1: double number");
  return 1;
 }

 ros::NodeHandle nh;                            // ROS 시스템과 통신을 위한 노드 핸들 선언

 // 서비스 클라이언트 선언, ros_tutorials_service 패키지의 SrvTutorial 서비스 파일을 이용한
 // 서비스 클라이언트 ros_tutorials_service_client를 선언한다
 // 서비스명은 "ros_tutorial_srv"이다
 ros::ServiceClient ros_tutorials_service_client =
nh.serviceClient<ros_tutorials_service::SrvTutorial>("ros_tutorial_srv");
```

# 7.3 서비스: 서비스 서버와 클라이언트 노드 작성 및 실행

## 6) 서비스 클라이언트 노드 작성

```
// srv라는 이름으로 SrvTutorial 서비스 파일을 이용하는 서비스를 선언한다
ros_tutorials_service::SrvTutorial srv;

// 서비스 요청 값으로 노드가 실행될 때 입력으로 사용된 매개변수를 각각의 a, b에 저장한다
srv.request.a = atoll(argv[1]);
srv.request.b = atoll(argv[2]);

// 서비스를 요청하고, 요청이 받아들여졌을 경우, 응답 값을 표시한다
if (ros_tutorials_service_client.call(srv))
{
  ROS_INFO("send srv, srv.Request.a and b: %ld, %ld", (long int)srv.request.a, (long int)srv.request.b);
  ROS_INFO("receive srv, srv.Response.result: %ld", (long int)srv.response.result);
}
else
{
  ROS_ERROR("Failed to call service ros_tutorial_srv");
  return 1;
}
return 0;
}
```

# 7.3 서비스: 서비스 서버와 클라이언트 노드 작성 및 실행

7) 노드 빌드

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

8) 서비스 서버 실행

```
$roscore
$rosrun ros_tutorials_service service_server
[INFO] [1495726541.268629564]: ready srv server! 서비스 요청 기다림
```

9) 서비스 클라이언트 실행

```
$rosrun ros_tutorials_service service_client 2 3
[INFO] [1495726541.277216401]: send srv, srv.Request.a and b: 2, 3
//입력매개변수
[INFO] [1495726541.277258018]: receive srv, srv.Response.result: 5   //응답값
```

# 7.3 서비스: 서비스 서버와 클라이언트 노드 작성 및 실행

## 10) rosservice call 명령어 사용법

rqt의 ServiceCaller도 이용 가능

```
$rosservice call /ros_tutorial_srv 10 2
result: 12
```

**※참고사항**
**하나의 노드는 복수의 퍼블리셔, 서브스크라이버, 서비스 서버, 서비스 클라이언트 역할도 할 수 있다!**

```
ros::NodeHandle nh;

ros::Publisher topic_publisher = nh.advertise<ros_tutorials::MsgTutorial>("ros_tutorial_msg", 100);
ros::Subscriber topic_subscriber = nh.subscribe("ros_tutorial_msg", 100, msgCallback);
ros::ServiceServer service_server = nh.advertiseService("ros_tutorial_srv", calculation);
ros::ServiceClient service_client = nh.serviceClient<ros_tutorials::SrvTutorial>("ros_tutorial_srv");
```

# 7.4 액션: 액션 서버와 클라이언트 노드 작성 및 실행

## 액션 파일 작성

```
$roscd ros_tutorials_action
$mkdir action
$cd action
$gedit Fibonacci.action
```

```
#goal definition
int32 order
---
#result definition
int32[] sequence
---
#feedback
int32[] sequence
```

# 7.5 파라미터 사용법

<서비스 실습에서 단순 a, b 덧셈을 파라미터를 활용한 사칙연산 수행으로 코드 수정>

1) 파라미터를 활용한 노드 작성

$roscd ros_tutorials_service/src → 패키지의 소스 코드 폴더인 src폴더로 이동

$gedit service_server.cpp → 소스 파일 내용 수정

```
#include "ros/ros.h"                          // ROS 기본 헤더파일
#include "ros_tutorials_service/SrvTutorial.h" // SrvTutorial 서비스 파일 헤더 (빌드 후 자동 생성됨)

#define PLUS            1   // 덧셈
#define MINUS           2   // 빼기
#define MULTIPLICATION  3   // 곱하기
#define DIVISION        4   // 나누기

int g_operator = PLUS;
```

# 7.5 파라미터 사용법

1) 파라미터를 활용한 노드 작성

```
// 서비스 요청이 있을 경우, 아래의 처리를 수행한다
// 서비스 요청은 req, 서비스 응답은 res로 설정하였다

bool calculation(ros_tutorials_service::SrvTutorial::Request &req,
                 ros_tutorials_service::SrvTutorial::Response &res)
{
 // 서비스 요청시 받은 a와 b 값을 파라미터 값에 따라 연산자를 달리한다.
 // 계산한 후 서비스 응답 값에 저장한다
 switch(g_operator)
 {
  case PLUS:
      res.result = req.a + req.b; break;
  case MINUS:
      res.result = req.a - req.b; break;
  case MULTIPLICATION:
      res.result = req.a * req.b; break;
```

# 7.5 파라미터 사용법

1) 파라미터를 활용한 노드 작성

```
case DIVISION:
        if(req.b == 0){
          res.result = 0; break;
        }
        else{
          res.result = req.a / req.b; break;
        }
    default:
        res.result = req.a + req.b; break;
  }

  // 서비스 요청에 사용된 a, b값의 표시 및 서비스 응답에 해당되는 result 값을 출력한다
  ROS_INFO("request: x=%ld, y=%ld", (long int)req.a, (long int)req.b);
  ROS_INFO("sending back response: [%ld]", (long int)res.result);

  return true;
}
```

# 7.5 파라미터 사용법

## 1) 파라미터를 활용한 노드 작성

```
int main(int argc, char **argv)                    // 노드 메인 함수
{
 ros::init(argc, argv, "service_server");          // 노드명 초기화

 ros::NodeHandle nh;                               // ROS 시스템과 통신을 위한 노드 핸들 선언

 nh.setParam("calculation_method", PLUS);          // 매개변수 초기설정

 // 서비스 서버 선언, ros_tutorials_service 패키지의 SrvTutorial 서비스 파일을 이용한
 // 서비스 서버 service_server를 작성한다. 서비스명은 "ros_tutorial_srv"이며,
 // 서비스 요청이 있을 때, calculation라는 함수를 실행하라는 설정이다.
 ros::ServiceServer ros_tutorial_service_server = nh.advertiseService("ros_tutorial_srv", calculation);

 ROS_INFO("ready srv server!");
```

nh.setParam(“calculation_method”,PLUS);
calculation_method 이름의 파라미터를 PLUS값으로 설정 ⇒ 1

# 7.5 파라미터 사용법

1) 파라미터를 활용한 노드 작성

파라미터 사용 가능 형태
: integers, floats, boolean, string, dictionaries, list 등

```
ros::Rate r(10);                                    // 10 hz

while (1)
{
  nh.getParam("calculation_method", g_operator);    // 연산자를 매개변수로부터 받은 값으로 변경한다
  ros::spinOnce();                                  // 콜백함수 처리루틴
  r.sleep();                                        // 루틴 반복을 위한 sleep 처리
}

return 0;
}
```

nh.getParam(“calculation_method”, g_operator);
0.1초마다 calculation_method 파라미터 불러와서 g_operator값으로 설정

# 7.5 파라미터 사용법

2) 노드 빌드 및 실행

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

```
$roscore
$rosrun ros_tutorials_service service_server
[INFO] [1495726541.268629564]: ready srv server! 서비스 요청 기다림
```

3) 매개변수 목록 보기

```
$rosparam list  ⇒ /calculation_method /rosdistro /rosverstion /run_id
```

# 7.5 파라미터 사용법

4) 파라미터 사용 예

**파라미터**
**⇒ 노드 외부로부터 노드의 흐름이나 설정, 처리 등을 바꿀 수 있음!! 매우 유용한 기능!!**

```
$ rosservice call /ros_tutorial_srv 10 5        → 사칙연산의 변수 a, b 입력
result: 15                                      → 디폴트 사칙연산인 덧셈 결괏값

$ rosparam set /calculation_method 2            → 뺄셈
$ rosservice call /ros_tutorial_srv 10 5
result: 5

$ rosparam set /calculation_method 3            → 곱셈
$ rosservice call /ros_tutorial_srv 10 5
result: 50

$ rosparam set /calculation_method 4            → 나눗셈
$ rosservice call /ros_tutorial_srv 10 5
result: 2
```

# 7.6 roslaunch 사용법

roslaunch란?

rosrun : 하나의 노드 실행
roslaunch : 하나 이상의 정해진 노드 실행

그 밖의 기능: 노드 실행 시 패키지의 파라미터나 노드 이름 변경, 노드 네임스페이스 설정, ROS_ROOT 및 ROS_PACKAGE_PATH 설정, 환경 변수 변경 등 옵션을 붙일 수 있음

'*.launch' 라는 파일 사용하여 실행 노드 설정 → XML 기반으로 태그별 옵션 제공

명령어: roslaunch [패키지명] [roslaunch 파일]

# 7.6 roslaunch 사용법

<토픽 실습에서 topic_publisher와 topic_subscriber 노드 이름 바꾸어 두 개씩 구동 및 통신>

1) roslaunch의 활용

```
$roscd ros_tutorials_topic        //해당 패키지 폴더에
$mkdir launch                     //launch라는 폴더 생성
$cd launch                        //그 안에
$gedit union.launch               //union.launch라는 새파일 생성
```

```
<launch>
  <node pkg="ros_tutorials_topic" type="topic_publisher" name="topic_publisher1"/>
  <node pkg="ros_tutorials_topic" type="topic_subscriber" name="topic_subscriber1"/>
  <node pkg="ros_tutorials_topic" type="topic_publisher" name="topic_publisher2"/>
  <node pkg="ros_tutorials_topic" type="topic_subscriber" name="topic_subscriber2"/>
</launch>
```

태그 옵션
pkg: 패키지 이름, type: 실제 실행할 노드명, name: 노드실행시 실행명

# 7.6 roslaunch 사용법

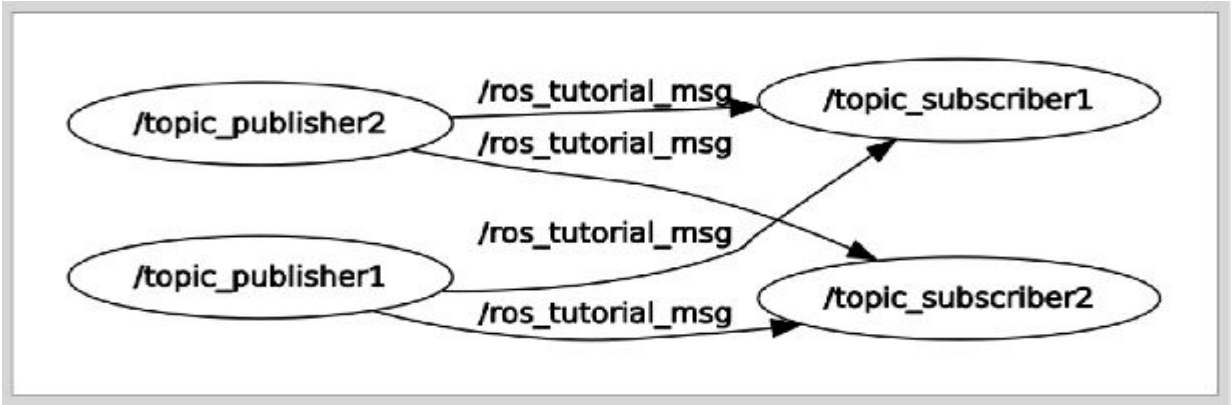
## 1) roslaunch의 활용

```
$roslaunch ros_tutorials_topic union.launch –screen          //union.launch 실행
```

**roslaunch 명령어로 여러 개의 노드가 실행될 때 실행되는 노드들의 출력(info, error 등)이 터미널 스크린에 표시되지 않아 디버깅 어려움**
**→ --screen 옵션 추가 : 해당 터미널에 실행되는 모든 노드들의 출력들이 터미널 스크린에 표시됨**

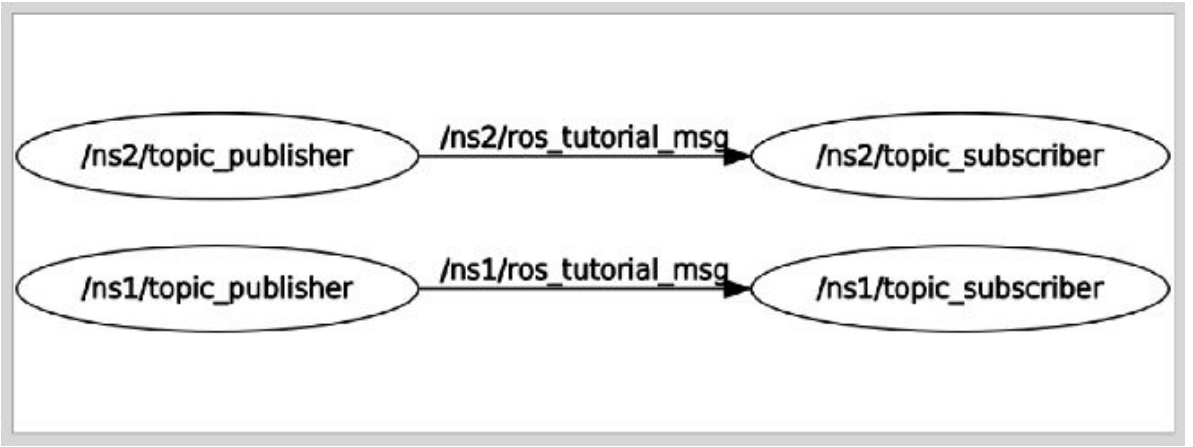
실행 결과 : 노드 이름만 바꾸고, 메세지 이름을 바꿔주지 않아 의도와 다르게 출력됨!

```
$ rosnode list
/topic_publisher1
/topic_publisher2
/topic_subscriber1
/topic_subscriber2
/rosout
```



# 7.6 roslaunch 사용법

변경 후 실행 결과




1) roslaunch의 활용

```
$roscd ros_tutorials_topic/launch
$gedit union.launch
//union.launch 수정
```

```
<launch>
  <group ns="ns1">
    <node pkg="ros_tutorials_topic" type="topic_publisher" name="topic_publisher"/>
    <node pkg="ros_tutorials_topic" type="topic_subscriber" name="topic_subscriber"/>
  </group>
  <group ns="ns2">
    <node pkg="ros_tutorials_topic" type="topic_publisher" name="topic_publisher"/>
    <node pkg="ros_tutorials_topic" type="topic_subscriber" name="topic_subscriber"/>
  </group>
</launch>
```

# 7.6 roslaunch 사용법

## 2) roslaunch 태그

```
<launch>
        <arg name="update_period" default="10" />
        <param name="timing" value="$(arg update_period)" />
</launch>
==========================================
$ roslaunch my_package my_package.launch update_period:=30
```

- <launch> : roslaunch구문의 시작과 끝
- <node> : 노드 실행에 대한 태그, 패키지, 노드명, 실행명을 변경할 수 있음
- <machine> : 노드를 실행하는 PC의 이름, address, ros-root, ros-package-path 등을 설정
- <include> : 다른 패키지나 같은 패키지에 속해 있는 다른 launch를 불러와 하나의 launch 파일처럼 실행할 수 있음
- <remap> : 노드 이름, 토픽 이름 등의 노드에서 사용중인 ROS 변수의 이름을 변경할 수 있음
- <env> : 경로, IP 등의 환경변수를 설정(거의 안씀)
- **<param>** : 파라미터 이름, 타이프, 값 등을 설정
- <rosparam> : rosparam명령어처럼 load, dump, delete 등 파라미터 정보의 확인 및 수정
- <group> : 실행되는 노드를 그룹화할 때 사용
- <test> : 노드를 테스트할 때 사용. <node>와 비슷하지만 테스트에 사용할 수 있는 옵션들이 추가되어 있음
- **<arg>** : launch파일 내에 변수를 정의할 수 있어 다음과 같이 실행할 때 파라미터를 변경시킬 수 있음

-The end-