ROS 2 스터디 9주차

구본우

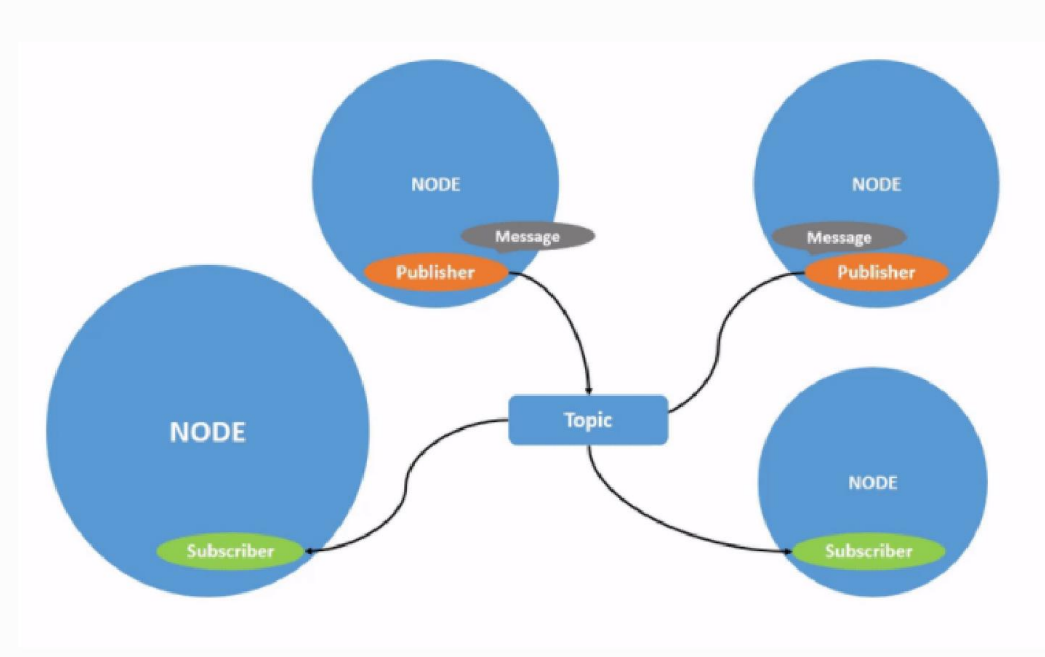
13장 토픽 프로그래밍 (C++)

- 13.1 토픽

 - 비동기식 단방향 메시지 송수신 방식으로 메시지를 퍼블리시하는 퍼블리셔와 메시지를 서브스크라이브하는 서브스크라이버 간의 통신이다.

 - 이는 1:1 통신을 기본으로 하지만 복수의 노드에서 하나의 토픽을 송수신하는 1:N도 가능하고 그 구성 방식에 따라 N:1, N:N 통신도 가능하다

 - ROS 메시지 통신에서 가장 널리 사용되는 통신 방법이다.



13장 토픽 프로그래밍 (C++)

- 토픽 퍼블리셔

1) Node 설정
2) QoS 설정
3) create_publisher 설정
4) 퍼블리시 함수 작성

- 토픽 서브스크라이버

1) Node 설정
2) QoS 설정
3) create_subscription 설정
4) 서브스크라이브 함수 설정

13장 토픽 프로그래밍 (C++)

노드 설정

```
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_;
};
```

13장 토픽 프로그래밍 (C++)

QoS 설정

```cpp
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));
}
```

13장 토픽 프로그래밍 (C++)

토픽 메시지 통신에 사용하는 msg 인터페이스

```cpp
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);
}
```

13장 토픽 프로그래밍 (C++)

퍼블리시한 램덤
숫자와 시간을
받아오고 CLI에 출력

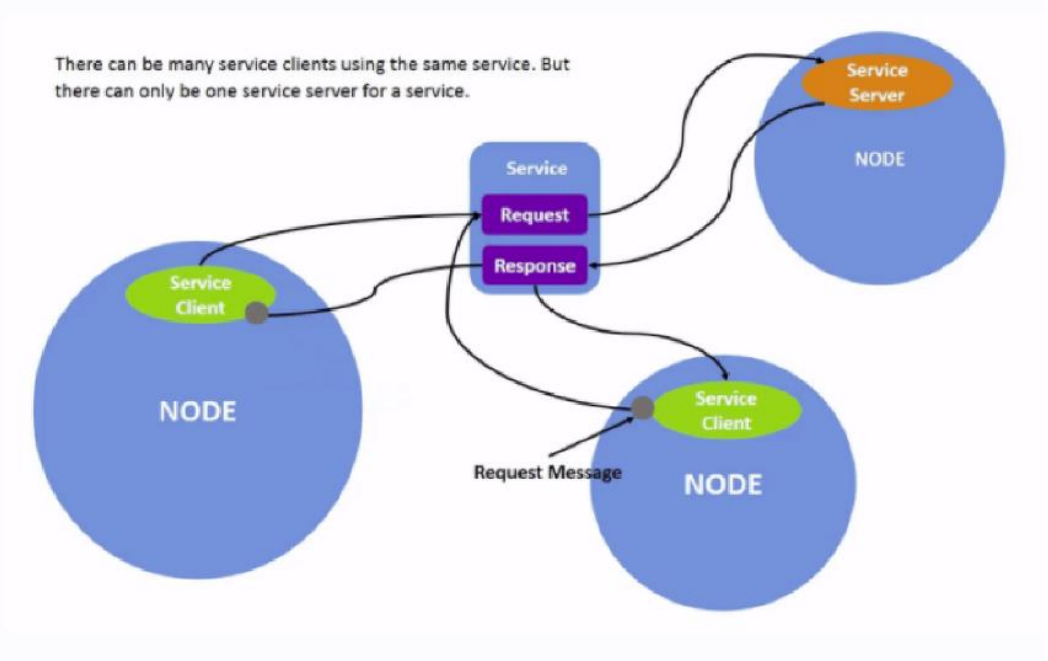
```
const auto QOS_RKL10V =
  rclcpp::QoS(rclcpp::KeepLast(qos_depth)).reliable().durability_volatile();

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(),
      "Subscribed at: 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_);
  }
);
```

14장 서비스 프로그래밍 (C++)

- 14.1 서비스

 - 동기식 양방향 메시지 송수신 방식으로 서비스의 **요청(Request)**을 하는 쪽을 서비스 클라이언트(Client)라고 하며 요청받은 서비스를 수행한 후 서비스의 **응답(Response)**을 하는 쪽을 서비스 서버(Server)라고 한다.

 - 서비스는 특정 요청을 하는 클라이언트 단과 요청 받은 일을 수행한 후에 결과값을 전달하는 서버 단과의 통신이다.



14장 서비스 프로그래밍 (C++)

- 서비스 서버

1) Node 설정
2) create_server 설정
3) 콜백함수 작성

- 서비스 클라이언트

1) Node 설정
2) create_client 설정
3) 요청함수 설정

14장 서비스 프로그래밍 (C++)

콜백함수 설정

(request와 response 인자를 이용한다.
미리 작성해둔 함수에 인자를 전달하고 그 결과를 리턴 받는다.

```
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());
  };

arithmetic_argument_server_ =
  create_service<ArithmeticOperator>("arithmetic_operator", get_arithmetic_operator);
```

14장 서비스 프로그래밍 (C++)

노드 설정

(서비스 요청을 위한 send_request 함수가 있다.)

```
class Operator : public rclcpp::Node
{
public:
  using ArithmeticOperator = msg_srv_action_interface_example::srv::ArithmeticOperator;

  explicit Operator(const rclcpp::NodeOptions & node_options = rclcpp::NodeOptions());
  virtual ~Operator();

  void send_request();

private:
  rclcpp::Client<ArithmeticOperator>::SharedPtr arithmetic_service_client_;
};
```

14장 서비스 프로그래밍 (C++)

create_client 설정

(서비스명을 인자로 받아 rclcpp::Client를 실체와 시켜 준다.)

```
class Operator : public rclcpp::Node
{
public:
  using ArithmeticOperator = msg_srv_action_interface_example::srv::ArithmeticOperator;

  explicit Operator(const rclcpp::NodeOptions & node_options = rclcpp::NodeOptions());
  virtual ~Operator();

  void send_request();

private:
  rclcpp::Client<ArithmeticOperator>::SharedPtr arithmetic_service_client_;
};
```

14장 서비스 프로그래밍 (C++)

요청함수 설정

(request와 response 콜백함수를 이용하여 로그로 출력한다.

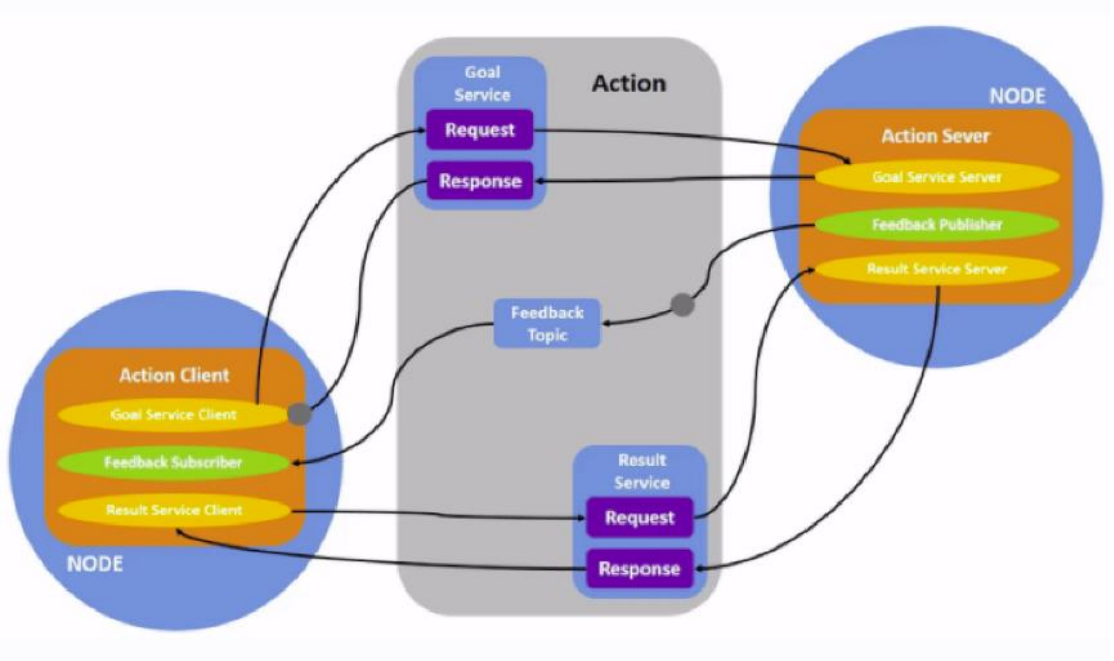
Async_send_request 함수를 통해 비동기식으로 서비스 요청을 보낸다.

```
void Operator::send_request()
{
  std::random_device rd;
  std::mt19937 gen(rd());
  std::uniform_int_distribution<int> distribution(1, 4);

  auto request = std::make_shared<ArithmeticOperator::Request>();
  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);
}
```

15장 액션 프로그래밍 (C++)

- 15.1 액션

 - 비동기식 + 동기식 양방향 메시지 송수신 방식으로 **액션 목표(Goal)**를 지정하는 액션 클라이언트와 액션 목표를 받아 특정 태스크를 수행하면서 중간 결과값을 전송하는 **액션 피드백(feedback)** 그리고 최종 결과값에 해당되는 **액션 결과(Action result)**를 전송하는 액션 서버 간의 통신이다.



15장 액션 프로그래밍 (C++)

- 액션 서버

1) Node 설정
2) create_server 설정
3) goal, cancel, accepted 콜백함수 작성

- 액션 클라이언트

1) Node 설정
2) create_client 설정
3) goal_response, feedback, result 콜백함수 설정

15장 액션 프로그래밍 (C++)

토픽과 서비스 통신을 위한 멤버 변수들은 rclcpp 네임스페이스를 가지는 반면에 액션 통신을 위한 멤버 변수들은 rclcpp_action 네임스페이스를 가짐

```
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)
);
```

15장 액션 프로그래밍 (C++)

다양한 콜백 함수 설정

```cpp
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;
}
```

```cpp
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;
}
```

15장 액션 프로그래밍 (C++)

노드 설정

```
class Checker : public rclcpp::Node
{
public:
  using ArithmeticChecker = msg_srv_action_interface_example::action::ArithmeticChecker;
  using GoalHandleArithmeticChecker = rclcpp_action::ClientGoalHandle<ArithmeticChecker>;

  explicit Checker(
    float goal_sum,
    const rclcpp::NodeOptions & node_options = rclcpp::NodeOptions());
  virtual ~Checker();

private:
  void send_goal_total_sum(float goal_sum);

  void get_arithmetic_action_goal(
    std::shared_future<rclcpp_action::ClientGoalHandle<ArithmeticChecker>::SharedPtr> future);

  void get_arithmetic_action_feedback(
    GoalHandleArithmeticChecker::SharedPtr,
    const std::shared_ptr<const ArithmeticChecker::Feedback> feedback);

  void get_arithmetic_action_result(
    const GoalHandleArithmeticChecker::WrappedResult & result);

  rclcpp_action::Client<ArithmeticChecker>::SharedPtr arithmetic_action_client_;
};
```

15장 액션 프로그래밍 (C++)

Create client 설정

```cpp
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);
}
```

15장 액션 프로그래밍 (C++)

goal_response_callback,
feedback_callback,
result_callback
함수를 초기화 시켜줌

```cpp
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);
}
```

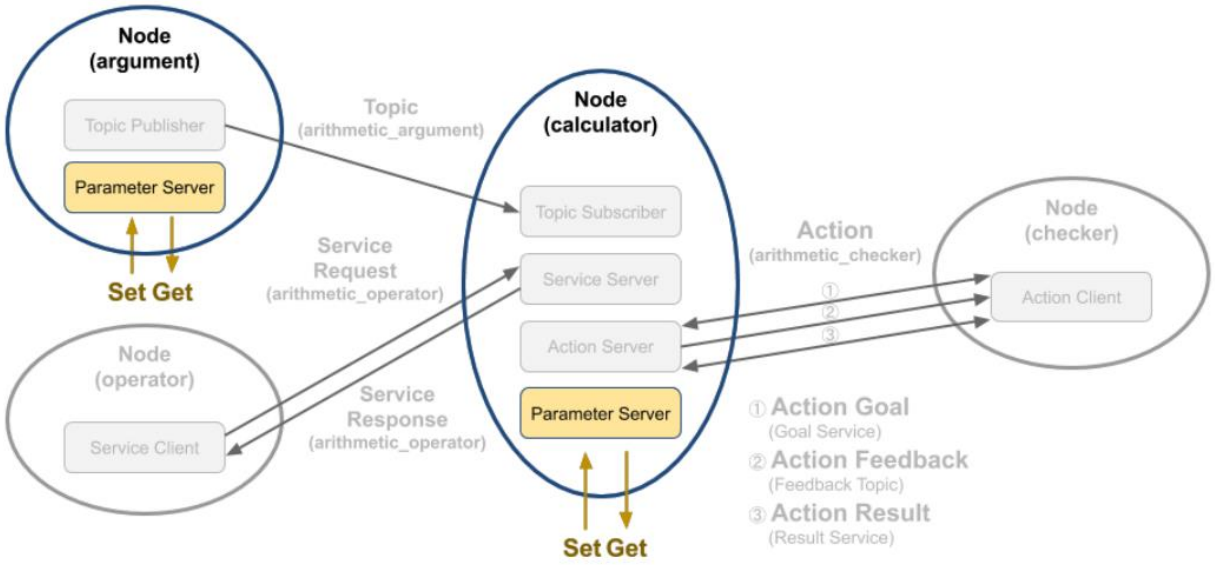
16장 파라미터 프로그래밍 (C++)

- 16.1 파라미터

 - ROS 1의 parameter server와 dynamic_reconfigure 패키지의 기능을 모두 가지고 있어 노드가 동작하는 동안 특정 값에 대한 저장, 변경, 회수가 가능하다.

 - ROS 2의 모든 노드는 파라미터 서버를 가지고 있어 파라미터 클라이언트와 서비스 통신을 통해 파라미터에 접근할 수 있도록 구현되어 있다.

 - 파라미터는 특정 매개변수를 노드 내부 또는 외부에서 쉽게 저장하거나 변경할 수 있고, 쉽게 회수하여 사용할 수 있는 점이 서비스와 구분된다.



16장 파라미터 프로그래밍 (C++)

- 파라미터 서버

1) parameter.yaml 설정
2) launch 설정

- 파라미터 클라이언트

1) declare_parameter 함수로
   사용할 파라미터 등록
2) get_parameter 함수로
   파라미터 값 회수
3) parameter_event
   콜백함수 설정

16장 파라미터 프로그래밍 (C++)

parameter/.yaml 설정

Launch 파일 설정

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

```python
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'),
```

16장 파라미터 프로그래밍 (C++)

declare_parameter
함수로 사용할
파라미터 등록

get_parameter
함수로 파라미터 값
회수

```
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();
```

16장 파라미터 프로그래밍 (C++)

parameter_event
콜백함수 설정

```cpp
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);
}
```

17장 실행 인자 프로그래밍 (C++)

- 17.1 실행 인자

▪ C++ 프로그램 실행 시 가장 먼저 호출되는 main 함수는 두 개의 매개변수를 가진다.
- argc(argument count) : 넘겨받은 인자들의 개수를 저장
- argv(argument vector) : 문자열, 포인터, 배열 타입으로 넘겨받은 인자들을 저장

▪ ROS 2에서 실행 인자는 크게 두 가지로 분류된다.
- --ros-args가 붙은 인자: ROS 2 API와 관련된 옵션(remapping, paramete등)을 변경할 수 있다.
- --ros-args가 붙지 않은 인자: 일반적으로 사용하는 사용자 정의 실행 인자라고 생각하면 된다.

17장 실행 인자 프로그래밍 (C++)

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

1. '-h'인자가 있는지 검사한다.
2. 만약 '-h'가 검출되면 print_help 함수를 출력하고 main 함수를 빠져나간다.
3. 이를 통해 사용자가 해당 노드를 처음 사용하게 될 때 필요한 정보를 제공해준다.

17장 실행 인자 프로그래밍 (C++)

```
float goal_total_sum = 50.0;
char * cli_option = rcutils_cli_get_option(argv, argv + argc, "-g");
if (nullptr != cli_option) {
  goal_total_sum = std::stof(cli_option);
}
printf("goal_total_sum : %2.f\n", goal_total_sum);

auto checker = std::make_shared<Checker>(goal_total_sum);
```

rcutils_cli_get_option 함수는 실행 인자를 확인하고 그 값을 문자열 포인터로 반환해주는 역할을 한다. 사용자는 쉽게 여러 개의 실행 인자를 파싱할 수 있고, 문자열 포인터를 원하는 변수 타입으로 변경하여 노드의 생성 인자로 넘겨줄 수 있다.

18장 런치 프로그래밍 (C++)

- ROS2 단일 노드 실행 : “ros2 run package_name executable_name”
- ROS2 복수 노드 실행 : “ros2 launch package_name launch_file_name”
  - 개발한 패키지 + 공개 패키지 동시 실행
  - 각 노드별 여러 개의 파라미터 변경 적용

- ROS2 Launch
  - 패키지 매개변수, 노드 이름 변겨, 노드 네임스페이스 설정, 환경 변수 변경의 옵션 가능
  - Type : Python(ROS2 활용도 높음), XML(ROS1 계승), YAML(새로운 방식)

18장 런치 프로그래밍 (C++)

런치 프로그래밍의
기본 매소드

```
def generate_launch_description():

    xxx = LaunchConfiguration(yyy)

    return LaunchDescription([
        DeclareLaunchArgument(aaa),
        Node(bbb),
        Node(ccc),
    ])
```

18장 런치 프로그래밍 (C++)

Remappings 기능

(내부 코드 변경없이 토픽, 서비스, 액션 등의 고유 이름을 변경할 수 있는 기능)

```
Node(
    package='topic_service_action_rclpy_example',
    executable='argument',
    name='argument',
    remappings=[
        ('/arithmetic_argument', '/argument'),
    ]
```

18장 런치 프로그래밍 (C++)

RCLCPP 패키지 계열 빌드

(C++ 언어를 사용하는 경우 빌드 설정 파일(CMakeLists.txt)의 install 구문에 launch 라는 폴더명만 기재하면 된다.)

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