Week9

토픽 프로그래밍을 C++로 해보자

rclcpp 헤더파일을 보자

토픽 퍼블리셔 코드(C++)

```
#include <chrono>
#include <memory>
#include <string>
#include <utility>

#include "rclcpp/rclcpp.hpp"

#include "msg_srv_action_interface_example/msg/arithmetic_argument.hpp"
```

rclcpp의 Node 클래스를 상속받는 Argument 클래스에 대해 알아보자. Argument 클래스의 생성자는 rclcpp의 NodeOptions 객체를 인자로 받는다. 이를 통해서는 context, arguments, intra-process communication, parameter, allocator와 같은 Node 생성을 위한 다양한 옵션을 정할 수 있다.

Arguments 클래스에는 스마트 포인터 타입의 멤버 변수 rclcpp::Publisher와 rclcpp::TimerBase가 선언되어 있으며 토픽 메시지에 담을 랜덤 변수의 샘플링 변위를 정해줄 멤버 변수도 함께 확인할 수 있다.

```
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_
```

이제 cpp 파일을 보자.

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

#include "rclcpp/rclcpp.hpp"
#include "rcutils/cmdline_parser.h"

#include "arithmetic/argument.hpp"
```

Argument 클래스 생성자에서 부모 클래스인 rclcpp::Node 에 대한 선언을 알아보자. 첫 번째 매개변수에는 노드의 이름을 적고, 두 번째 매개변수에는 NodeOptions 객체를 명시한다. 첫 번째 매개변수에는 메시지 통신에 사용될 토픽명을 적어준다. timer의 경우 1초당 한 번씩 publish_random_arithmetic_arguments 멤버 함수를 호출하도록 설정하였다.’

```
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));
}
```

다음으로 publish_random_arithmetic_arguments 멤버 함수에 대해 알아보자. 해당 함수는 timer에 의해 1초당 한 번씩 호출된다. 토픽 메시지 통신에서 사용할 인터페이스(msg)를 선언하고 멤버 변수들을 올바르게 채워주고 난 뒤, Argument 클래스에서 초기화한 publisher를 통해 해당 인터페이스를 publish 함수로 퍼블리시한다. 마지막으로 인터페이스를 통해 퍼블리시한 랜덤한 숫자를 로그로 표시하는 코드를 확인할 수 있다.

```
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);
}
```

13.3 토픽 서브스크라이버 코드

Calculator 클래스는 토픽 퍼블리셔 노드와 마찬가지로 rclcpp::Node를 상속받고 있으며 생성자에서 calculator 노드로 초기화되었다.

subscriber는 rclcpp::Node 의 멤버 함수 create_subscription을 통해 초기화되며 해당 함수는 토픽명과 QoS 그리고 콜백 함수 매개변수로 구성되어 있다. 토픽명과 QoS는 Argument 클래스에서 입력했던 것과 동일하게 입력해주었고, 콜백함수는 std::bind 함수를 사용하지 않고 람다(Lambda) 표현식을 이용하였다.

콜백함수는 매개변수를 통해 서브스크라이브한 토픽 메시지에 접근하여 멤버 변수에 저장하고 있다.
ㅜ

```
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_);
    }
  );
```

13.4 노드 실행 코드

Subscribe 노드인 calculator와 Publisher 노드인 argument 노드가 있는데, 이 두 노드를 실행할 수 있도록 설정한 부분은 빌드설정파일(CmakeLists.txt)에서 확인할 수 있다.

CMake의 add_executable 명령어는 main 함수가 포함된 소스파일을 실행 가능하도록 만들어준다. 그 첫 번째 매개변수에는 실행명을 넣고, 다음 매개변수에는 main 함수가 포함된 파일의 이름을 넣어준다. 필요하다면 그 다음 매개변수로 main함수의 의존 라이브러리가 담긴 파일의 이름을 넣어준다. 이를 통해 ros2 run 명령어를 이용하여 두 노드를 실행할 수 있다.

```
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
)
```

argument.cpp 파일의 main 함수를 보자. cmdline_parser 라이브러리의 함수를 사용해 확인한 인자 중

“-h”가 있다면 print_help() 함수를 호출하고 main 함수를 빠져나가는 코드를 먼저 확인할 수 있다.

rclcpp::init 함수를 통해 namespace, remap 등을 포함하는 Ros2 argument를 전달할 수 있고, 이를 토대로 프로그램을 초기화한다. 그 다음 앞서 정의해 놓은 Argument 클래스를 객체화해서 노드를 생성한다.

rclcpp::spin 함수는 생성한 노드를 spin 시켜 선언된 콜백 함수가 호출될 수 있도록 한다.

```
int main(int argc, char * argv[])
{
  if (rcutils_cli_option_exist(argv, argv + argc, "-h")) {
    print_help();
    return 0;
  }

  rclcpp::init(argc, argv);

  auto calculator = std::make_shared<Calculator>();

  rclcpp::spin(calculator);

  rclcpp::shutdown();

  return 0;
}
```

14.3 서비스 클라이언트 코드

서비스 클라이언트 역할을 하는 operator 노드를 보자
헤더 파일에 선언된 Operator 클래스를 보면 rclcpp:Node클래스를 상속받는 자식 클래스이고, 생성자에서 rclcpp::NodeOptions를 매개변수로 가진다. 그리고 서비스 요청을 위한 send_request함수와 스마트 포인터 타입의 멤버 변수 rclcpp::Client를 가지고 있다. 생성자 내부를 보면 부모클래스인

rclcpp::Node를 노드명과 node_options 인자로 먼저 초기화하고 멤버 함수 create_client를 통해 서비스명을 인자로 받아 rclcpp::Clioent를 실체화시킨다.

```
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_;
};
#endif  // ARITHMETIC__OPERATOR_HPP_
```

실제로 요청 수행하는 send_request 멤버 함수를 보자. 랜덤으로 연산자가 지정될 수 있도록 1부터 4사이의 랜덤한 정수를 생성하여 request 변수에 저장하낟. 그리고 요청에 의한 응답이 왔을 때 불려질 response_rexeived_callback 콜백 함수를 람다 표현식을 이용하여 정의했다. 콜백 함수가 불리면 future의 get함수를 이용하여 response 값에 접근할 수 있다.

14.4 노드 실행 코드

calculator가 서비스 서버노드 / operator는 서비스 클라이언트 노드다.
operator 노드의 메인 함수는 Operator 클래스를 객체화하고 반복문으로 send_request를 호출한다.

```
int main(int argc, char * argv[])
{
  if (rcutils_cli_option_exist(argv, argv + argc, "-h")) {
    print_help();
    return 0;
  }

  rclcpp::init(argc, argv);

  auto operator_node = std::make_shared<Operator>();

  while (rclcpp::ok()) {
    rclcpp::spin_some(operator_node);
    operator_node->send_request();

    printf("Press Enter for next service call.\n");
    if (pull_trigger() == false) {
      rclcpp::shutdown();
      return 0;
    }
  }
}
```

calculator 노드는 토픽서브스크라이버, 서비스 서버, 액션 서버를 포함한다.

15. 액션 프로그래밍

Calculator 노드는 액션 서버역할이다.

handle_goal 함수는 액션 클라이언트에서 액션 목표를 요청했을 때 콜백되는 함수로, 해당 함수의 인자로 액션 클라이언트 메시지의 UUID와 목푯값을 확인할 수 있다.

```
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;
}
```

액션 클라이언트 코드

액션 클라이언트 역할을 하는 checker 노드

헤더 파일에 선언된 Checker 클래스는 rclcpp::Node 클래스를 상속받고 생성자에서 goal_sum변수와 rclcpp::NodeOptions를 인자로 받는다.

send_goal_total_sum 함수는 goal_sum변수를 인자로 받아 액션 목푯값을 보내주는 역할을 한다. 이를 위해 ArithmeticChecker 액션 인터페이스ㅣ goal 메시지를 불러와 함수의 인자로 받은 goal_sum 변수를 넘겨준다.

get_arithmetic_action_goal 함수는 앞서 설정한 sendgoaloptions의 goal_response_callback으로 앞서 설명했던 handle_goal함수와 연결되어 있다.

노드 실행코드

calculator가 액션 서버노드, checker는 액션 클라이언트 노드

```
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;
}
```

16. 파라미터 프로래밍(C++)

Ros2의 의 모든 노드는 파라미터 서버를 가지고 있어 파라터 클라이언트와 서비스 통신을 통해 파라미터에 접근할 수 있다.

argument 노드는 파라미터 클라이언트가 포함되어 있다.

Argument 클래스는 rclcpp::Node를 상속받는 해당클래스의 멤버 변수를 살펴보면 parameterEvent 타입으로 선언된 토픽 서브스크라이버와 AsyncParametersClient클래스의 스마트포인터를 확인한다.

update_parameter 함수에서는 AsyncParametersClient를 this 포인터로 초기화한다. 그리고 런타임에서 파라미터 서버에 이벤트가 있을 때 콜백되는 함수를 등록할수있고, 이를 통해 파라미터가 변경되었을 때를 확인할수있다.

17. 실행 인자 프로그래밍(C++)

C++ 프로그램 실행 시 가장 먼저 호출되는 main 함수는 두개의 매개변수를 가진다. 먼저 첫번째 매개변수인 argc는 argument count의 약자로 넘겨받은 인자들의 갯수를 담는다. 두 번재 매개변수인 argv는 argument vector의 약자로 문자열, 포인터, 배열 타입으로 넘겨받은 인자들을 저장하고 있다.

파라미터와 실행인자 중 언제 무엇을 사용하는게 ㅐ 좋을까?

먼저 변수가 런ㄴ타임에서 사용자 혹은 다른 프로그램에 의해 변경되어야 할 값인지 확인해보자. 만약 그렇다면 Ros2 파라미터 서버에 변수를 등록하고, 서비스 통신을 이용해 값을 변경해가며 프로그램의 옵션을 변경할 수 있게 한다. 만약 그렇지 않다면 실행인자를 사용하여 변수의 값을 초기화 하고 노드 생성 시 필요한 옵션을 설정할 수 있다.

18. 런치 프로그래밍(파이썬, C++)

Ros2 에서는 하나의 노드를 실행시키기 위해 ros2 run명령어를 사용한다. Ros2의 Launch는 하나 이상의 정해진 노드를 실행시킬 수 있다. 더불어 노드를 실행할 때 패키지의 매개변수나 노드 이름 변경, 노드 네임스페이스 설정등의 옵션을 사용할수있다.

```
#!/usr/bin/env python3
# Copyright 2021 OROCA
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

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'),
    ])
```

Launch 파일은 generate_launch_description 메소드를 이용해 LaunchConfiguration 클래슬ㄹ 이용하여 실행 관련 설정을 초기화하고, 리턴값으로 LaunchDescription 클래스를 반환한다.

패키지 빌드

Launch 파일은 파이썬 파일이기에 빌드 자체가 필요없지만 해당 파일을 ros2 에코 시스템 환경에서 사용하기 위해선 빌드를 통해 정해진 위치에 설치해여ㅑ 만한다.

RCLPY 패키지 계열

setup.py의 data_files 옵션 부분에 launch옵션을 지정한다.

```
#!/usr/bin/env python3

import glob
import os

from setuptools import find_packages
from setuptools import setup

package_name = 'topic_service_action_rclpy_example'
share_dir = 'share/' + package_name

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'))),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    author='Pyo, Darby Lim',
    author_email='passionvirus@gmail.com, routiful@gmail.com',
    maintainer='Pyo',
    maintainer_email='passionvirus@gmail.com',
    keywords=['ROS'],
    classifiers=[
        'Intended Audience :: Developers',
        'License :: OSI Approved :: Apache Software License',
        'Programming Language :: Python',
        'Topic :: Software Development',
    ],
    description='ROS 2 rclpy example package for the topic, service, action',
    license='Apache License, Version 2.0',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'argument = topic_service_action_rclpy_example.arithmetic.argument:main',
            'operator = topic_service_action_rclpy_example.arithmetic.operator:main',
            'calculator = topic_service_action_rclpy_example.calculator.main:main',
            'checker = topic_service_action_rclpy_example.checker.main:main',
        ],
    },
)
```

빌드

```
$ cbp topic_service_action_rclpy_example
```

Launch 실행

```
$ ros2 launch <package_name> <launch_file_name>
```