Week9

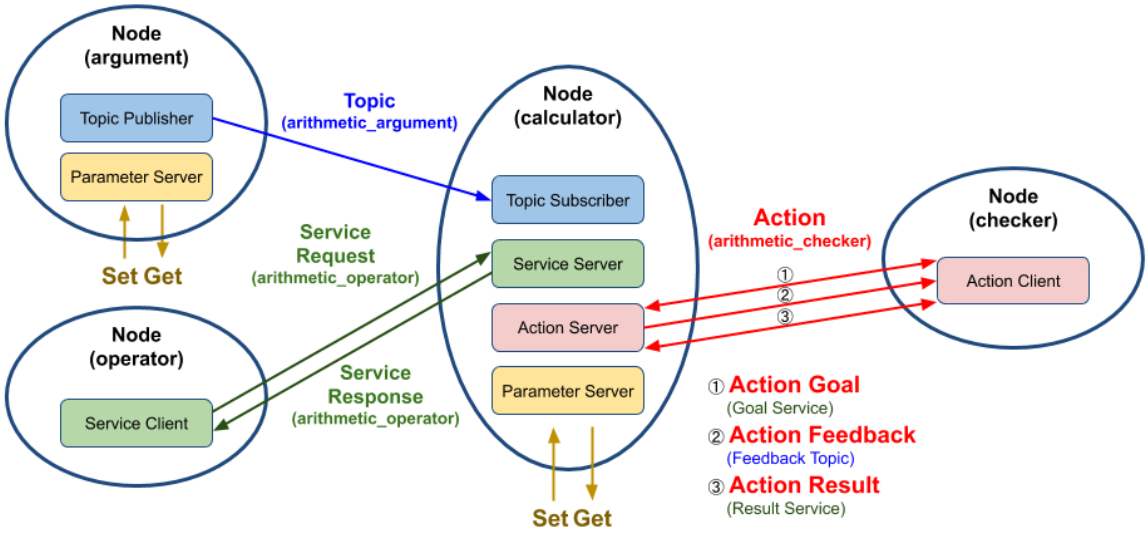
태그

내용

12장. ROS2 패키지 설계 (C++)

이제까지 배운 ROS의 토픽, 서비스, 액션을 C++으로 구현합니다.

topic_service_action_rclpy_example 를 다음과 같이 4개의 Node 로 설계합니다.



1. argument node

   arithmetic_argument 토픽으로 현재의 시간과 변수 a 와 b 를 발행합니다.

2. calculator node

   arithmetic_argument 토픽이 생성된 시간과 변수 a와 b를 구독합니다.

3. operator node

   arithmetic_operator 서비스를 통해 계산기 노드에게 연산자(+, -, *, /)를 서비스 요청 값으로 보냅니다.

4. calculator node

   a. 변수 a와 b를 연산자를 이용하여 연산

   b. operator node에게 서비스 응답값

   c. 연산 결과값을 누적하고, 액션 피드백으로 연산식을 보냅니다.

   d. 누적된 연산 결과값이 목표값보다 크면 연산 결과와 연산식을 보냅니다.

5. checker node

   a. 연산값의 합계의 한계치를 액션 목표값으로 전달

b. calculator 노드는 이를 받은 후 부터의 연산 값을 합한다.

c. 액션 피드백으로 각 연산 계산식을 보낸다.

d. 지정한 연산값의 목표 합계를 넘기면 액션 결괏값으로 최종 연산 합계를 보낸다.

노드작성

topic_service_action_rclpy_example 패키지는 argument 노드, operator 노드, calculator 노드, checker 노드로 구성되어 있습니다.

패키지 설정 파일 (package.xml)

topic_service_action_rclpy_example 패키지 설정 파일을 다음과 같이 작성 합니다.

```
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd" schematypens="http://www.w3.org/2001/XMLSchema"?>
<package format="3">
  <name>topic_service_action_rclcpp_example</name>
  <version>0.6.0</version>
  <description>ROS 2 rclcpp example package for the topic, service, action</description>
  <maintainer email="passionvirus@gmail.com">Pyo</maintainer>
  <license>Apache License 2.0</license>
  <author email="passionvirus@gmail.com">Pyo</author>
  <author email="routiful@gmail.com">Darby Lim</author>
  <buildtool_depend>ament_cmake</buildtool_depend>
  <depend>rclcpp</depend>
  <depend>rclcpp_action</depend>
  <depend>msg_srv_action_interface_example</depend>
  <test_depend>ament_lint_auto</test_depend>
  <test_depend>ament_lint_common</test_depend>
  <export>
    <build_type>ament_cmake</build_type>
  </export>
</package>
```

1. depend 태그를 통해 종속패키지 rclcpp, rclcpp_action, msg_srv_action_interface_example를 추가합니다.
2. build_type 은 ament_cmake 입니다.

빌드 설정 파일 (CMakeList.txt)

topic_service_action_rclcpp_example 패키지의 CMakeList 파일은 다음과 같습니다.

```
cmake_minimum_required(VERSION 3.5)
project(topic_service_action_rclcpp_example)

if(NOT CMAKE_C_STANDARD)
  set(CMAKE_C_STANDARD 99)
endif()
if(NOT CMAKE_CXX_STANDARD)
  set(CMAKE_CXX_STANDARD 14)
endif()

if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
  add_compile_options(-Wall -Wextra -Wpedantic)
endif()

# find dependencies
# find_package(ament_cmake REQUIRED)
# find_package(msg_srv_action_interface_example REQUIRED)
# find_package(rclcpp REQUIRED)
# find_package(rclcpp_action REQUIRED)

include_directories(include)

add_executable(argument src/arithmetic/argument.cpp)
ament_target_dependencies(argument
```

```
  msg_srv_action_interface_example
  rclcpp
)

add_executable(calculator src/calculator/main.cpp src/calculator/calculator.cpp)
ament_target_dependencies(calculator
  msg_srv_action_interface_example
  rclcpp
  rclcpp_action
)

add_executable(checker src/checker/main.cpp src/checker/checker.cpp)
ament_target_dependencies(checker
  msg_srv_action_interface_example
  rclcpp
  rclcpp_action
)

add_executable(operator src/arithmetic/operator.cpp)
ament_target_dependencies(operator
  msg_srv_action_interface_example
  rclcpp
)

if(BUILD_TESTING)
  find_package(ament_lint_auto REQUIRED)
  ament_lint_auto_find_test_dependencies()

  find_package(ament_cmake_gtest REQUIRED)
  add_subdirectory(test)
endif()

install(TARGETS
  argument
  calculator
  checker
  operator
  DESTINATION lib/${PROJECT_NAME}
)

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

##############################################################################
# Find and load build settings from external packages
##############################################################################
ament_package()
```

소스코드 내려받기 및 빌드

```
git clone https://github.com/robotpilot/ros2-seminar-examples.git
```

cbp 를 통해서 빌하는 방법

```
cbp
cbp topic_service_action_rcl
```

topic_service_action_rclcpp_example 패키지의 calculator 를 실행 합니다.

```
makepluscode@worktop:~$ ros2 run topic_service_action_rclcpp_example calculator
[INFO] [1687005716.038227437] [calculator]: Run calculator
```

topic_service_action_rclcpp_example 패키지의 argument 를 실행합니다.

```
$ ros2 run topic_service_action_rclcpp_example argument
[INFO] [1687005738.683602630] [argument]: Published argument_a 7.66
[INFO] [1687005738.683770058] [argument]: Published argument_b 5.93
```

위에서 실행된 calculator 를 보면 argument 가 수신된 것을 확인할 수 있습니다.

```
makepluscode@worktop:~$ ros2 run topic_service_action_rclcpp_example calculator
[INFO] [1687005716.038227437] [calculator]: Run calculator
[INFO] [1687005738.683842989] [calculator]: Timestamp of the message: sec 1687005738 nanosec 683533451
[INFO] [1687005738.683912308] [calculator]: Subscribed argument a: 7.66
[INFO] [1687005738.683935218] [calculator]: Subscribed argument b: 5.93
```

topic_service_action_rclcpp_example 패키지의 operator 를 실행합니다.

```
makepluscode@worktop:~$ ros2 run topic_service_action_rclcpp_example operator
Press Enter for next service call.
[INFO] [1687005880.265237185] [operator]: Result 0.36
```

위에서 실행된 calculator를 보면, 수신된 operator에 따라 두 인자(argument)의 연산 결과가 표시됩니다.

```
[INFO] [1687005879.846467914] [calculator]: Subscribed argument a: 4.07
[INFO] [1687005879.846475664] [calculator]: Subscribed argument b: 7.23
[INFO] [1687005880.265514362] [calculator]: 4.065844 * 7.227545 = 29.3861
```

마지막으로, checker 노드를 실행합니다. goal_total_sum을 50으로 설정하고, 목표가 달성되면 결과를 받습니다.

```
makepluscode@worktop:~$ ros2 run topic_service_action_rclcpp_example checker
goal_total_sum : 50
[INFO] [1687005952.524448841] [checker]: Action goal accepted.
[INFO] [1687005952.524721878] [checker]: Action succeeded!
[INFO] [1687005952.524748168] [checker]: Action result(all formula):
[INFO] [1687005952.524766028] [checker]:
[INFO] [1687005952.524781538] [checker]: Action result(total sum):
[INFO] [1687005952.524795977] [checker]:        0.00
```

아래 명령어를 사용하면 한 번에 실행할 수 있습니다.

```
makepluscode@worktop:~$ ros2 launch topic_service_action_rclcpp_example arithmetic.launch.py
```

13장 C++ 프로그래밍

토픽이란

토픽은 비동기식 단방향 메시지 송수신 방식으로, 퍼블리셔와 서브스크라이버 간의 통신입니다. 이는 ROS에서 가장 많이 사용되는 통신 방법입니다.

QoS 설정

```
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_num").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_volatile();

  arithmetic_argument_publisher_ =
    this->create_publisher<ArithmeticArgument>("arithmetic_argument", QOS_RKL10V);

  timer_ =
    this->create_wall_timer(1s, std::bind(&Argument::publish_random_arithmetic_arguments, this));
}
```

1. Node 클래스를 상속받아 Argument 클래스를 작성합니다.
2. QOS_RKL10V 변수에 QoS 인스턴스를 할당합니다.
    a. keepLast(1)
    b. reliable()
    c. durability_volatile()
3. create_publisher 생성시 QOS_RKL10V 변수로 QoS 를 지정합니다.

토픽 퍼플리시

```
void Argument::publish_random_arithmetic_arguments()
{
  std::random_device rd;
  std::mt19937 gen(rd());
  std::uniform_real_distribution<float> distribution(min_random_num_, max_random_num_);

  msg_srv_action_interface_example::msg::ArithmeticArgument 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);
}
```

1. publish_random_arithmetic_arguments 은 프레임워크에 의해 주기적으로 호출 됩니다.
2. ArithmeticArgument 형식의 msg 를 생성합니다.
3. 2개의 랜덤 변수를 생성 하고, publish 함수를 통해 송신 합니다.

```
arithmetic_argument_subscriber_ = this->create_subscription<ArithmeticArgument>(
    "arithmetic_argument",
    QOS_RKL10V,
    [this](const ArithmeticArgument::SharedPtr msg) -> void
    {
      argument_a_ = msg->argument_a;
      argument_b_ = msg->argument_b;

      RCLCPP_INFO(
        this->get_logger(),
        "Timestamp of the message: sec %ld nanosec %ld",
        msg->stamp.sec,
        msg->stamp.nanosec);

      RCLCPP_INFO(this->get_logger(), "Subscribed argument a: %.2f", argument_a_);
      RCLCPP_INFO(this->get_logger(), "Subscribed argument b: %.2f", argument_b_);
    }
  );
```

1. arithmetic_argument_subscriber_ 이름의 서브스크라이버 객체를 생성합니다.
2. msg 를 송신하면 callback 될 함수를 작성합니다.
3. callback 함수 내에서 msg 를 수신하여 결과를 출력 합니다.

14장 서비스 프로그래밍 (C++)

서비스

서비스(Service)는 양방향 메시지 송수신 방식입니다. 클라이언트가 특정 요청을 하고, 서버가 해당 요청을 수행하여 결과를 반환하는 통신입니다.

서비스 서버

```
auto get_arithmetic_operator =
    [this](
    const std::shared_ptr<ArithmeticOperator::Request> request,
    std::shared_ptr<ArithmeticOperator::Response> response) -> void
    {
      argument_operator_ = request->arithmetic_operator;
      argument_result_ =
        this->calculate_given_formula(argument_a_, argument_b_, argument_operator_);
      response->arithmetic_result = argument_result_;

      std::ostringstream oss;
      oss << std::to_string(argument_a_) << ' ' <<
        operator_[argument_operator_ - 1] << ' ' <<
        std::to_string(argument_b_) << " = " <<
        argument_result_ << std::endl;
      argument_formula_ = oss.str();

      RCLCPP_INFO(this->get_logger(), "%s", argument_formula_.c_str());
    };
```

1. Calculator 클래스에서 create_service를 통해 서비스 서버를 생성할 때, get_arithmetic_operator 함수가 전달됩니다.
2. get_arithmetic_operator 는 수신한 operator 와 2개의 숫자로 사칙 연산을 수행합니다.
3. 결과를 argument_result 에 저장하고, console 에 출력합니다.

서비스 클라이언트

```
Operator::Operator(const rclcpp::NodeOptions & node_options)
: Node("operator", node_options)
{
  arithmetic_service_client_ = this->create_client<ArithmeticOperator>("arithmetic_operator");
  while (!arithmetic_service_client_->wait_for_service(1s)) {
    if (!rclcpp::ok()) {
      RCLCPP_ERROR(this->get_logger(), "Interrupted while waiting for the service.");
      return;
    }
    RCLCPP_INFO(this->get_logger(), "Service not available, waiting again...");
  }
}
```

1. Operator 는 Node 를 상속합니다.
2. Node 클래스를 통해 create_client 로 arithmetic_service_client_ 인스턴스를 생성합니다.

```
void Operator::send_request()
{
  auto request = std::make_shared<ArithmeticOperator::Request>();
  std::random_device rd;
  std::mt19937 gen(rd());
  std::uniform_int_distribution<int> distribution(1, 4);
  request->arithmetic_operator = distribution(gen);

  using ServiceResponseFuture = rclcpp::Client<ArithmeticOperator>::SharedFuture;
  auto response_received_callback = [this](ServiceResponseFuture future) {
      auto response = future.get();
      RCLCPP_INFO(this->get_logger(), "Result %.2f", response->arithmetic_result);
      return;
    };

  auto future_result =
    arithmetic_service_client_->async_send_request(request, response_received_callback);
}
```

1. main()에서 send_request를 호출합니다.
2. send_request는 arithmetic_service_client_를 통해 서비스를 요청합니다.
3. response_received_callback에서 서비스의 결과를 받아 출력합니다.

15장 액션 프로그래밍 (C++)

`액션`은 비동기식+동기식 양방향 송수신 방식으로 `액션목표`를 지정하는 `액션클라이언트`와 `액션목표`를 받아 특정 태스크를 수행하면서 중간 결과값을 전송하는 `액션패드백`, `액션결과`를 전송하는 `액션서버`간의 통신 입니다.

액션서버 코드

```
arithmetic_action_server_ = rclcpp_action::create_server<ArithmeticChecker>(
    this->get_node_base_interface(),
    this->get_node_clock_interface(),
    this->get_node_logging_interface(),
    this->get_node_waitables_interface(),
    "arithmetic_checker",
    std::bind(&Calculator::handle_goal, this, _1, _2),
    std::bind(&Calculator::handle_cancel, this, _1),
    std::bind(&Calculator::execute_checker, this, _1)
  );
```

1. arithmetic_action_server_ 는 `액션서버`를 위한 인스턴스 입니다.

```
rclcpp_action::GoalResponse Calculator::handle_goal(
  const rclcpp_action::GoalUUID & uuid,
  std::shared_ptr<const ArithmeticChecker::Goal> goal)
{
  (void)uuid;
  (void)goal;
  return rclcpp_action::GoalResponse::ACCEPT_AND_EXECUTE;
}
```

1. handle_goal 은 `액션클라이언트`에서 `액션목표`를 요청 했을때 호출되는 함수 입니다.

```
rclcpp_action::CancelResponse Calculator::handle_cancel(
  const std::shared_ptr<GoalHandleArithmeticChecker> goal_handle)
{
  RCLCPP_INFO(this->get_logger(), "Received request to cancel goal");
  (void)goal_handle;
  return rclcpp_action::CancelResponse::ACCEPT;
}
```

1. handle_cancel 은 `액션클라이언트`에서 `액션취소`를 요청 했을때 호출되는 함수 입니다.

```
void Calculator::execute_checker(const std::shared_ptr<GoalHandleArithmeticChecker> goal_handle)
{
  RCLCPP_INFO(this->get_logger(), "Execute arithmetic_checker action!");
  rclcpp::Rate loop_rate(1);

  auto feedback_msg = std::make_shared<ArithmeticChecker::Feedback>();
  float total_sum = 0.0;
  float goal_sum = goal_handle->get_goal()->goal_sum;

  while ((total_sum < goal_sum) && rclcpp::ok()) {
    total_sum += argument_result_;
    feedback_msg->formula.push_back(argument_formula_);
    if (argument_formula_.empty()) {
      RCLCPP_WARN(this->get_logger(), "Please check your formula");
      break;
    }
    RCLCPP_INFO(this->get_logger(), "Feedback: ");
    for (const auto & formula : feedback_msg->formula) {
      RCLCPP_INFO(this->get_logger(), "\t%s", formula.c_str());
    }
    goal_handle->publish_feedback(feedback_msg);
    loop_rate.sleep();
  }

  if (rclcpp::ok()) {
    auto result = std::make_shared<ArithmeticChecker::Result>();
    result->all_formula = feedback_msg->formula;
    result->total_sum = total_sum;
    goal_handle->succeed(result);
  }
}
```

1. execute_checker 는 `액션클라이언트`에서 요청한 `액션목표`를 가지고 실제 테스크가 진행되는 함수 입니다.
2. get_goal 함수를 통해 `액션목표`를 goal_sum 에 저장합니다.
3. total_sum 에 argument_result_ 를 누적합니다.
4. total_sum 과 goal_sum 를 비교해서 목표값에 도달할때 까지 반복합니다.

액션클라이언트 코드

```
Checker::Checker(float goal_sum, const rclcpp::NodeOptions & node_options)
: Node("checker", node_options)
{
  arithmetic_action_client_ = rclcpp_action::create_client<ArithmeticChecker>(
    this->get_node_base_interface(),
    this->get_node_graph_interface(),
    this->get_node_logging_interface(),
    this->get_node_waitables_interface(),
    "arithmetic_checker");

  send_goal_total_sum(goal_sum);
}
```

1. Node 를 상속받아 Checker 클래스를 정의합니다.
2. create_client 로 arithmetic_action_client_ 를 생성합니다.
3. send_goal_total_sum 함수를 통해 `액션목표`를 `액션서버`에게 전송합니다.

```
void Checker::send_goal_total_sum(float goal_sum)
{
  using namespace std::placeholders;
```

```
  if (!this->arithmetic_action_client_) {
    RCLCPP_WARN(this->get_logger(), "Action client not initialized");
  }

  if (!this->arithmetic_action_client_->wait_for_action_server(std::chrono::seconds(10))) {
    RCLCPP_WARN(this->get_logger(), "Arithmetic action server is not available.");
    return;
  }

  auto goal_msg = ArithmeticChecker::Goal();
  goal_msg.goal_sum = goal_sum;

  auto send_goal_options = rclcpp_action::Client<ArithmeticChecker>::SendGoalOptions();
  send_goal_options.goal_response_callback =
    std::bind(&Checker::get_arithmetic_action_goal, this, _1);
  send_goal_options.feedback_callback =
    std::bind(&Checker::get_arithmetic_action_feedback, this, _1, _2);
  send_goal_options.result_callback =
    std::bind(&Checker::get_arithmetic_action_result, this, _1);
  this->arithmetic_action_client_->async_send_goal(goal_msg, send_goal_options);
}
```

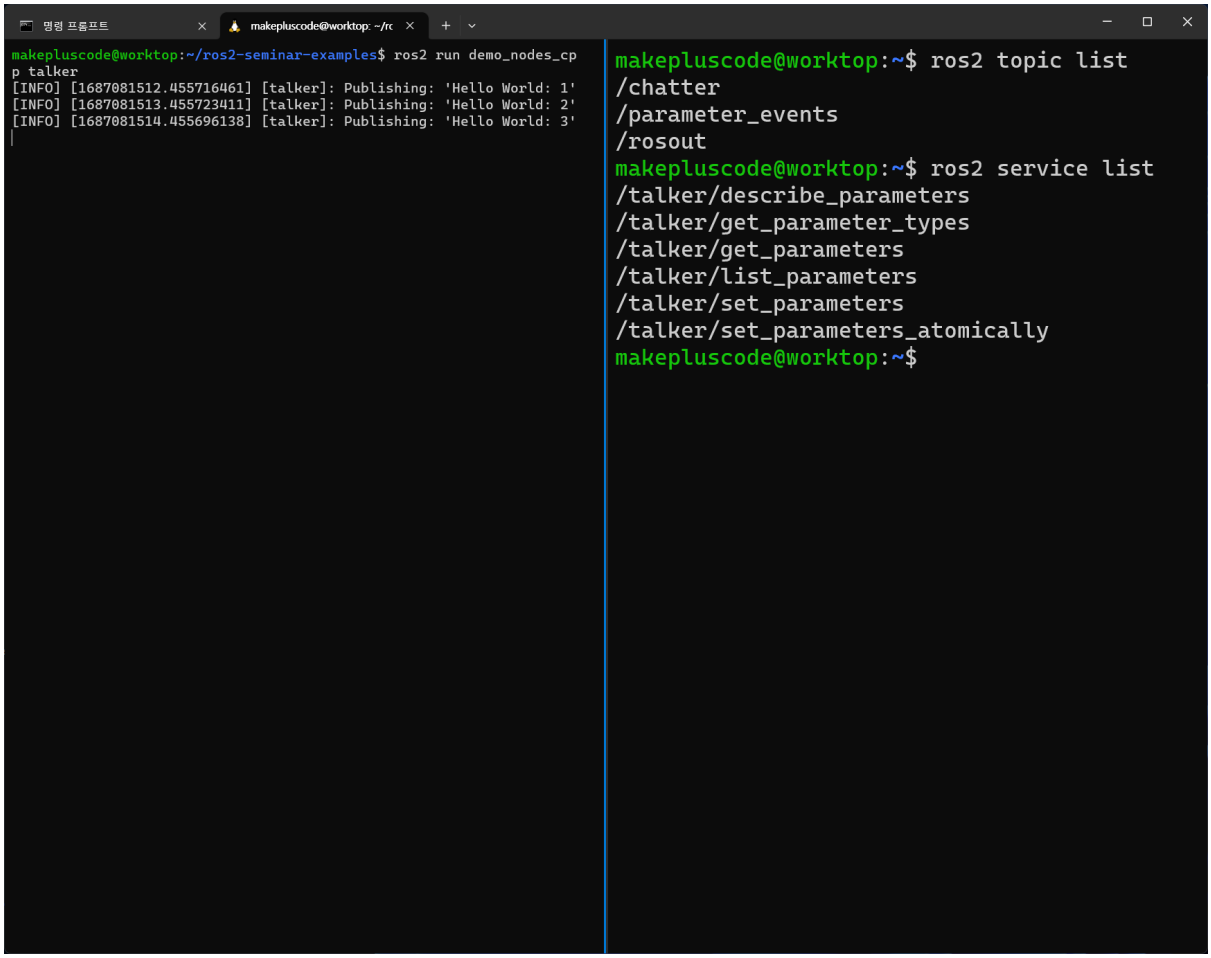
send_goal_total_sum 는 `액션목표`를 `액션서버`에게 전송합니다.

1. wait_for_action_server 를 이용하여 10초동안 `액션서버` 가 연결되길 기다립니다.
2. `액션서버` 가 연결 되면 SendGoalOptions 를 통해 `액션목표` 를 전송합니다.
3. goal_response, action_feedback, action_result 를 사전 정의된 함수에서 callback 될 수 있도록 설정합니다.

16장 파라미터 프로그래밍 (C++)

ROS2 노드는 모두 파라미터 서버를 가지고 있습니다. 파라미터 클라이언트를 통해 파라미터에 접근할 수 있습니다. 만약 서비스가 특정 작업을 위한 원격 프로시저 호출(RPC)에 가까운 목적이라면, 파라미터는 외부에서 특정 파라미터를 쉽게 설정하고 가져올 수 있도록 하는 기능입니다.

demo_node_cpp 패키지의 talker 를 실행합니다.



1. topic list 에 parameter_events 라는 토픽이 있습니다. 이를 서브스크라이브하여 노드는 런타임에서 생성, 변경, 삭제되는 파라미터를 확인할 수 있습니다.
2. service list에는 코드상에 구현되지 않은 파라미터와 관련된 서비스가 있습니다. 이는 Node를 상속받았기 때문에 생성된 서비스입니다.

파라미터 클라이언트

```
/**: # namespace and node name
  ros__parameters:
    qos_depth: 30
    min_random_num: 0.0
    max_random_num: 9.0
```

1. arithmetic_config.yaml 파일에 다음의 파라미터가 정의됩니다.

```
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_num").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_volatile();

  arithmetic_argument_publisher_ =
    this->create_publisher<ArithmeticArgument>("arithmetic_argument", QOS_RKL10V);

  timer_ =
    this->create_wall_timer(1s, std::bind(&Argument::publish_random_arithmetic_arguments, this));
}
```

1. get_parameter 함수를 통해서 해당 노드의 파라미터를 가지고 올 수 있습니다.

```
void Argument::update_parameter()
{
  parameters_client_ = std::make_shared<rclcpp::AsyncParametersClient>(this);
  while (!parameters_client_->wait_for_service(1s)) {
    if (!rclcpp::ok()) {
      RCLCPP_ERROR(this->get_logger(), "Interrupted while waiting for the service. Exiting.");
      return;
    }
    RCLCPP_INFO(this->get_logger(), "service not available, waiting again...");
  }

  auto param_event_callback =
    [this](const rcl_interfaces::msg::ParameterEvent::SharedPtr event) -> void
    {
      for (auto & changed_parameter : event->changed_parameters) {
        if (changed_parameter.name == "min_random_num") {
          auto value = rclcpp::Parameter::from_parameter_msg(changed_parameter).as_double();
          min_random_num_ = value;
        } else if (changed_parameter.name == "max_random_num") {
          auto value = rclcpp::Parameter::from_parameter_msg(changed_parameter).as_double();
          max_random_num_ = value;
        }
      }
    };

  parameter_event_sub_ = parameters_client_->on_parameter_event(param_event_callback);
}
```

1. update_parameter 함수에서는 AsyncParametersClient 의 포인터를 생성 합니다.
2. 파라미터가 업데이트 될 때, param_event_callback 이 호출되어 클래스 맴버 변수에 파라미터 값을 갱신 합니다.

17장 실행인자 프로그래밍 (C++)

C++ 프로그램을 실행하면, 가장 먼저 호출되는 main 함수는 두 개의 매개변수를 가집니다. 첫 번째 매개변수인 argc는 인자의 개수를 나타냅니다. 두 번째 매개변수인 argv는 문자열, 포인터, 배열 타입으로 넘겨받은 인자들을 저장합니다.

ROS2 실행 인자는 크게 2가지로 분류됩니다. 첫째는 `-ros-args`가 붙은 인자들로 ROS2 API와 관련된 옵션을 변경할 수 있습니다. 둘째는 `-ros-args`가 붙지 않은 인자들로 사용자 정의 인자입니다.

예를 들어, checker를 실행할 때 다음과 같은 인자를 전달할 수 있습니다.

```
$ ros2 topic_service_action_rclcpp_example checker -g 100 --ros-args -r __ns:=/demo goal_total_sum : 100
```

“--ros-args -r __ns:=/demo”는 rclcpp::init 함수의 인자로 전달됩니다.

```
rclcpp::init(argc, argv);
```

rcutils_cli_option_exist 함수는 -h 옵션이 있는지 확인합니다. -h 옵션이 있을 경우 print_help를 통해 도움말을 출력하고 종료합니다.

```
if (rcutils_cli_option_exist(argv, argv + argc, "-h")) {
    print_help();
    return 0;
  }
```

18장 런치 프로그래밍 (파이썬, C++)

ROS2에서는 하나의 노드를 실행하기 위해 `ros2 run`을 사용합니다. ROS2에서는 런치(launch)를 사용하여 하나 이상의 노드를 실행할 수 있습니다. 노드를 실행할 때 인자, 노드 이름 변경, 네임스페이스 변경, 환경 변수 설정 변경 등의 옵션을 사용할 수 있습니다.

```
import os

from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch.actions import DeclareLaunchArgument
from launch.substitutions import LaunchConfiguration
from launch_ros.actions import Node


def generate_launch_description():
    param_dir = LaunchConfiguration(
        'param_dir',
        default=os.path.join(
            get_package_share_directory('topic_service_action_rclcpp_example'),
            'param',
            'arithmetic_config.yaml'))

    return LaunchDescription([
        DeclareLaunchArgument(
            'param_dir',
            default_value=param_dir,
            description='Full path of parameter file'),

        Node(
            package='topic_service_action_rclcpp_example',
            executable='argument',
            name='argument',
            parameters=[param_dir],
            output='screen'),

        Node(
            package='topic_service_action_rclcpp_example',
            executable='calculator',
            name='calculator',
            parameters=[param_dir],
            output='screen'),
    ])
```

1. generate_launch_description 함수는 LaunchConfiguration을 통해 실행 관련 설정을 초기화합니다.
2. 이 함수는 LaunchDescription를 생성하여 반환합니다.
3. LaunchDescription는 실행할 노드의 패키지명, 실행 파일명, 이름, 파라미터 등의 설정을 포함합니다.