ORC-PG-ROS2-WEEK9

13장 - 토픽 프로그래밍(C++)

토픽(Topic)은 비동기식 단방향 메시지 송수신 방식으로 메시지를 퍼블리시하는 퍼블리셔(Publisher)와 메시지를 서브스크라이브하는 서브스크라이버(Subscriber) 간의 통신이다. 이는
1 : 1 통신을 기본으로 하지만 복수의 노드에서 하나의 토픽을 송수신하는 1 : N도 가능하고 그 구성
방식에 따라 N : I, N : N 통신도 가능하다. 이는 ROS 메시지 통신에서 가장 널리 사용되는 통신 방법
이다.

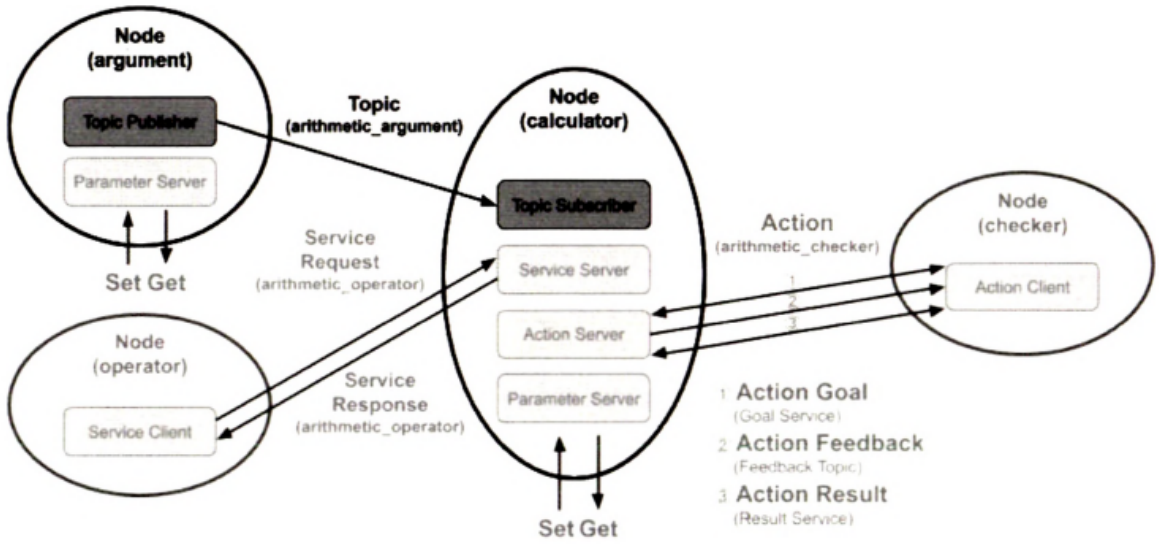


그림 13-1 토픽 퍼블리셔와 서브스크라이버

토픽 퍼블리셔 코드

토픽 퍼블리셔 역할을 하는 argument 노드의 전체 소스코드는 다음과 같다.

```
#ifndef ARITHMETIC__ARGUMENT_HPP_
#define ARITHMETIC__ARGUMENT_HPP_

#include <chrono> 시간을 다루는 라이브러리
#include <memory> 동적 메모리를 다루는 라이브러리
#include <string> 문자열을 다루는 라이브러리
#include <utility> 다양한 기능을 담고 있는 라이브러리

#include "rclcpp/rclcpp.hpp" rclcpp 헤더파일
#include "msg_srv_action_interface_example/msg/arithmetic_argument.hpp"

class Argument : public rclcpp:：Node
{
public:
using ArithmeticArgument =
msg_srv_action_interface_example::msg::ArithmeticArgument;
explicit Argument(const rclcpp::NodeOptions & node_options = rclcpp：：
NodeOptions());
virtual ~Argument();
private:
void publish_random_arithmetic_arguments();

void update_parameter();
float min_random_num_;
float max_random_num_;
rclcpp::Publisher<ArithmeticArgument>::SharedPtr arithmetic_argument_publisher_;
rclcpp::TimerBase::SharedPtr timer_;
rclcpp::Subscription<rcl_interfaces::msg::ParameterEvent>::SharedPtr
parameter_event_sub_;
rclcpp::AsyncParametersClient::SharedPtr parameters_client_;
}；
#endif // ARITHMETIC__ARGUMENT_HPP_


#include <cstdio>
#include <memory>
#include <string>
```

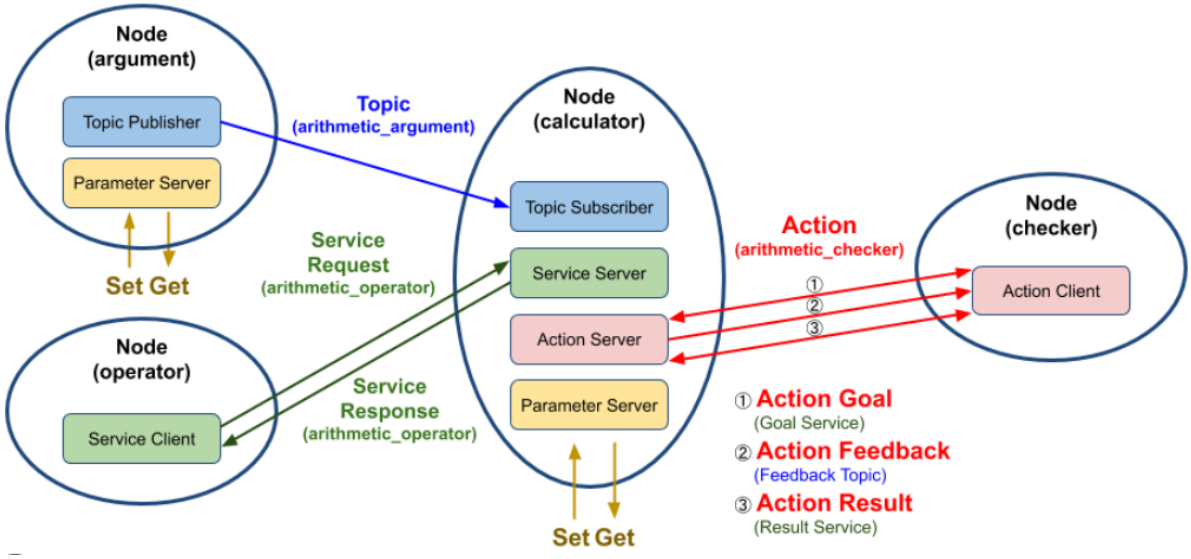
```
#include <utility>
#include <random>
#include "rclcpp/rclcpp.hpp"
#include "^cutils/cmdline_parse^.h,,
#include "arithmetic/argument.hpp"
i using namespace std：:chrono_literals;
Argument::Argument(const rclcpp::NodeOptions & node_options)
: Node("argument,\ node_options),
min_random_num_(0.0),
max_random_num_(0.0)
{
this->declare_parameter("qos_depth", 10);
int8_t qos_depth = this->get_parameter("qos_depth").get_value<int8_t>();
this->declare_parameter("min_random_num", 0.0);
min_random_num_ = this->get_parameter("min_random_nijm").get_value<float>();
this->declare_parameter("max_random__num", 9.0);
max_random_num_ = this->get_parameter("max_random_num”).get_value<float>();


this->update_parameter();
const auto QOS_RKL10V =
rclcpp::QoS(rclcpp::KeepLast(qos_depth)).reliable().durability_voIatile();
arithmetic_argument_publisher_ =
this->create_p)lisher<ArithmeticArgument>("arithmetic_argument,\ QOS_RKL10V);
timer_ =
this->create_wall_timer(!s,
std::bind(&Argument::publish_rando(n_arithmetic_arguments, this));
}
Argument: :~Argument()
{
}
void Argument::publish_random_arithmetic_arguments()
{
std：:random_device rd;
std::mt19937 gen(rd());
std：:uniform_real_distrib┗┫tion<float> distribution(niin_random_num_,
max_random_num_);
msg_srv_action_interface_example:msg::ArithmeticArg┗┃ment msg;
msg.stamp = this->now();
```

```
msg.argument_a = distribution(gen);
msg.argument_b = distribution(gen);
arithmetic_argument_publisher_->publish(msg);
RCLCPP_INFO(this->get_logger(), "Published argument_a %.2f", msg.argument_a);
RCLCPP_INFO(this->get_logger(), "Published argument_b %.2f", msg.argument_b);
}
void Argument::update_parameter()
{
parameters_client_ = std::make_sharedrclcpp::AsyncParametersClient(this);
while (!parameters_client_->wait_for_service(ls)) {
if (Irclcpp：:ok()) {
RCLCPP_ERROR(this->get_logger(), "Interrupted while waiting for the service.
Exiting.”);
return;
}
```

~~~ 이하생략

rclcpp의 Node 클래스를 상속받는 Argument 클래스에 대해

Argument 클래스의 생성자는 rclcpp의 NodeOptions6 객체를 인자로 받는다. NodeOptions 객체를 통해서는 context, arguments, intra-process comm니nication, parameter, allocator와 같은 Node 생성을 위한 다양한 옵션을 정할 수 있다.

토픽 서브스크라이버 코드

caculator 노드 : 토픽 서브스크라이버 역할

subscriber : clcpp::Node

노드 실행 코드

두 개의 노드 실행 명령어

$ ros2 run topic_service_action_rclcpp_example calculator (서브스크라이버 노드)
$ ros2 run topic_service_action_rclcpp_example argument (토픽 퍼블리셔 노드)
~~~

14장 서비스 프로그래밍(C++)

서비스(Service)는 동기식 양방향 메시지 송수신 방식으로 서비스의 요청(Request)을 하는 쪽
을 서비스 클라이언트(Service Client)라고 하며 요청받은 서비스를 수행한 후 서비스의 응답
(Response)을 하는 쪽을 서비스 서버(Service Server)라고 한다. 결국 서비스는 특정 요청을 하
는 클라이언트 단과 요청 받은 일을 수행한 후에 결괏값을 전달하는 서버 단과의 통신이다.



서비스 서버 코드

topic_service_action_rclcpp_example/include/calculator/calculator.hpp
topic_service_action_rclcpp_example/src/calculator/calculator.cpp

서비스 클라이언트 코드

topic_service_action_rclcpp_example/include/arithmetic/**operator.hpp**
topic_service_action_rclcpp_example/src/arithmetic/**operator.cpp**

노드 실행 코드

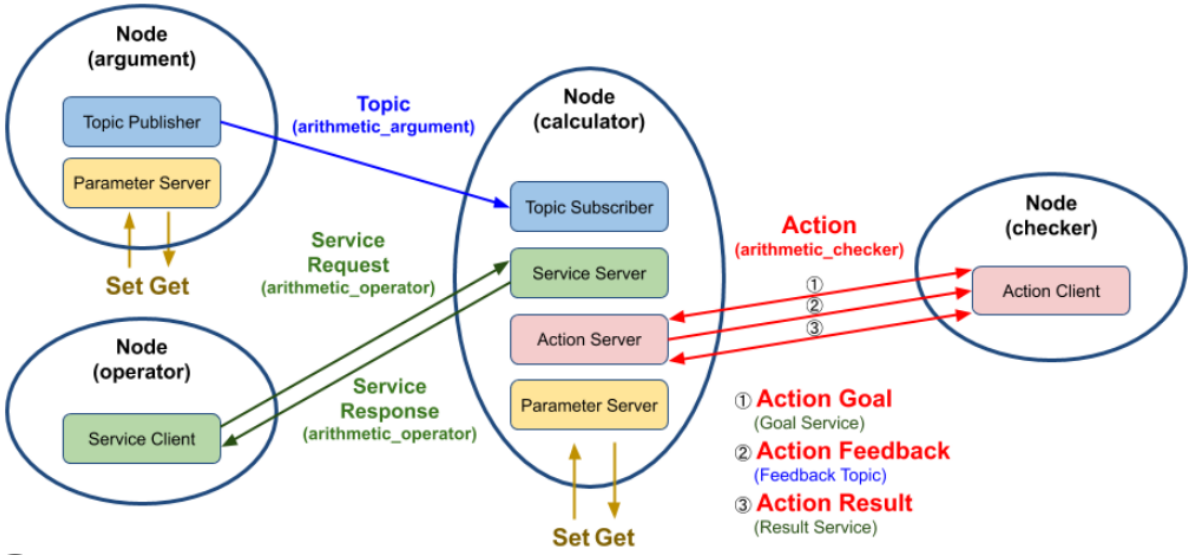
$ ros2 run topic_service_action_rclcpp_example calculator (서브스크라이버 노드)
$ ros2 run topic_service_action_rclcpp_example operator (서비스 클라이언트 노드)

Operator 노드의 메인 함수는 Operator 클래스를 객체화하고 반복문을 통해 send_reqni
est 함수

를 호출하는 로직을 가지고 있다.

15장 액션 프로그래밍(C++)

액션(Action)은 비동기식+동기식 양방향 메시지 송수신 방식으로 액션 목표(Goal)를 지정하는 액
션 클라이언트(Action client)와 액션 목표를 받아 특정 태스크를 수행하면서 중간 결괏값을 전송
하는 액션 피드백(Action feedback) 그리고 최종 결괏값에 해당되는 액션 결과(Action result)
를 전송하는 액션 서버(Action Server) 간의 통신이다.



액션 서버 코드

topic_service_action_rclcpp_example/include/calculator/calculator.hpp
topic_service_action_rclcpp_example/src/calculator/calculator.cpp

액션 클라이언트 코드

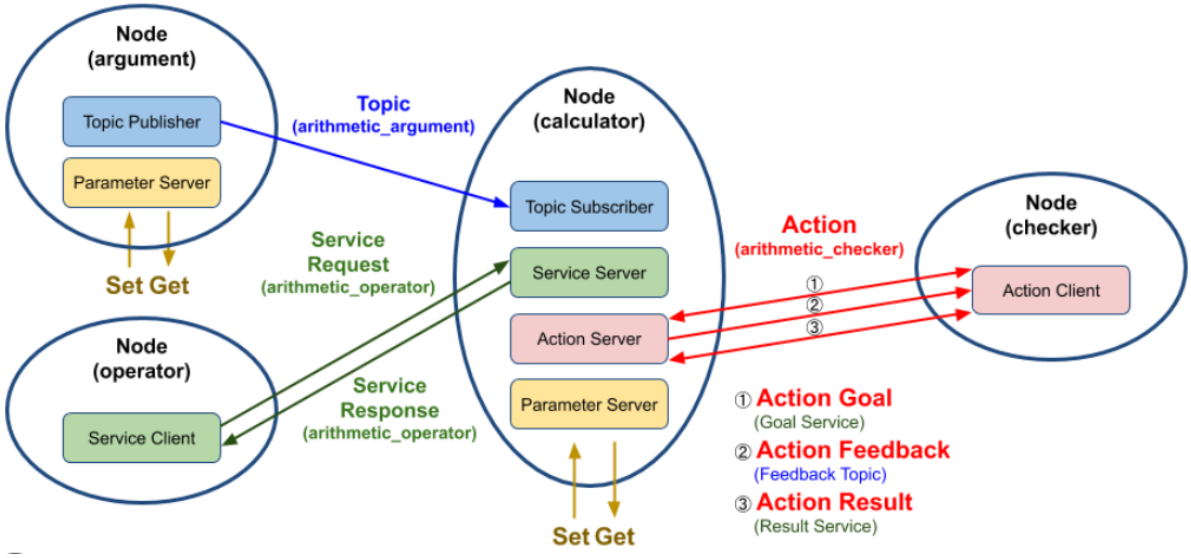
topic_service_action_rclcpp_example/include/checker/checker.hpp
topic_service_action_rclcpp_example/src/checker/checker.cpp

노드 실행 코드

$ ros2 run topic_service_action_rclcpp_example calculator
$ ros2 run topic_service_action_rclcpp_example checker

16장 파라미터 프로그래밍(C++)

ROS 2의 파라미터(Parameter)는 ROS 1의 parameter server와 dynamic_reconfigure 패
키지의 기능을 모두 가지고 있어 노드가 동작하는 동안 특정 값에 대한 저장, 변경, 회수가 가능하
다. ROS 2의 모든 노드는 파라미터 서버(Parameter server)를 가지고 있어서 파라미터 클라이
언트(Parameter client)와 서비스 통신을 통해 파라미터에 접근할 수 있도록 구현되어 있다. 이
는 1부 12장 ROS 2 서비스(Service)에서 다루었던 서비스와 그 목적은 다르지만 데이터 처리 자
체는 동일하다고 볼 수 있다. 서비스가 특정 태스크 수행을 위한 요청과 응답이라는 RPC(Remote
Procedure Call)에 가까운 목적이었다면, 파라미터는 특정 매개변수를 노드 내부 또는 외부에서 쉽
게 저장(Set)하거나 변경할 수 있고, 쉽게 회수(Get)하여 사용할 수 있게 하는 점에서 그 사용 목적
이 다르다고 볼 수 있다.



파라미터 서버에 파라미터를 등록하는 방법

1. yaml 포맷의 파일의 경로를 프로그램 실행 인자로 rclcpp::Node에 전달
2. ros2cli를 이용한 파라미터 등록
3. rclcpp::Node의 declare, set 파라미터 함수 사용
4. 파라미터 클라이언트 AI기를 이용

파라미터 클라이언트

topic_service_action_rclcpp_example/include/arithmetic/argument.hpp
topic_service_action_rclcpp_example/src/arithmetic/argument.cpp

17장 실행 인자 프로그래밍(C++)

실행인자

C十十 프로그램 실행 시 가장 먼저 호출되는 main 함수는 두 개의 매개변수를 가진다. 먼저 첫 번째 매개변수인 argc는 argument count의 약자로 넘겨받은 인자들의 개수를 담고 있다. 두 번째 매개 변수인 argv는 argument vector의 약자로 문자열, 포인터. 배열 타입으로 넘겨받은 인자들을 저장하고 있다.
ROS 2에서 실행 인자는 크게 두 가지로 분류된다. 첫 번째는 --ros-args가 붙은 인자들로 ROS 2
API와 관련된 옵션(remapping, parameter 등)을 변경할 수 있다. 두 번째는 --ros-args가 붙지
않은 인자들로 일반적으로 사용하는 사용자 정의 실행 인자라고 생각하면 된다.

18장 런치 프로그래밍(파이썬, C++)

ROS 2 Launch System

ROS 2에서는 하나의 노드를 실행시키기 위해서는 “ros2 run” 명령어를 사용한다. 이 명령어만으로도 노드를 실행시키는 것에는 큰 문제가 없다. 하지만 ROS 2에서는 하나의 노드만을 실행시키는 일보다 복수의 노드를 함께 실행시켜 노드 간의 메시지를 주고받게

되는 경우가 더 많다. 그리고 직접 개발한 패키지의 노드만을 실행하기도 하지만 이미 개발되어 공개된 패키지의 노드들을 사용하는 경우도 많다. 또한 각 노드마다 여러 개의 파라미터를 변경해야 할 때도 있을 것이다.
ROS 2의 Launch는 하나 이상의 정해진 노드를 실행시킬 수 있다. 더불어, 노드를 실행할 때 패키
지의 매개변수나 노드 이름 변경, 노드 네임스페이스 설정, 환경변수 변경 등의 옵션을 사용할 수 있다. ROS 1에서는 이를 roslaunch라 하여 “*.launch” 파일을 사용하여 실행 노드를 설정하는데 이
는 XML 기반이었으며, 여러 태그별 옵션을 제공하여 사용자 편의성을 제공하였다.
ROS 2에서는
기존 XML 방식 이외에도 파이썬 프로그래밍 방식도 추가되어 확장성을 높였다.

Launch 작성

topic_service_action_rclpy_example 패키지에 새로운 Launch 파일을 만들어 보자. 이 런치 파
일은 기본적으로 argument 노드와 calculator 노드를 실행시키는 역할을 하게 되며 두 노드에서
사용할 파라미터가 설정된 파일을 불러오는 역할을 하게 된다.

런치 파일을 사용하기 위해서는 해당 패키지에 launch 폴더가 있어야 하며 이 폴더에 런치 파일
(*.launch.py)을 만들어 사용한다.

LaunchDescription 반환 구문. DeclareLaunchArgument 클래스를 이용한 param_dir 변수를 런치 인수로 선언 → Node 클래스로 실행할 노드를 설정

기본적으로는 package, executable, name, parameters, output을 설정

package 실행할 패키지 이름을 기재
executable 실행 가능한 노드의 이름을 기재
name 지정한 노드를 실행할 때 실제로 사용할 이름을 기재
parameters 특정 파라미터 값을 / DeclareLaunchArgument에서 지정한 변수를 사용

(param_dir 변수를 사용하여 지정된 파라미터 파일(arithmetic_config.yaml)을 사용)
output 로깅 설정으로 기본적으로 특정 파일 이름(~/.ros/log/xxx/launch.log)에 로깅 정보가 기록되고 터미널 창에 출력하고 싶다면 screen이라고 지정

패키지빌드

ROS 2 에코시스템 환경에서 사용하기 위해서는 패키지 빌드를 통해 정해진 위치에 설치

Launch 파일의 빌드와 관련해서는 C++ 언어를 사용하는 RCLCPP 패키지 계열이냐 파이썬 언어를 사용하는 RCLPY 패키지 계열에 따라 다름

RCLCPP 패키지 계열

C++ 언어를 사용하는 경우 다음과 같이 빌드 설정 파일(CMakeLists.txt)의 install 구문에 launch
폴더명만 기재

```
install(DIRECTORY
  launch
  DESTINATION share/${PROJECT_NAME}/
)
```

RCLPY 패키지 계열

파이썬 언어를 사용하는 경우 파이썬 패키지 설정 파일(setup.py)의 data_files 옵션 부분에 launch 옵션을 지정

```
setup(
    name=package_name,
    version='0.6.0',
    packages=find_packages(exclude=['test']),
    data_files=[
        ('share/ament_index/resource_index/packages', ['resource/' + package_name]),
        (share_dir, ['package.xml']),
        (share_dir + '/launch', glob.glob(os.path.join('launch', '*.launch.py'))),
        (share_dir + '/param', glob.glob(os.path.join('param', '*.yaml'))),
    ],
```

빌드 (특정 패키지만을 빌드하는 cbp 사용)

$ cw
$ cbp topic_service_action_rclpy_example

실행

$ ros2 launch <package_name> <launch_file_name>