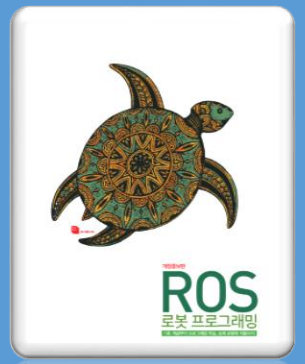
ROS 기본 프로그래밍

ROBOTIS

Open Source Team

Yoonseok Pyo

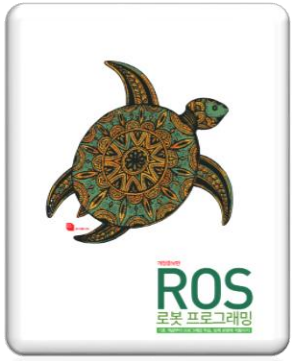


온라인강좌

YouTube Subscribe

교재
P. 156~202

Contents



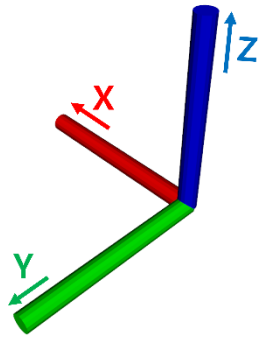
온라인강좌

교재
P. 156~202

ROS 프로그래밍 전에 알아둬야 할 사항

ROS 프로그래밍 전에 알아둬야 할 사항

- **표준 단위**
  - SI 단위 사용



Quantity	Unit
angle	radian
frequency	hertz
force	newton
power	watt
voltage	volt

Quantity	Unit
length	meter
mass	kilogram
time	second
current	ampere
temperature	celsius

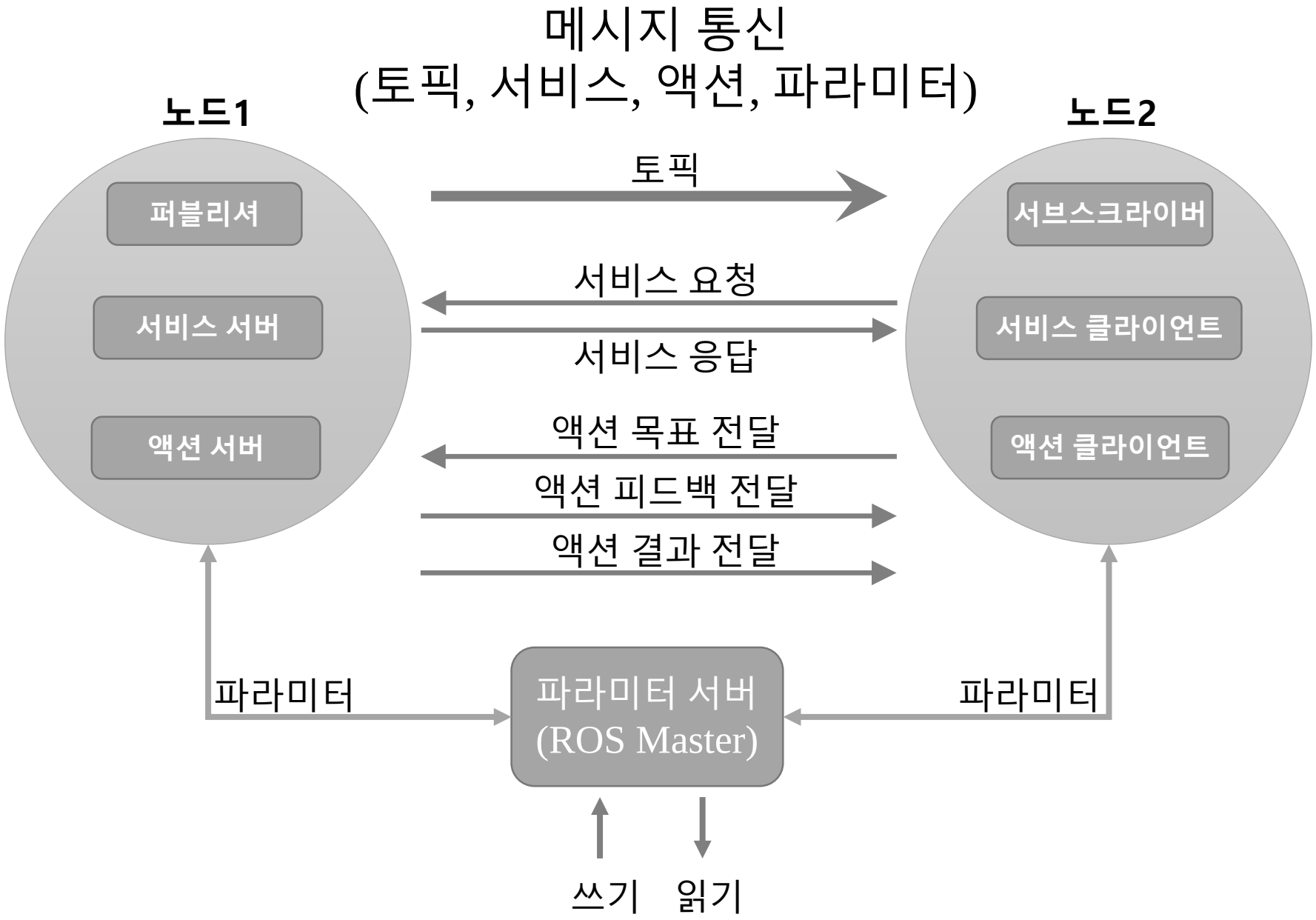
- **좌표 표현 방식**
  - x: forward, y: left, z: up
  - 오른손 법칙

- **프로그래밍 규칙**

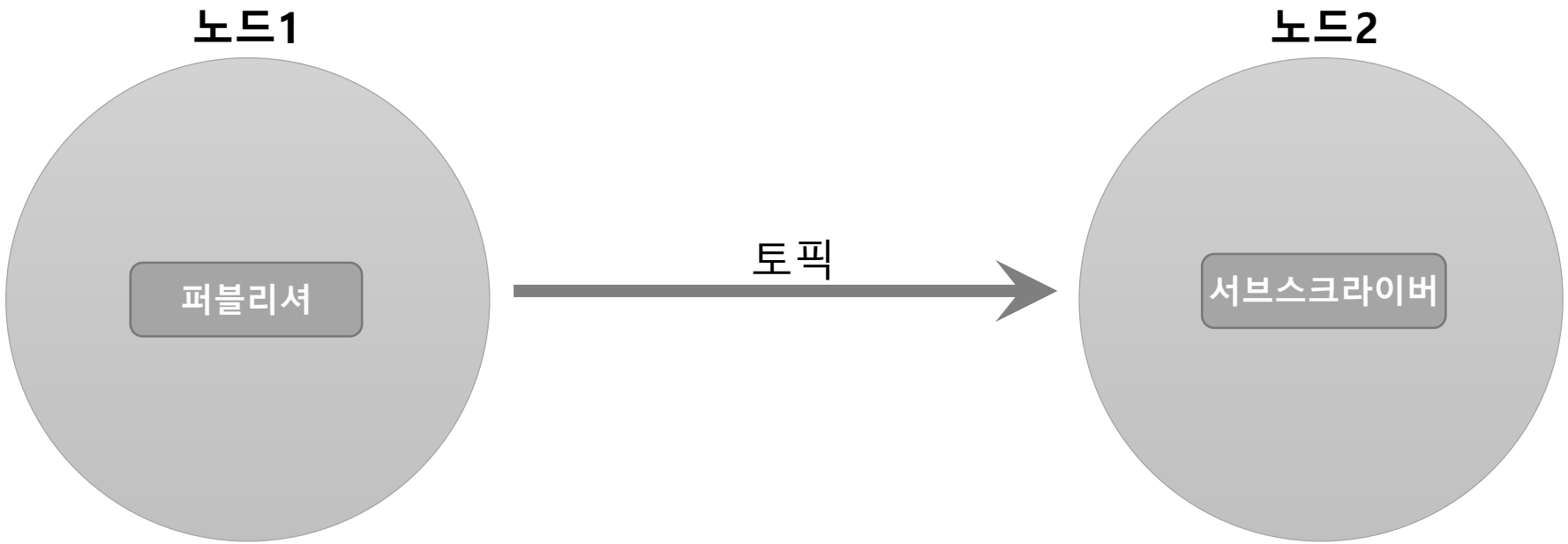
대상	명명 규칙	예제
패키지	under_scored	Ex) first_ros_package
토픽, 서비스	under_scored	Ex) raw_image
파일	under_scored	Ex) turtlebot3_fake.cpp
네임스페이스	under_scored	Ex) ros_awesome_package
변수	under_scored	Ex) string table_name;
타입	CamelCased	Ex) typedef int32_t PropertiesNumber;
클래스	CamelCased	Ex) class UrlTable
구조체	CamelCased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

메시지 통신의 종류

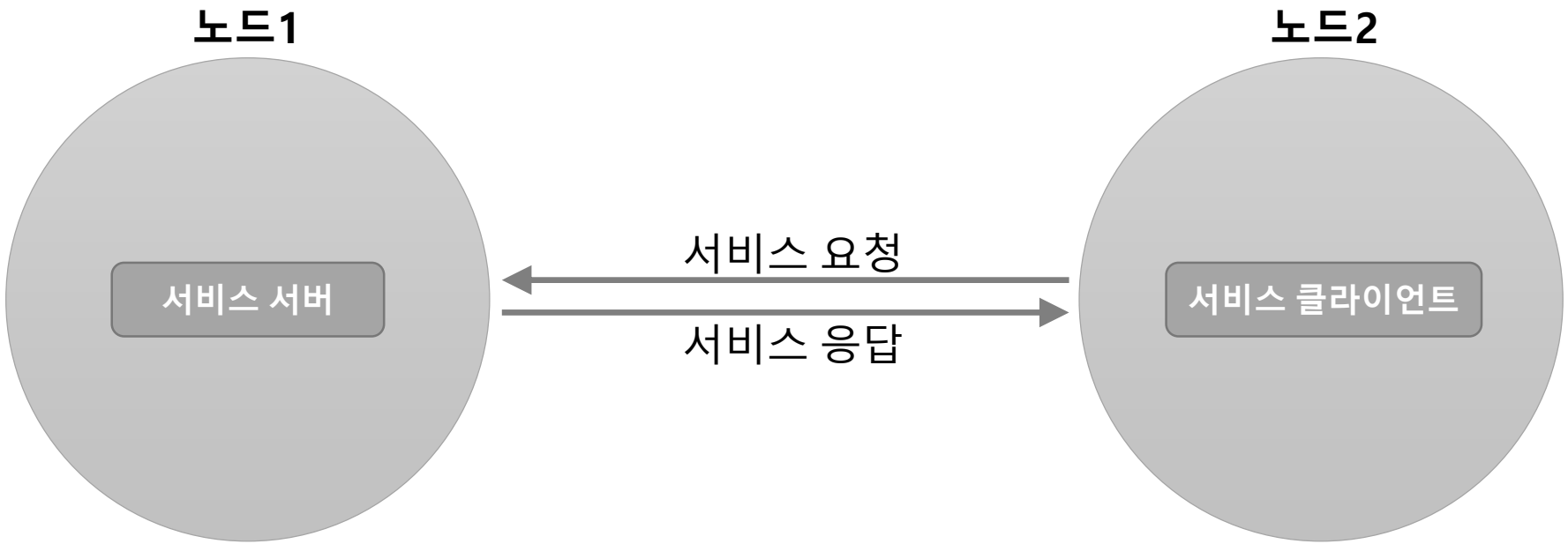
ROS 메시지 통신


메시지 통신
(토픽, 서비스, 액션, 파라미터)
노드1
노드2
퍼블리셔
서비스 서버
액션 서버
서브스크라이버
서비스 클라이언트
액션 클라이언트
토픽
서비스 요청
서비스 응답
액션 목표 전달
액션 피드백 전달
액션 결과 전달
파라미터
파라미터 서버
(ROS Master)
파라미터
쓰기
읽기

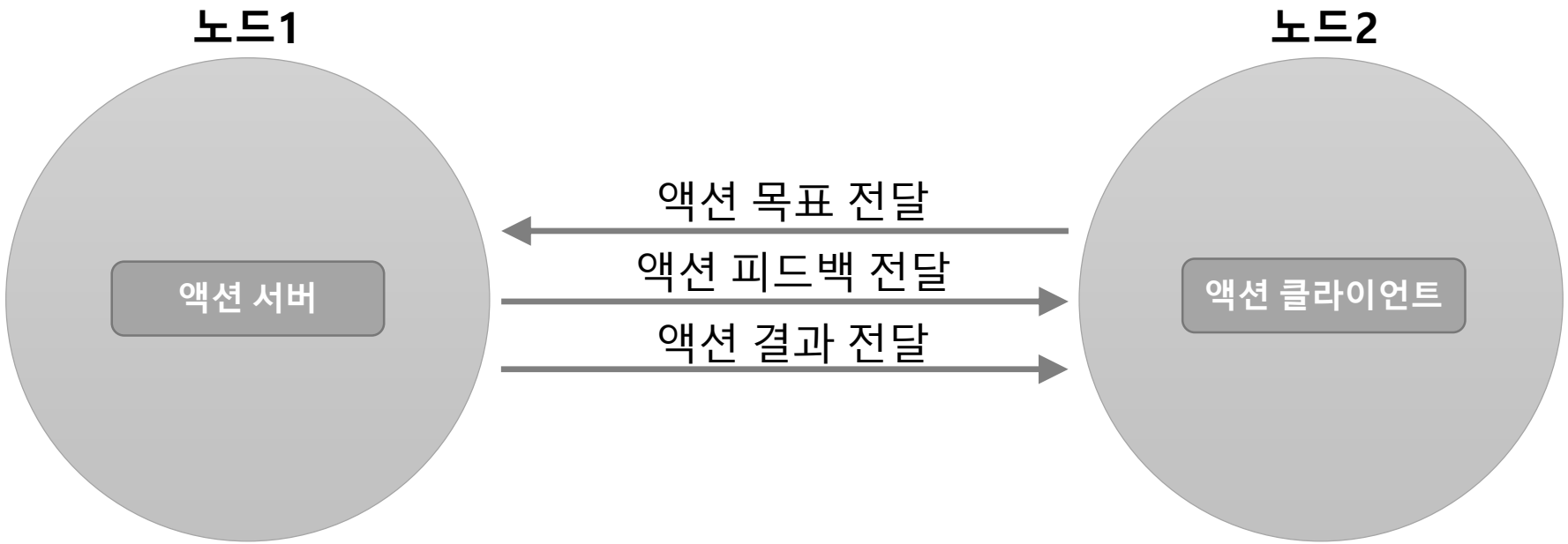
토픽(Topic)



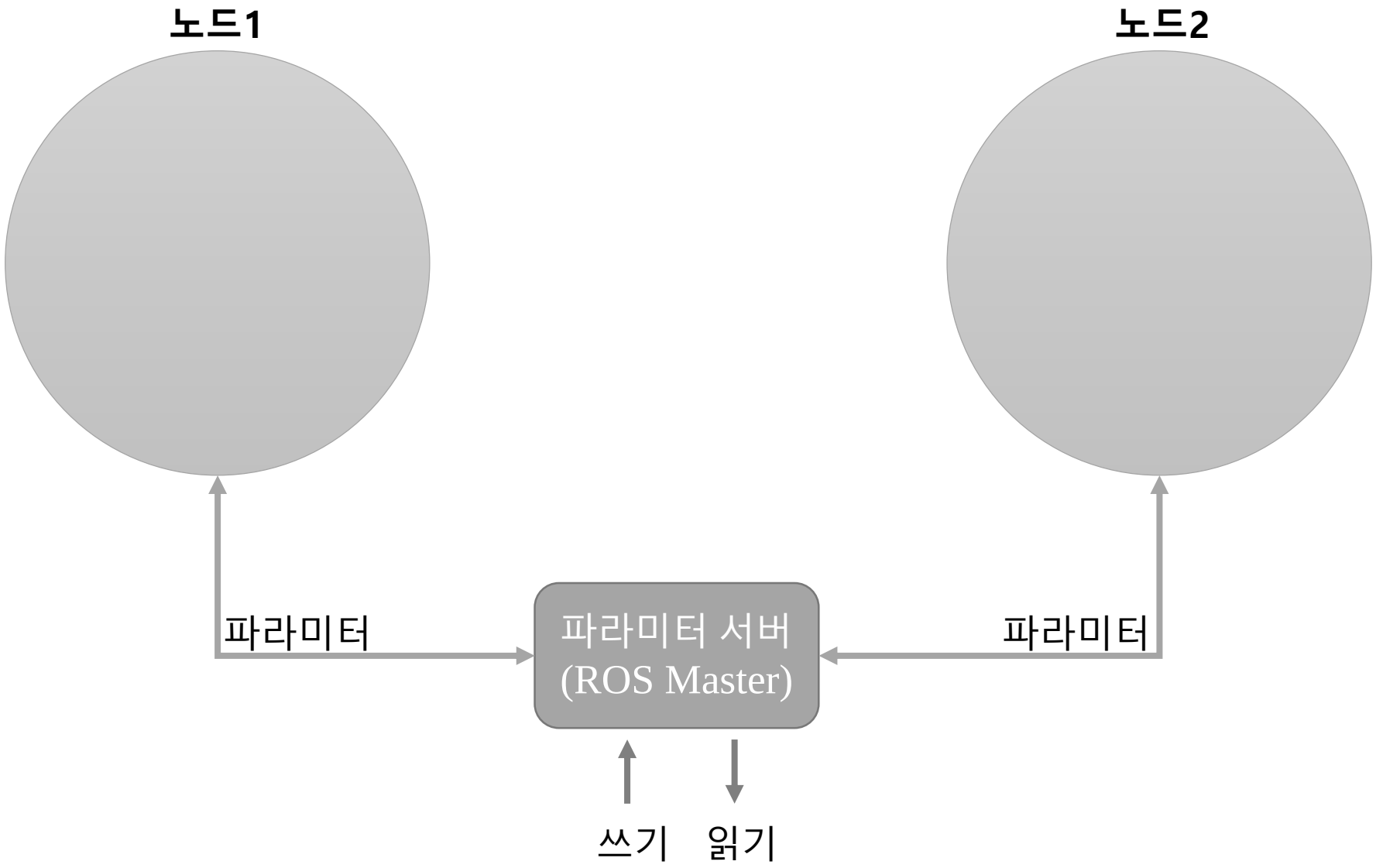
서비스(Service)



액션(Action)



매개변수(Parameter)



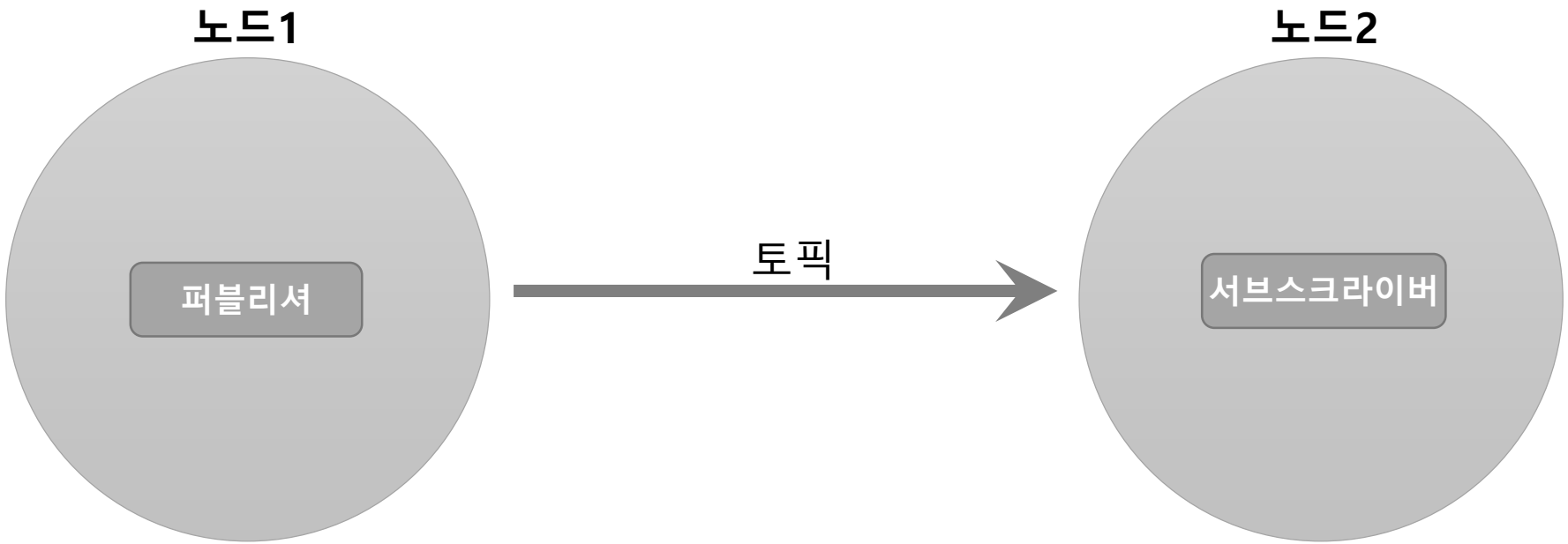
자~ 이번에는

Topic: 퍼블리셔, 서브스크라이버

Service: 서비스 서버, 서비스 클라이언트

작성에 들어갑니다.

토픽(Topic)



Topic / Publisher / Subscriber

- ROS에서는 단방향 통신일때 'Topic' 이라는 메시지 통신을 사용한다. 이때 송신 측을 'Publisher', 수신 측을 'Subscriber'라고 부른다.

1) 패키지 생성

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

```
$ cd ros_tutorials_topic
$ ls
include          → 헤더 파일 폴더
src              → 소스 코드 폴더
CMakeLists.txt   → 빌드 설정 파일
package.xml      → 패키지 설정 파일
```

Topic / Publisher / Subscriber

2) 패키지 설정 파일(package.xml) 수정

- ROS의 필수 설정 파일 중 하나인 package.xml은 패키지 정보를 담은 XML 파일로서 패키지 이름, 저작자, 라이선스, 의존성 패키지 등을 기술하고 있다.

```
$ 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 >
```

Topic / Publisher / Subscriber

```
  <buildtool_depend>catkin</buildtool_depend>
  <depend>roscpp</depend>
  <depend>std_msgs</depend>
  <depend>message_generation</depend>
  <export></export >
</package>
```

Package Format v2

- package.xml의 태그는 2가지의 다른 버전의 포맷이 있다.

- 최근 제안된 package 포맷2는 기능적으로 더 많은 내용을 포함하고 있고 편리하기에
  이 강의에서는 package 포맷2를 기준으로 설명한다. 책에는 package 포맷1을 기준으로 설명하였다.

- Package Format v2에 대한 내용은 하기 링크 참조.

- http://docs.ros.org/jade/api/catkin/html/howto/format2/index.html

Topic / Publisher / Subscriber

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
)
```

Topic / Publisher / Subscriber

```
## 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})
```

Topic / Publisher / Subscriber

4) 메시지 파일 작성

- 앞서 CMakeLists.txt 파일에 다음과 같은 옵션을 넣었다.

```
add_message_files(FILES MsgTutorial.msg)
```

- 노드에서 사용할 메시지인 MsgTutorial.msg를 빌드할 때 포함하라는 이야기

```
$ mkdir msg          → ros_tutorials_topic 패키지에 msg라는 메시지 폴더를 신규 작성
$ cd msg             → 작성한 msg 폴더로 이동
$ gedit MsgTutorial.msg  → MsgTutorial.msg 파일 신규 작성 및 내용 수정
$ cd ..              → ros_tutorials_topic 패키지 폴더로 이동
```

- Time (메시지 형식), stamp (메시지 이름)
- int32 (메시지 형식), data (메시지 이름)
- 메시지 타입은 time과 int32 이외에도 bool, int8, int16, float32, string, time, duration 등의 메시지 기본 타입과 ROS 에서 많이 사용되는 메시지를 모아놓은 common_msgs 등도 있다. 여기서는 간단한 예제를 만들어 보기 위한 것으로 time과 int32를 이용하였다. (부록C 및 http://wiki.ros.org/msg 를 참고 할 것!)

```
time stamp
int32 data
```

Topic / Publisher / Subscriber

5) 퍼블리셔 노드 작성

- 앞서 CMakeLists.txt 파일에 다음과 같은 실행 파일을 생성하는 옵션을 주었다.

```
add_executable(topic_publisher src/topic_publisher.cpp)
```

- src 폴더의 topic_publisher.cpp라는 파일을 빌드하여 topic_publisher라는 실행 파일을 만들라는 이야기

```
$ cd src                                  → ros_tutorials_topic 패키지의 소스 폴더인 src 폴더로 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
```

Topic / Publisher / Subscriber

```
// 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;
```

Topic / Publisher / Subscriber

```
 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;
}
```

Topic / Publisher / Subscriber

6) 서브스크라이버 노드 작성

- 앞서 CMakeLists.txt 파일에 다음과 같은 실행 파일을 생성하는 옵션을 주었다.

```
add_executable(topic_subscriber src/topic_subscriber.cpp)
```

- 즉, topic_subscriber.cpp라는 파일을 빌드하여 topic_subscriber라는 실행 파일을 만들라는 이야기

```
$ roscd ros_tutorials_topic/src          → 패키지의 소스 폴더인 src 폴더로 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
}
```

Topic / Publisher / Subscriber

```
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;
}
```

Topic / Publisher / Subscriber

7) ROS 노드 빌드

- 다음 명령어로 ros_tutorials_topic 패키지의 메시지 파일, 퍼블리셔 노드, 서브스크라이버 노드를 빌드하자.

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

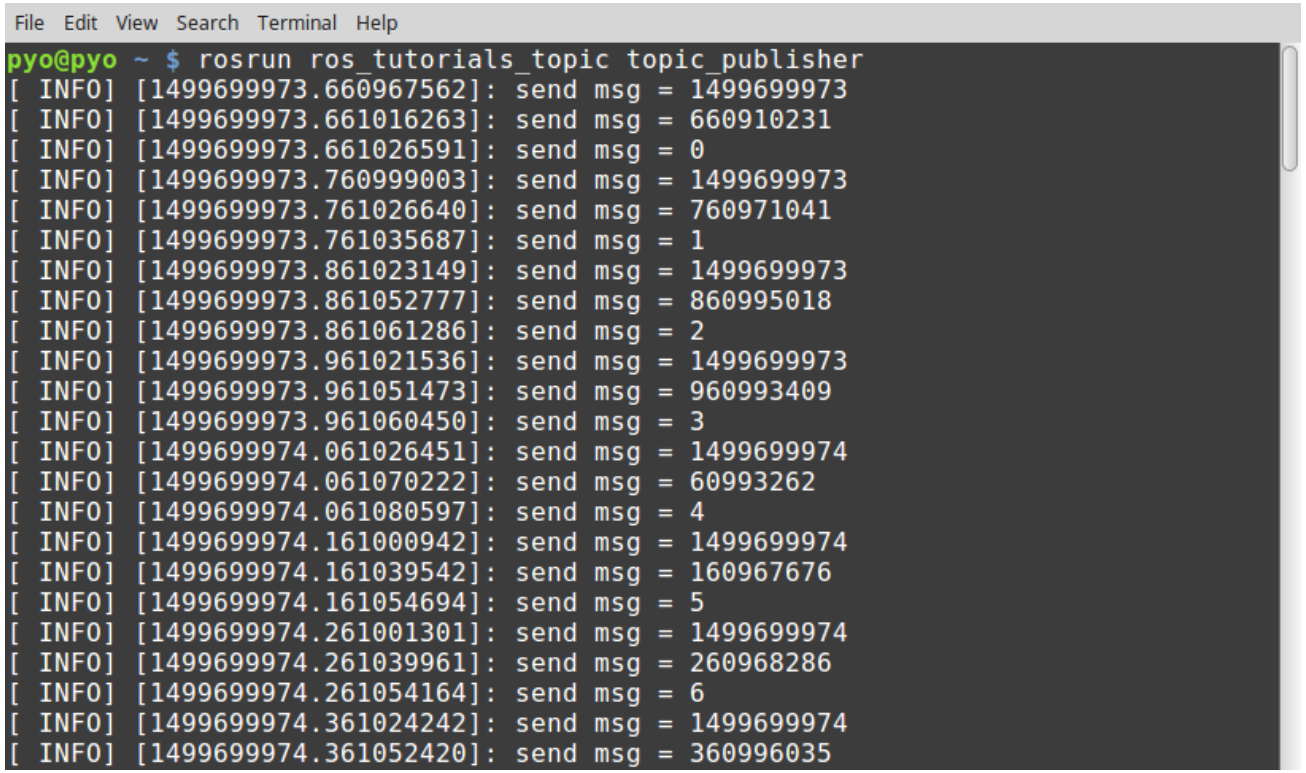
- [참고] 파일시스템
  - ros_tutorials_topic 패키지의 **소스 코드 파일**: ~/catkin_ws/src/ros_tutorials_topic/src
  - ros_tutorials_topic 패키지의 **메시지 파일**: ~/catkin_ws/src/ros_tutorials_topic/msg
  - 빌드된 결과물은 /catkin_ws의 /build와 /devel 폴더에 각각 생성
    - /build 폴더에는 캐킨 빌드에서 사용된 **설정** 내용이 저장
    - /devel/lib/ros_tutorials_topic 폴더에는 **실행 파일**이 저장
    - /devel/include/ros_tutorials_topic 폴더에는 메시지 파일로부터 자동 생성된 **메시지 헤더 파일**이 저장

Topic / Publisher / Subscriber

8) 퍼블리셔 실행 [참고로 노드 실행에 앞서 roscore를 실행하는 것을 잊지 말자.]

- ROS 노드 실행 명령어인 rosrun을 이용하여, ros_tutorials_topic 패키지의 topic_publisher 노드를 구동하라는 명령어

```
$ rosrun ros_tutorials_topic topic_publisher
```



Topic / Publisher / Subscriber

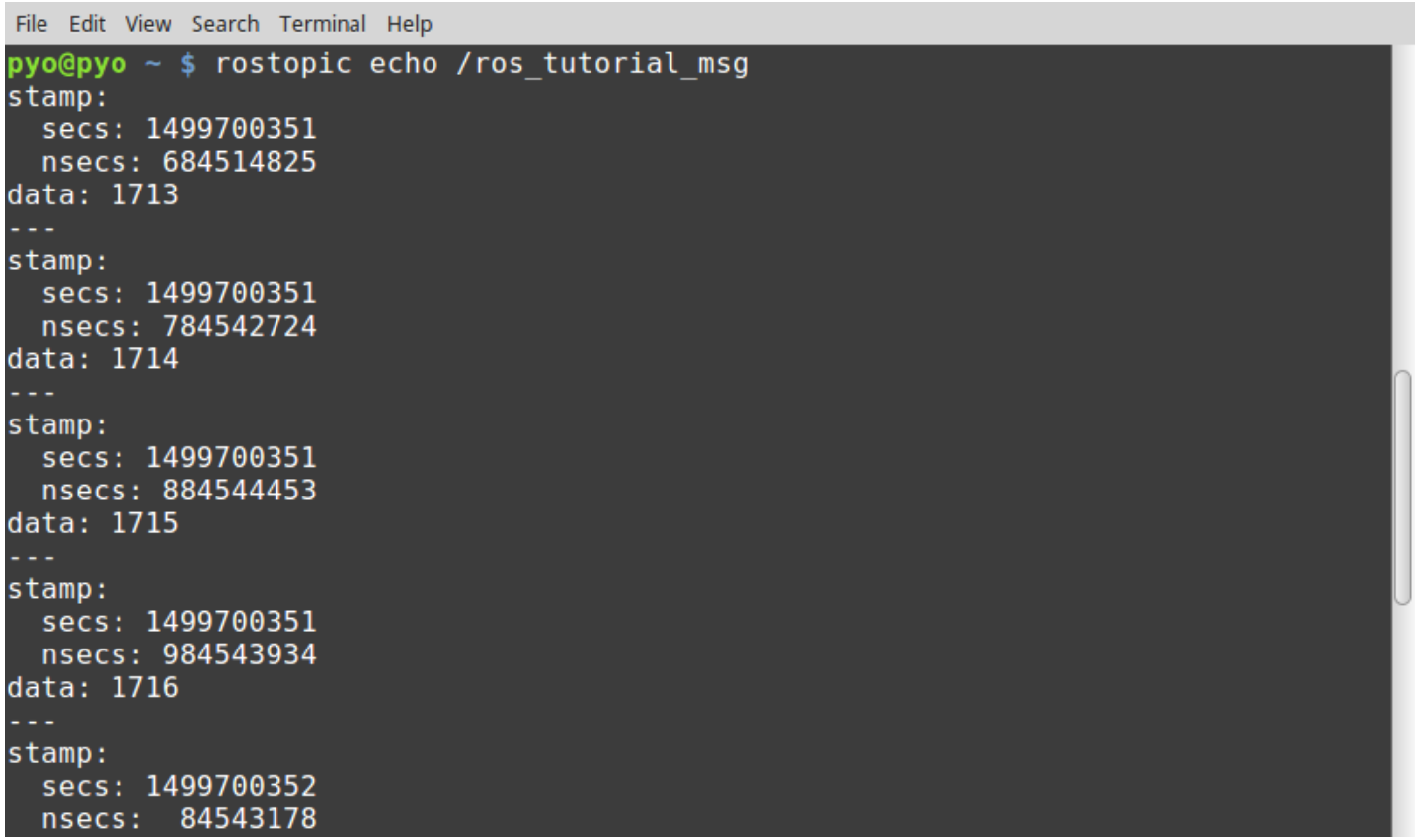
- [참고] rostopic
  - rostopic 명령어를 이용하여 현재 ROS 네트워크에서 사용 중인 토픽 목록, 주기, 데이터 대역폭, 내용 확인 등이 가능하다.

```
$ rostopic list
/ros_tutorial_msg
/rosout
/rosout_agg
```

```
$ rostopic echo /ros_tutorial_msg
```

```
$ rostopic hz /ros_tutorial_msg
```

```
$ rostopic bw /ros_tutorial_msg
```



Topic / Publisher / Subscriber

9) 서브스크라이버 실행

- ROS 노드 실행 명령어인 rosrun을 이용하여, ros_tutorials_topic 패키지의 topic_subscriber 노드를 구동하라는 명령어

```
$ rosrun ros_tutorials_topic topic_subscriber
```

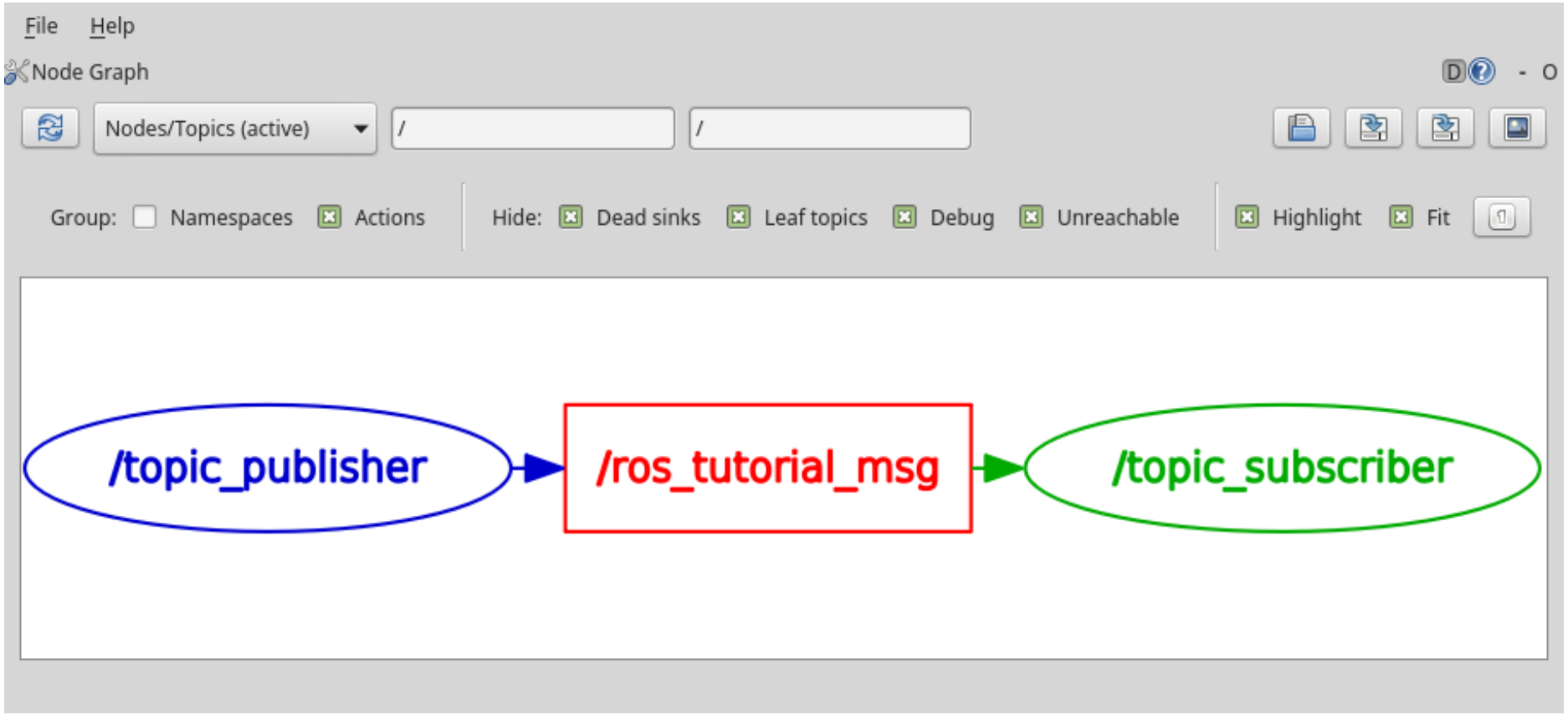
```
File  Edit  View  Search  Terminal  Help
pyo@pyo ~ $ rosrun ros_tutorials_topic topic_subscriber
[ INFO] [1499700485.184875537]: recieve msg = 1499700485
[ INFO] [1499700485.184946471]: recieve msg = 184567102
[ INFO] [1499700485.184957742]: recieve msg = 3048
[ INFO] [1499700485.284812298]: recieve msg = 1499700485
[ INFO] [1499700485.284836776]: recieve msg = 284574255
[ INFO] [1499700485.284844492]: recieve msg = 3049
[ INFO] [1499700485.384811804]: recieve msg = 1499700485
[ INFO] [1499700485.384839629]: recieve msg = 384569171
[ INFO] [1499700485.384849957]: recieve msg = 3050
[ INFO] [1499700485.484795619]: recieve msg = 1499700485
[ INFO] [1499700485.484824179]: recieve msg = 484569717
[ INFO] [1499700485.484838747]: recieve msg = 3051
[ INFO] [1499700485.584792760]: recieve msg = 1499700485
[ INFO] [1499700485.584820628]: recieve msg = 584569677
[ INFO] [1499700485.584830560]: recieve msg = 3052
[ INFO] [1499700485.684824324]: recieve msg = 1499700485
[ INFO] [1499700485.684852121]: recieve msg = 684581217
[ INFO] [1499700485.684861556]: recieve msg = 3053
[ INFO] [1499700485.785495346]: recieve msg = 1499700485
[ INFO] [1499700485.785527583]: recieve msg = 785156898
[ INFO] [1499700485.785552403]: recieve msg = 3054
[ INFO] [1499700485.884855517]: recieve msg = 1499700485
[ INFO] [1499700485.884885781]: recieve msg = 884544763
```

Topic / Publisher / Subscriber

10) 실행된 노드들의 통신 상태 확인

```
$ rqt_graph
```

```
$ rqt  [플러그인(Plugins)] → [인트로스펙션(Introspection)] → [노드 그래프(Node Graph)
```



소스코드

- 토픽에서 사용되는 퍼블리셔와 서브스크라이버 노드를 작성하고 이를 실행해서 노드 간 토픽 통신 방법을 알아보았다. 관련 소스는 하기의 github 주소에서 확인할 수 있다.

- https://github.com/ROBOTIS-GIT/ros_tutorials/tree/master/ros_tutorials_topic

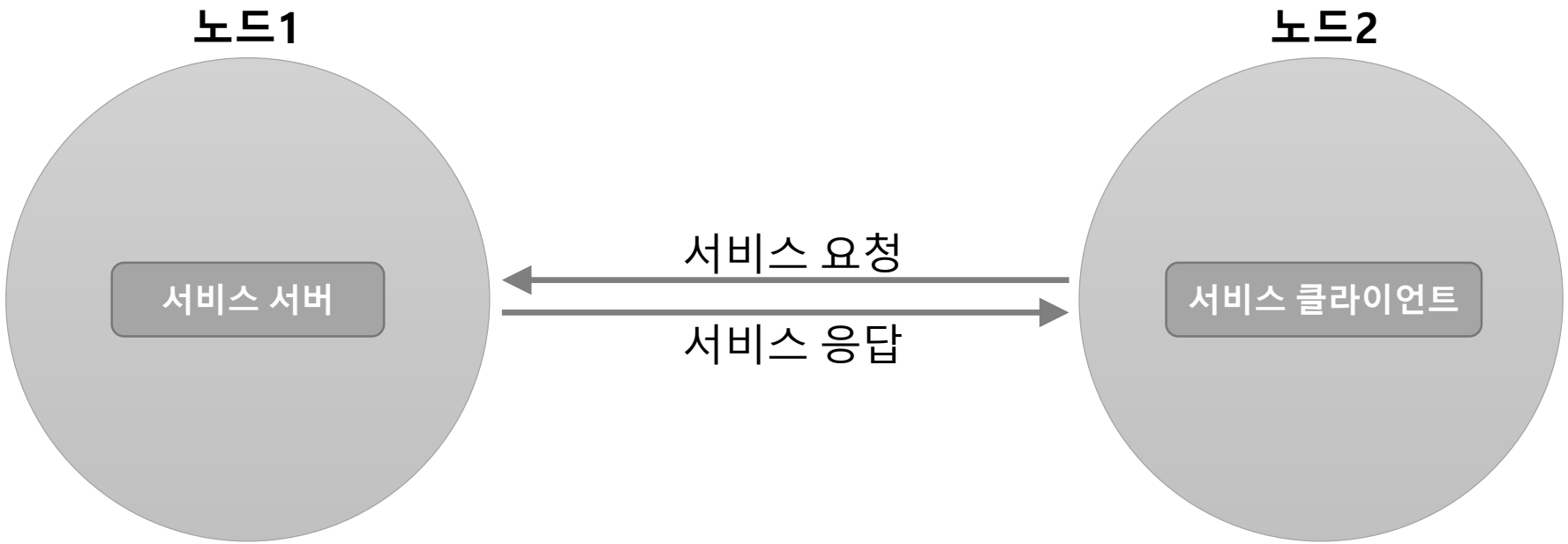
- 바로 적용해보고 싶다면 catkin_ws/src 폴더에서 다음 명령어로 소스 코드를 클론한 후 빌드를 실행하면 된다. 그 뒤 topic_publisher 및 topic_subscriber 노드를 실행하면 된다.

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

```
$ rosrun ros_tutorials_topic topic_publisher
```

```
$ rosrun ros_tutorials_topic topic_subscriber
```

서비스(Service)



Service / Service server / Service client

- 서비스는 요청(request)이 있을 때만 응답(response)하는 서비스 서버(service server)와 요청하고 응답받는 서비스 클라이언트(service client)로 나뉜다. 서비스는 토픽과는 달리 일회성 메시지 통신이다. 따라서 서비스의 요청과 응답이 완료되면 연결된 두 노드는 접속이 끊긴다.
- 이러한 서비스는 로봇에 특정 동작을 수행하도록 요청할 때에 명령어로써 많이 사용된다. 혹은 특정 조건에 따라 이벤트를 발생해야 할 노드에 사용된다. 또한, 일회성 통신 방식이라서 네트워크에 부하가 적기 때문에 토픽을 대체하는 수단으로도 사용되는 등 매우 유용한 통신수단이다.
- 이번 강의에서는 간단한 서비스 파일을 작성해보고, 서비스 서버(server) 노드와 서비스 클라이언트(client) 노드를 작성하고 실행하는 것을 목적으로 한다.

Service / Service server / Service client

- ROS에서는 양방향 통신이 필요할때 '**Service**' 이라는 메시지 통신을 사용한다. 이때 요청(request)이 있을 때만 응답(response)하는 '**서비스 서버(service server)**'와 요청하고 응답 받는 '**서비스 클라이언트 (service client)**'로 나뉜다.

1) 패키지 생성

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

```
$ cd ros_tutorials_service
$ ls
include                 → 헤더 파일 폴더
src                     → 소스 코드 폴더
CMakeLists.txt          → 빌드 설정 파일
package.xml             → 패키지 설정 파일
```

Service / Service server / Service client

2) 패키지 설정 파일(package.xml) 수정

- ROS의 필수 설정 파일 중 하나인 package.xml은 패키지 정보를 담은 XML 파일로서 패키지 이름, 저작자, 라이선스, 의존성 패키지 등을 기술하고 있다.

```
$ gedit package.xml
```

```
<?xml version="1.0"?>
<package>
  <name>ros_tutorials_service</name>
  <version>0.1.0</version>
  <description>ROS turtorial package to learn the service</description>
  <license>Apache License 2.0</license>
  <author email="pyo@robotis.com">Yoonseok Pyo</author>
  <maintainer email="pyo@robotis.com">Yoonseok Pyo</maintainer>
  <url type="bugtracker">https://github.com/ROBOTIS-GIT/ros_tutorials/issues</url>
  <url type="repository">https://github.com/ROBOTIS-GIT/ros_tutorials.git</url>
  <url type="website">http://www.robotis.com</url>
```

Service / Service server / Service client

```
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>roscpp</build_depend>
  <build_depend>std_msgs</build_depend>
  <build_depend>message_generation</build_depend>
  <run_depend>roscpp</run_depend>
  <run_depend>std_msgs</run_depend>
  <run_depend>message_runtime</run_depend>
  <export></export>
</package>
```

Service / Service server / Service client

3) 빌드 설정 파일(CMakeLists.txt) 수정

```
$ gedit CMakeLists.txt
```

```
cmake_minimum_required(VERSION 2.8.3)
project(ros_tutorials_service)

## 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)

## 서비스 선언: SrvTutorial.srv
add_service_files(FILES SrvTutorial.srv)

## 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_service
  CATKIN_DEPENDS std_msgs roscpp
)
```

Service / Service server / Service client

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

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

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

Service / Service server / Service client

4) 서비스 파일 작성

- 앞서 CMakeLists.txt 파일에 다음과 같은 옵션을 넣었다.

```
add_service_files(FILES SrvTutorial.srv)
```

- 노드에서 사용할 메시지인 SrvTutorial.srv를 빌드할 때 포함하라는 이야기

```
$ roscd ros_tutorials_service      → 패키지 폴더로 이동
$ mkdir srv                        → 패키지에 srv라는 서비스 폴더를 신규 작성
$ cd srv                           → 작성한 srv 폴더로 이동
$ gedit SrvTutorial.srv            → SrvTutorial.srv 파일 신규 작성 및 내용 수정
```

- int64 (메시지 형식), a, b (서비스 요청: request), result (서비스 응답: response), '---' (요청과 응답을 구분하는 구분자)

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

Service / Service server / Service client

5) 서비스 서버 노드 작성

- 앞서 CMakeLists.txt 파일에 다음과 같은 실행 파일을 생성하는 옵션을 주었다.

```
add_executable(service_server src/service_server.cpp)
```

- src 폴더의 service_server.cpp라는 파일을 빌드하여 service_server라는 실행 파일을 만들라는 이야기

```
$ roscd ros_tutorials_service/src          → 패키지의 소스 폴더인 src 폴더로 이동
$ gedit service_server.cpp                 → 소스 파일 신규 작성 및 내용 수정
```

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

Service / Service server / Service client

```
// 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;
}
```

Service / Service server / Service client

```
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;
}
```

Service / Service server / Service client

6) 서비스 클라이언트 노드 작성

- 앞서 CMakeLists.txt 파일에 다음과 같은 실행 파일을 생성하는 옵션을 주었다.

```
add_executable(service_client src/service_client.cpp)
```

- src 폴더의 service_client.cpp라는 파일을 빌드하여 service_client라는 실행 파일을 만들라는 이야기

```
$ roscd ros_tutorials_service/src          → 패키지의 소스 폴더인 src 폴더로 이동
$ gedit service_client.cpp                 → 소스 파일 신규 작성 및 내용 수정
```

```
#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
```

Service / Service server / Service client

```
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");
```

Service / Service server / Service client

```
// 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;
}
```

Service / Service server / Service client

7) ROS 노드 빌드

- 다음 명령어로 ros_tutorials_service 패키지의 서비스 파일, 서비스 서버 노드와 클라이언트 노드를 빌드한다.

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

- [참고] 파일시스템
  - ros_tutorials_service 패키지의 **소스 코드 파일**: ~/catkin_ws/src/ros_tutorials_service/src
  - ros_tutorials_service 패키지의 **메시지 파일**: ~/catkin_ws/src/ros_tutorials_service/msg
  - 빌드된 결과물은 /catkin_ws의 /build와 /devel 폴더에 각각 생성
    - /build 폴더에는 캐킨 빌드에서 사용된 **설정** 내용이 저장
    - /devel/lib/ros_tutorials_service 폴더에는 **실행 파일**이 저장
    - /devel/include/ros_tutorials_service 폴더에는 메시지 파일로부터 자동 생성된 **메시지 헤더 파일**이 저장

Service / Service server / Service client

8) 서비스 서버 실행

[참고로 노드 실행에 앞서 roscore를 실행하는 것을 잊지 말자.]

- 서비스 서버는 서비스 요청이 있기 전까지 아무런 처리를 하지 않고 기다리도록 프로그래밍하였다. 그러므로 다음 명령어를 실행하면 서비스 서버는 서비스 요청을 기다린다.

```
$ rosrun ros_tutorials_service service_server
[INFO] [1495726541.268629564]: ready srv server!
```

Service / Service server / Service client

9) 서비스 클라이언트 실행

- 서비스 서버를 실행했으면 이어서 다음 명령어로 서비스 클라이언트를 실행한다.

```
$ rosrun ros_tutorials_service service_client 2 3
[INFO] [1495726543.277216401]: send srv, srv.Request.a and b: 2, 3
[INFO] [1495726543.277258018]: receive srv, srv.Response.result: 5
```

- 서비스 클라이언트를 실행하면서 입력해준 실행 매개변수 2와 3을 서비스 요청 값으로 전송하도록 프로그래밍하였다.
- 2와 3은 각각 a, b 값으로 서비스를 요청하게 되고, 결괏값으로 둘의 합인 5를 응답값으로 전송 받았다.
- 여기서는 단순하게 실행 매개변수로 이를 이용하였으나, 실제 활용에서는 명령어로 대체해도 되고, 계산되어야 할 값, 트리거용 변수 등을 서비스 요청 값으로 사용할 수도 있다.

Service / Service server / Service client

[참고] rosservice call 명령어 사용 방법

- 서비스 요청은 서비스 클라이언트 노드를 실행하는 방법도 있지만 "rosservice call"이라는 명령어나 rqt의 ServiceCaller를 이용하는 방법도 있다.
- 그 중 우선, rosservice call를 사용하는 방법에 대해서 알아보자.

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

$ rosservice call /ros_tutorial_srv 5 15
result: 20
```

Service / Service server / Service client

[참고] GUI 도구인 Service Caller 사용 방법

- ROS의 GUI 도구인 rqt를 실행하자.

```
$ rqt
```

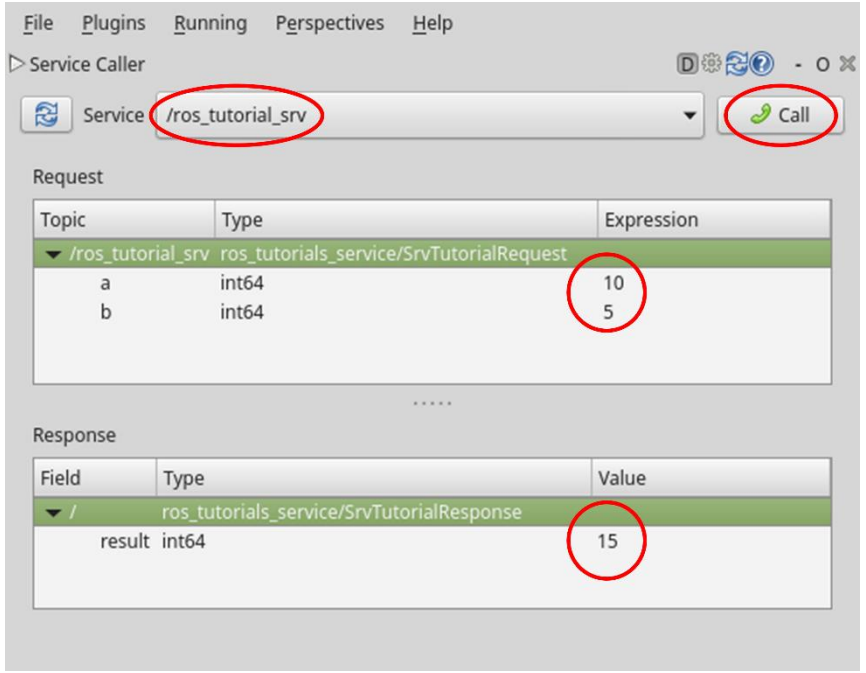
- rqt 프로그램의 메뉴에서 [플러그인(Plugins)] → [서비스(Service)] → [Service Caller]를 선택하면 다음과 같은 화면이 나온다.

  (1) servic에 /ros_tutorial_srv 입력

  (2) a = 10, b = 5 입력

  (3) Call 버튼을 누른다.

  (4) Result에 15라는 값이 표시된다.



한 가지 더!!!

- 하나의 노드는 복수의 퍼블리셔, 서브스크라이버, 서비스 서버, 서비스 클라이언트 역할도 할 수 있다!
- 마음껏 요리해 보세요~ ^^

```
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");
```

소스코드

- 서비스 서버와 클라이언트 노드를 작성하고 이를 실행해보면서 노드 간 서비스 통신 방법에 대해 알아보았다. 관련 소스는 하기의 GitHub 주소에서 확인할 수 있다.
- https://github.com/ROBOTIS-GIT/ros_tutorials/tree/master/ros_tutorials_service

- 바로 적용해보고 싶다면 catkin_ws/src 폴더에서 다음 명령어로 소스 코드를 클론한 후 빌드를 실행하면 된다. 그 뒤 service_server 및 service_client 노드를 실행하면 된다.

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

```
$ rosrun ros_tutorials_service service_server
```

```
$ rosrun ros_tutorials_service service_client
```

파라미터

Parameter

1) 파라미터를 활용한 노드 작성

- 이전 강의에서는 작성한 서비스 서버와 클라이언트 노드에서 service_server.cpp 소스를 수정하여 서비스 요청으로 입력된 a와 b를 단순히 덧셈하는 것이 아니라, 사칙연산을 할 수 있도록 파라미터를 활용해 볼 것이다.
- 다음 순서대로 service_server.cpp 소스를 수정해보자.

```
$ roscd ros_tutorials_service/src          → 패키지의 소스 코드 폴더인 src 폴더로 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;
```

Parameter

```
// 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;
```

Parameter

```
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;
}
```

Parameter

```
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!");
```

Parameter

```
  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;
}
```

- 파라미터 관련하여 설정(setParam) 및 읽기(getParam) 사용법에 주목 할 것!

[참고] 파라미터로 사용 가능 형태

- 파라미터는 integers, floats, boolean, string, dictionaries, list 등으로 설정할 수 있다.
- 간단히 예를 들자면, 1은 integer, 1.0은 floats, “Internet of Things”은 string, true는 boolean, [1,2,3]은 integers의 list, a: b, c: d는 dictionary이다.

Parameter

2) 노드 빌드 및 실행

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

```
$ rosrun ros_tutorials_service service_server
[INFO] [1495767130.149512649]: ready srv server!
```

3) 매개변수 목록 보기

- "rosparam list" 명령어로 현재 ROS 네트워크에 사용된 파라미터의 목록을 확인할 수 있다. /calculation_method가 우리가 사용한 파라미터이다.

```
$ rosparam list
/calculation_method
/rosdistro
/rosversion
/run_id
```

Parameter

4) 파라미터 사용 예

- 다음 명령어대로 파라미터를 설정하여, 매번 같은 서비스 요청을 하여 서비스 처리가 달라짐을 확인해보자.
- ROS에서 파라미터는 노드 외부로부터 노드의 흐름이나 설정, 처리 등을 을 바꿀 수 있다. 매우 유용한 기능이므로 지금 당장 쓰지 않더라도 꼭 알아두도록 하자.

```
$ 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
```

소스코드

- 기존에 작성해둔 서비스 서버를 수정하여 파라미터를 사용하는 방법에 대해 알아보았다.
- 관련 소스는 기존에 작성해둔 서비스 소스 코드와 구별짓기 위하여 ros_tutorials_parameter 패키지라 이름을 바꾸어 작성해두었고 하기의 github 주소에서 확인할 수 있다.

- https://github.com/ROBOTIS-GIT/ros_tutorials/tree/master/ros_tutorials_parameter

- 바로 적용해보고 싶다면 catkin_ws/src 폴더에서 다음 명령어로 소스 코드를 클론한 후 빌드를 실행하면 된다. 그 뒤 service_server 및 service_client 노드를 실행하면 된다.

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

```
$ rosrun ros_tutorials_parameter service_server_with_parameter
```

```
$ rosrun ros_tutorials_parameter service_client_with_parameter 2 3
```

roslaunch

roslaunch 사용법

- rosrun이 하나의 노드를 실행하는 명령어하는 명령어 이다.
- roslaunch는 하나 이상의 정해진 노드를 실행시킬 수 있다.

- 그 밖의 기능으로 노드를 실행할 때 패키지의 매개변수나 노드 이름 변경, 노드 네임스페이스 설정, ROS_ROOT 및 ROS_PACKAGE_PATH 설정, 환경 변수 변경 등의 옵션을 붙일 수 있는 ROS 명령어이다.

- roslaunch는 '*.launch'라는 파일을 사용하여 실행 노드를 설정하는데 이는 XML 기반이며, 태그별 옵션을 제공한다.

- 실행 명령어는 "roslaunch [패키지명] [roslaunch 파일]"이다.

roslaunch 사용법

1) roslaunch의 활용

- 이전에 작성한 topic_publisher와 topic_subscriber 노드의 이름을 바꾸어서 실행해 보자. 이름만 바꾸면 의미가 없으니, 퍼블리쉬 노드와 서브스크라이버 노드를 각각 두 개씩 구동하여 서로 별도의 메시지 통신을 해보자.
- 우선, *.launch 파일을 작성하자. roslaunch에 사용되는 파일은 *.launch라는 파일명을 가지며 해당 패키지 폴더에 launch라는 폴더를 생성하고 그 폴더 안에 넣어야 한다. 다음 명령어대로 폴더를 생성하고 union.launch라는 새 파일을 생성해보자.

```
$ roscd ros_tutorials_topic
$ mkdir launch
$ cd launch
$ gedit union.launch
```

roslaunch 사용법

```
<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>
```

- **<launch>** 태그 안에는 roslaunch 명령어로 노드를 실행할 때 필요한 태그들이 기술된다. **<node>**는 roslaunch로 실행할 노드를 기술하게 된다. 옵션으로는 pkg, type, name이 있다.
  - **pkg** 패키지의 이름
  - **type** 실제 실행할 노드의 이름(노드명)
  - **name** 위 type에 해당하는 노드가 실행될 때 붙여지는 이름(실행명), 일반적으로는 type과 같게 설정하지만 필요에 따라 실행할 때 이름을 변경하도록 설정할 수 있다.

roslaunch 사용법

- roslaunch 파일을 작성하였으면 다음처럼 union.launch를 실행하자.

```
$ roslaunch ros_tutorials_topic union.launch --screen
```

- [참고] roslaunch 사용시에 상태 출력 방법
  - roslaunch 명령어로여럿의 노드가 실행될 때는 실행되는 노드들의 출력(info, error 등)이 터미널 스크린에 표시되지 않아 디버깅하기 어렵게 된다. 이때에 --screen 옵션을 추가해주면 해당 터미널에 실행되는 모든 노드들의 출력들이 터미널 스크린에 표시된다.

roslaunch 사용법

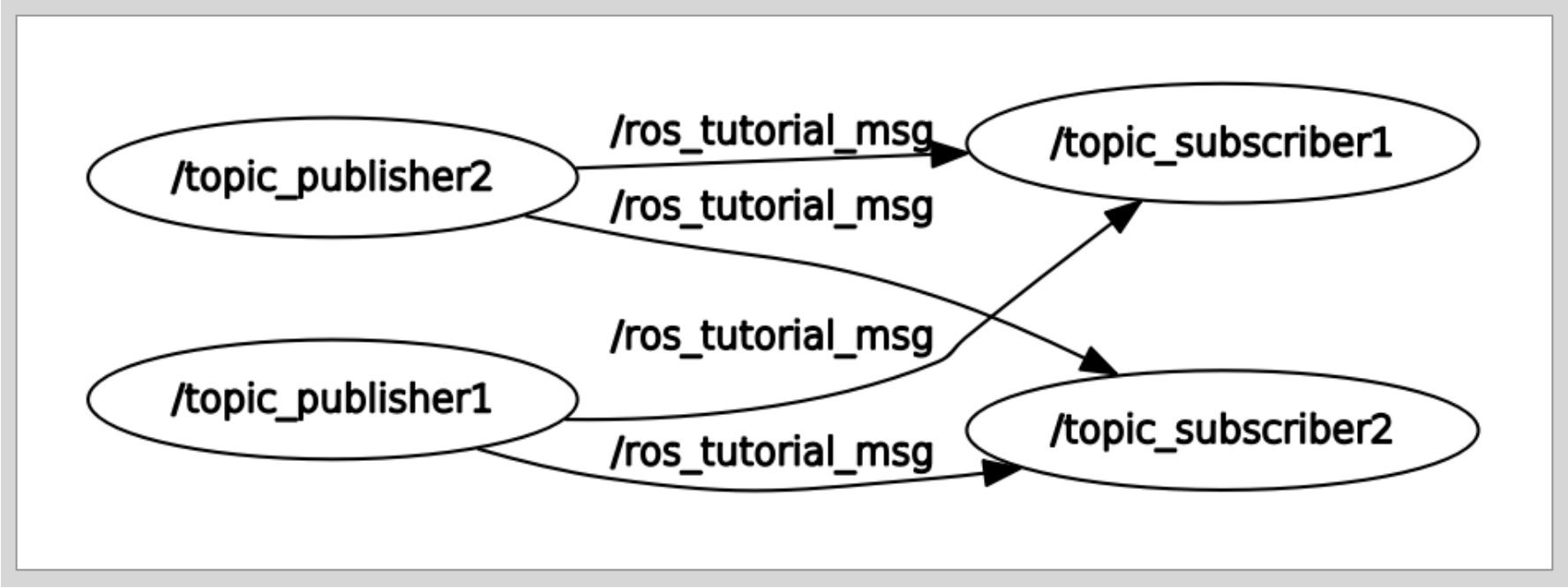
- 실행 결과?

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

- 결과적으로 topic_publisher 노드가 topic_publisher1과 topic_publisher2로 이름이 바뀌어 두 개의 노드가 실행되었다.
- topic_subscriber 노드도 topic_subscriber1과 topic_subscriber2로 이름이 바뀌어 실행되었다.

roslaunch 사용법

- 문제는 "퍼블리쉬 노드와 서브스크라이버 노드를 각각 두 개씩 구동하여 서로 별도의 메시지 통신하게 한다"는 처음 의도와는 다르게 rqt_graph를 통해 보면 서로의 메시지를 모두 서브스크라이브하고 있다는 것이다.
- 이는 단순히 실행되는 노드의 이름만을 변경해주었을 뿐 사용되는 메시지의 이름을 바꿔주지 않았기 때문이다.
- 이 문제를 다른 roslaunch 네임스페이스 태그를 사용하여 해결해보자.



roslaunch 사용법

- union.launch을 수정하자.

```
$ roscd ros_tutorials_service/launch
$ gedit union.launch
```

```
<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>
```

roslaunch 사용법

- 변경 후의 실행된 노드들의 모습

roslaunch 사용법

2) launch 태그

<launch>	roslaunch 구문의 시작과 끝을 가리킨다.
<node>	노드 실행에 대한 태그이다. 패키지, 노드명, 실행명을 변경할 수 있다.
<machine>	노드를 실행하는 PC의 이름, address, ros-root, ros-package-path 등을 설정할 수 있다.
<include>	다른 패키지나 같은 패키지에 속해 있는 다른 launch를 불러와 하나의 launch파일처럼 실행할 수 있다.
<remap>	노드 이름, 토픽 이름 등의 노드에서 사용 중인 ROS 변수의 이름을 변경할 수 있다.
<env>	경로, IP 등의 환경 변수를 설정한다. (거의 안쓰임)
<param>	매개변수 이름, 타이프, 값 등을 설정한다
<rosparam>	rosparam 명령어 처럼 load, dump, delete 등 매개변수 정보의 확인 및 수정한다.
<group>	실행되는 노드를 그룹화할 때 사용한다.
<test>	노드를 테스트할 때 사용한다. <node>와 비슷하지만 테스트에 사용할 수 있는 옵션들이 추가되어 있다.
<arg>	launch 파일 내에 변수를 정의할 수 있어서 아래와 같이 실행할 때 매개변수를 변경 시킬 수도 있다.

```
<launch>
  <arg name="update_ period" default="10" />
  <param name="timing" value="$(arg update_ period)"/>
</launch>

roslaunch my_package my_package.launch update_ period:=30
```

이번 강좌의 리포지토리

- Example 리포지토리
  - https://github.com/ROBOTIS-GIT/ros_tutorials

- 웹에서 바로 보거나 다운로드도 가능하지만 바로 적용해 보고 싶다면 다음 명령어로 리포지토리를 복사하여 빌드하면 된다.

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

이제 ROS 초보탈출 Bye Bye~!

질문
대환영!

* 온라인 상의 질문이라면
오르카 및 로열로를 이용해주세요!

여기서! 광고 하나 나가요~

개정 증보판

✔ 한국어판 구매 링크

✔ 4개 언어로 출판!

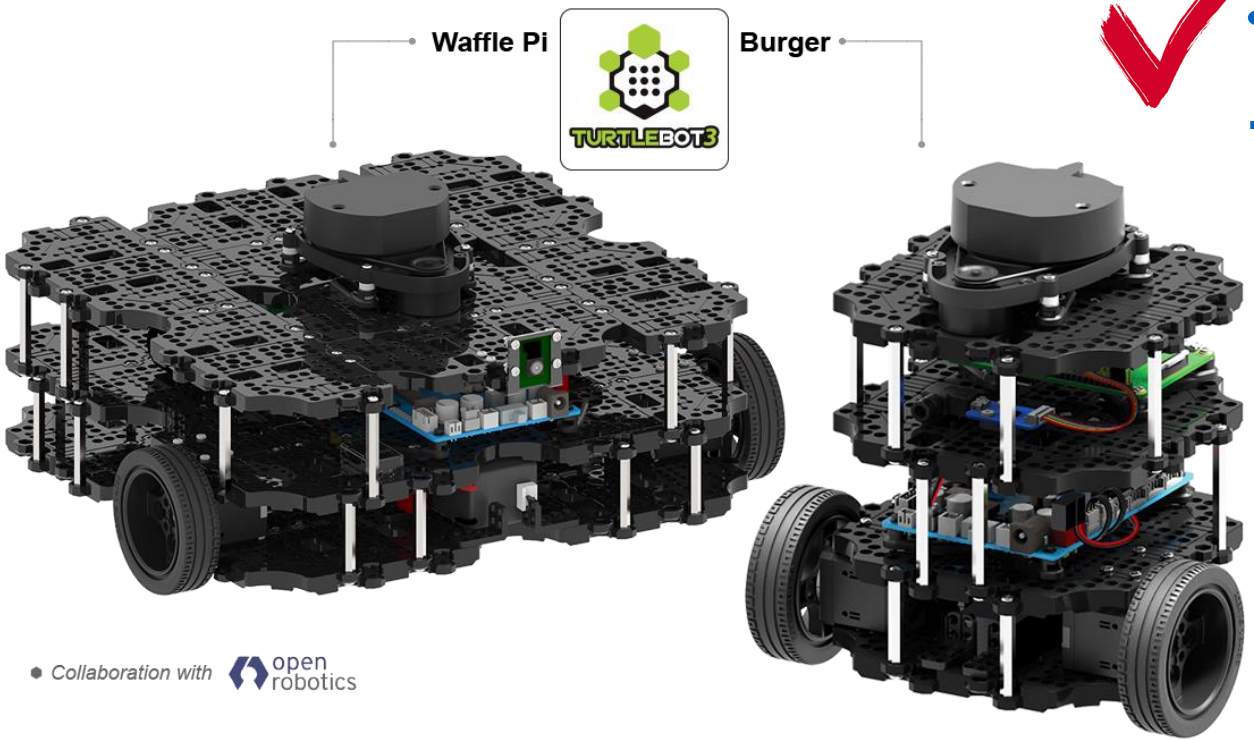
국내 유일! 최초! ROS 참고서!
ROS 공식 플랫폼 **TurtleBot3** 개발팀이
직접 저술한 바이블급 ROS 책

여기서! 광고 둘 나가요~

TURTLEBOT3

인공지능(AI) 연구의 시작,
ROS 교육용 공식 로봇 플랫폼

터틀봇3는 ROS기반의 저가형 모바일 로봇으로 교육, 연구, 제품개발, 취미 등 다양한 분야에서 활용할 수 있습니다.



Direct Link

여기서! 광고 셋 나가요~

오로카

- 오로카
- www.oroca.org
- 오픈 로보틱스 지향
- 풀뿌리 로봇공학의 저변 활성화
- 공개 강좌, 세미나, 프로젝트 진행

- 로봇공학을 위한 열린 모임 (KOS-ROBOT)
- www.facebook.com/groups/KoreanRobotics
- 로봇공학 통합 커뮤니티 지향
- 일반인과 전문가가 어울러지는 한마당
- 로봇공학 정보 공유
- 연구자 간의 협력

Robot Source

- RobotSource
- www.robotsource.org
- 글로벌 로보틱스 커뮤니티 지향
- 로봇공학 정보 공유
- 자신의 로봇 프로젝트 공유
- DIY 로봇 프로젝트 진행

혼자 하기에 답답하시다고요?

커뮤니티에서 함께 해요~

끝.

표윤석

Yoonseok Pyo
pyo@robotis.com
www.robotpilot.net

www.facebook.com/yoonseok.pyo