Week8

태그

내용

5장. 토픽, 서비스, 액션 인터페이스

ROS2에서 기본으로 제공하는 std_msgs, geometry_msgs, sensor_msgs 등이 아닌 사용자 인터페이스를 작성하는 방법을 알아봅시다. 사용자 인터페이스를 응용 프로그램 패키지에 넣을 수도 있지만, 별도의 인터페이스 패키지를 만들어 사용하는 것을 추천합니다.

인터페이스 패키지 만들기

```
ros2 pkg create --build-type ament_cmake msg_srv_action_interface_example
```

msg_srv_action_interface_example 디렉토리 아래에 action, msg, srv 디렉토리를 생성합니다. 그리고 각 디렉토리에 ArithmeticChecker.action, ArithmeticArgument.msg, ArithmeticOperator.srv 파일을 만들고 코드를 작성합니다.

```
~/robot_ws/src/msg_srv_action_interface_example$ tree
.
├── CMakeLists.txt
├── action
│   └── ArithmeticChecker.action
├── build.sh
├── msg
│   └── ArithmeticArgument.msg
├── package.xml
├── src
└── srv
    └── ArithmeticOperator.srv
```

ArithmeticChecker.action

```
# Goal
float32 goal_sum
---
# Result
string[] all_formula
float32 total_sum
---
# Feedback
string[] formula
```

ArithmeticArgument.msg

```
# Messages
builtin_interfaces/Time stamp
float32 arg_a
float32 arg_b
```

ArithmeticOperator.srv

```
# Constants
int8 PLUS = 1
int8 MINUS = 2
int8 MULTIPLY = 3
int8 DIVISION = 4
# Request
int8 arithmetic_operator
---
# Response
float32 arithmetic_result
```

패키지 설정파일

```
<?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>msg_srv_action_interface_example</name>
  <version>0.0.0</version>
  <description>TODO: Package description</description>
  <maintainer email="makepluscode@todo.todo">makepluscode</maintainer>
  <license>TODO: License declaration</license>

  <buildtool_depend>ament_cmake</buildtool_depend>
  <buildtool_depend>rosidl_default_generators</buildtool_depend>
  <exec_depend>builtin_interfaces</exec_depend>
  <exec_depend>rosidl_default_runtime</exec_depend>
  <member_of_group>rosidl_interface_packages</member_of_group>

  <export>
    <build_type>ament_cmake</build_type>
  </export>
</package>
```

빌드 설정 파일

```
cmake_minimum_required(VERSION 3.8)
project(msg_srv_action_interface_example)

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(builtin_interfaces REQUIRED)
find_package(rosidl_default_generators REQUIRED)

# interfaces
set(msg_files "msg/ArithmeticArgument.msg")
set(srv_files "srv/ArithmeticOperator.srv")
set(action_files "action/ArithmeticChecker.action")

rosidl_generate_interfaces(${PROJECT_NAME}
  ${msg_files}
  ${srv_files}
  ${action_files}
  DEPENDENCIES builtin_interfaces
)

ament_export_dependencies(rosidl_default_runtime)
ament_package()
```

빌드 하기

```
#!/bin/sh
```

```
colcon build --symlink-install --packages-select msg_srv_action_interface_example
```

6장. ROS2 패키지 설계 (파이썬)

이제까지 배운 토픽, 서비스, 액션을 파이썬으로 구현합니다.

예를 들어, 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 연산자를 서비스 요청 값으로 보냅니다.

4. calculator node

   변수 a와 b를 연산자를 이용하여 연산하고, operator node에게 서비스 응답값을 보냅니다. 연산 결과값을 누적하고, 액션 피드백으로 연산식을 보냅니다. 누적된 연산 결과값이 목표값보다 크면 연산 결과와 연산식을 보냅니다.

5. checker node

   연산 결과값의 누적 한계치를 액션 목표값으로 전달합니다.

노드작성

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

파이썬 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_rclpy_example</name>
  <version>0.6.0</version>
  <description>ROS 2 rclpy 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>
  <depend>rclpy</depend>
  <depend>std_msgs</depend>
  <depend>msg_srv_action_interface_example</depend>
  <test_depend>ament_copyright</test_depend>
  <test_depend>ament_flake8</test_depend>
  <test_depend>ament_pep257</test_depend>
  <test_depend>python3-pytest</test_depend>
  <export>
    <build_type>ament_python</build_type>
  </export>
</package>
```

1. depend 태그를 통해 종속패키지 rclpy, std_msgs 를 추가합니다.
2. build_type 은 ament_python 입니다.

파이썬 패키지 설정 파일 (setup.py)

topic_service_action_rclpy_example 패키지의 파이썬 패키지 설정 파일은 다음과 같습니다.

```
#!/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',
        ],
    },
)
```

1. `data_files` 는 패키지에서 사용되는 파일을 기입하여 같이 배포되기 위함
2. `entry_points` 는 명령줄에서 사용할 스크립트의 이름과 호출함수를 등록합니다. ros2 run 과 같은 실행 명령어를 통하여 각각의 노드를 실행할 예정이기에 다음과 같이 entry_points 를 추가 합니다.

소스코드 내려받기 및 빌드

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

cbp 를 통해서 실행하는 방법

```
cbp topic_service_action_rcl
```

가장 먼저 calculator 노드를 실행

```
$ ros2 run topic_service_action_rclpy_example calculator
```

토픽 퍼블리셔 실행

```
$ ros2 run topic_service_action_rclpy_example argument
```

서비스클라이언트 실행

```
$ ros2 run topic_service_action_rclpy_example operator
```

실제 연산식은 calculator 노드가 실행중인 터미널 창에서 확인할 수 있다.

액션 클라이언트 실행

```
$ ros2 run topic_service_action_rclpy_example checker
```

런치 파일 실행

```
$ ros2 launch topic_service_action_rclpy_example arithmeic.launch.py
```

7장. 토픽 프로그래밍 (파이썬)

ROS 토픽(Topic) 은 무엇인가?

- Node 간의 메시지를 주고 받는 통신의 하나의 방법이다.
- Topic 을 주는 쪽을 송신자 (퍼블리셔, Publisher), 받는 쪽을 수신자, 구독자 (서브스크라이버, Subscriber) 라고 한다.
- 1:1, 1:N, N:N 등, 다양한 네트워크를 구성할 수 있다.

토픽 퍼블리셔 코드

```
import random

from msg_srv_action_interface_example.msg import ArithmeticArgument
from rcl_interfaces.msg import SetParametersResult
import rclpy
from rclpy.node import Node
from rclpy.parameter import Parameter
from rclpy.qos import QoSDurabilityPolicy
from rclpy.qos import QoSHistoryPolicy
from rclpy.qos import QoSProfile
from rclpy.qos import QoSReliabilityPolicy


class Argument(Node):

    def __init__(self):
        super().__init__('argument')
        self.declare_parameter('qos_depth', 10)
        qos_depth = self.get_parameter('qos_depth').value
        self.declare_parameter('min_random_num', 0)
        self.min_random_num = self.get_parameter('min_random_num').value
        self.declare_parameter('max_random_num', 9)
        self.max_random_num = self.get_parameter('max_random_num').value
        self.add_on_set_parameters_callback(self.update_parameter)

        QOS_RKL10V = QoSProfile(
            reliability=QoSReliabilityPolicy.RELIABLE,
            history=QoSHistoryPolicy.KEEP_LAST,
            depth=qos_depth,
            durability=QoSDurabilityPolicy.VOLATILE)

        self.arithmetic_argument_publisher = self.create_publisher(
```

```
            ArithmeticArgument,
            'arithmetic_argument',
            QOS_RKL10V)

        self.timer = self.create_timer(1.0, self.publish_random_arithmetic_arguments)

    def publish_random_arithmetic_arguments(self):
        msg = ArithmeticArgument()
        msg.stamp = self.get_clock().now().to_msg()
        msg.argument_a = float(random.randint(self.min_random_num, self.max_random_num))
        msg.argument_b = float(random.randint(self.min_random_num, self.max_random_num))
        self.arithmetic_argument_publisher.publish(msg)
        self.get_logger().info('Published argument a: {0}'.format(msg.argument_a))
        self.get_logger().info('Published argument b: {0}'.format(msg.argument_b))

    def update_parameter(self, params):
        for param in params:
            if param.name == 'min_random_num' and param.type_ == Parameter.Type.INTEGER:
                self.min_random_num = param.value
            elif param.name == 'max_random_num' and param.type_ == Parameter.Type.INTEGER:
                self.max_random_num = param.value
        return SetParametersResult(successful=True)


def main(args=None):
    rclpy.init(args=args)
    try:
        argument = Argument()
        try:
            rclpy.spin(argument)
        except KeyboardInterrupt:
            argument.get_logger().info('Keyboard Interrupt (SIGINT)')
        finally:
            argument.destroy_node()
    finally:
        rclpy.shutdown()


if __name__ == '__main__':
    main()
```

1. Argument 클래스는 rclpy.node 모듈의 Node 클래스를 상속받았고, 생성자를 통해 argument 라는 노드 이름을 초기화했습니다.
2. QoS 설정을 reliable, keep_last, depth 10, volatile로 하였습니다.

토픽 서브스크라이버 코드

```
import time

from msg_srv_action_interface_example.action import ArithmeticChecker
from msg_srv_action_interface_example.msg import ArithmeticArgument
from msg_srv_action_interface_example.srv import ArithmeticOperator
from rclpy.action import ActionServer
from rclpy.callback_groups import ReentrantCallbackGroup
from rclpy.node import Node
from rclpy.qos import QoSDurabilityPolicy
from rclpy.qos import QoSHistoryPolicy
from rclpy.qos import QoSProfile
from rclpy.qos import QoSReliabilityPolicy


class Calculator(Node):

    def __init__(self):
        super().__init__('calculator')
        self.argument_a = 0.0
        self.argument_b = 0.0
        self.argument_operator = 0
        self.argument_result = 0.0
        self.argument_formula = ''
        self.operator = ['+', '-', '*', '/']
        self.callback_group = ReentrantCallbackGroup()
```

```
        self.declare_parameter('qos_depth', 10)
        qos_depth = self.get_parameter('qos_depth').value

        QOS_RKL10V = QoSProfile(
            reliability=QoSReliabilityPolicy.RELIABLE,
            history=QoSHistoryPolicy.KEEP_LAST,
            depth=qos_depth,
            durability=QoSDurabilityPolicy.VOLATILE)

        self.arithmetic_argument_subscriber = self.create_subscription(
            ArithmeticArgument,
            'arithmetic_argument',
            self.get_arithmetic_argument,
            QOS_RKL10V,
            callback_group=self.callback_group)

        self.arithmetic_service_server = self.create_service(
            ArithmeticOperator,
            'arithmetic_operator',
            self.get_arithmetic_operator,
            callback_group=self.callback_group)

        self.arithmetic_action_server = ActionServer(
            self,
            ArithmeticChecker,
            'arithmetic_checker',
            self.execute_checker,
            callback_group=self.callback_group)

    def get_arithmetic_argument(self, msg):
        self.argument_a = msg.argument_a
        self.argument_b = msg.argument_b
        self.get_logger().info('Timestamp of the message: {0}'.format(msg.stamp))
        self.get_logger().info('Subscribed argument a: {0}'.format(self.argument_a))
        self.get_logger().info('Subscribed argument b: {0}'.format(self.argument_b))

    def get_arithmetic_operator(self, request, response):
        self.argument_operator = request.arithmetic_operator
        self.argument_result = self.calculate_given_formula(
            self.argument_a,
            self.argument_b,
            self.argument_operator)
        response.arithmetic_result = self.argument_result
        self.argument_formula = '{0} {1} {2} = {3}'.format(
                self.argument_a,
                self.operator[self.argument_operator-1],
                self.argument_b,
                self.argument_result)
        self.get_logger().info(self.argument_formula)
        return response

    def calculate_given_formula(self, a, b, operator):
        if operator == ArithmeticOperator.Request.PLUS:
            self.argument_result = a + b
        elif operator == ArithmeticOperator.Request.MINUS:
            self.argument_result = a - b
        elif operator == ArithmeticOperator.Request.MULTIPLY:
            self.argument_result = a * b
        elif operator == ArithmeticOperator.Request.DIVISION:
            try:
                self.argument_result = a / b
            except ZeroDivisionError:
                self.get_logger().error('ZeroDivisionError!')
                self.argument_result = 0.0
                return self.argument_result
        else:
            self.get_logger().error(
                'Please make sure arithmetic operator(plus, minus, multiply, division).')
            self.argument_result = 0.0
        return self.argument_result

    def execute_checker(self, goal_handle):
        self.get_logger().info('Execute arithmetic_checker action!')
        feedback_msg = ArithmeticChecker.Feedback()
        feedback_msg.formula = []
        total_sum = 0.0
```

```
        goal_sum = goal_handle.request.goal_sum
        while total_sum < goal_sum:
            total_sum += self.argument_result
            feedback_msg.formula.append(self.argument_formula)
            self.get_logger().info('Feedback: {0}'.format(feedback_msg.formula))
            goal_handle.publish_feedback(feedback_msg)
            time.sleep(1)
        goal_handle.succeed()
        result = ArithmeticChecker.Result()
        result.all_formula = feedback_msg.formula
        result.total_sum = total_sum
        return result
```

1. Calculator 클래스는 rclpy.node 모듈의 Node 클래스를 상속받았고, 생성자를 통해 argument 라는 노드 이름을 초기화했습니다.
2. QoS 설정을 reliable, keep_last, depth 10, volatile로 하였습니다.

8장. 서비스 프로그래밍 (파이썬)

서비스는 클라이언트 단에서 특정 요청을 하고, 서버 단에서 해당 요청을 수행한 후 결과 값을 전달하는 통신 과정으로 이해할 수 있습니다.

서비스 서버 코드

```
import time

from msg_srv_action_interface_example.action import ArithmeticChecker
from msg_srv_action_interface_example.msg import ArithmeticArgument
from msg_srv_action_interface_example.srv import ArithmeticOperator
from rclpy.action import ActionServer
from rclpy.callback_groups import ReentrantCallbackGroup
from rclpy.node import Node
from rclpy.qos import QoSDurabilityPolicy
from rclpy.qos import QoSHistoryPolicy
from rclpy.qos import QoSProfile
from rclpy.qos import QoSReliabilityPolicy


class Calculator(Node):

    def __init__(self):
        super().__init__('calculator')
        self.argument_a = 0.0
        self.argument_b = 0.0
        self.argument_operator = 0
        self.argument_result = 0.0
        self.argument_formula = ''
        self.operator = ['+', '-', '*', '/']
        self.callback_group = ReentrantCallbackGroup()

        self.declare_parameter('qos_depth', 10)
        qos_depth = self.get_parameter('qos_depth').value

        QOS_RKL10V = QoSProfile(
            reliability=QoSReliabilityPolicy.RELIABLE,
            history=QoSHistoryPolicy.KEEP_LAST,
            depth=qos_depth,
            durability=QoSDurabilityPolicy.VOLATILE)

        self.arithmetic_argument_subscriber = self.create_subscription(
            ArithmeticArgument,
            'arithmetic_argument',
            self.get_arithmetic_argument,
            QOS_RKL10V,
            callback_group=self.callback_group)

        self.arithmetic_service_server = self.create_service(
            ArithmeticOperator,
            'arithmetic_operator',
```

```
            self.get_arithmetic_operator,
            callback_group=self.callback_group)

        self.arithmetic_action_server = ActionServer(
            self,
            ArithmeticChecker,
            'arithmetic_checker',
            self.execute_checker,
            callback_group=self.callback_group)

    def get_arithmetic_argument(self, msg):
        self.argument_a = msg.argument_a
        self.argument_b = msg.argument_b
        self.get_logger().info('Timestamp of the message: {0}'.format(msg.stamp))
        self.get_logger().info('Subscribed argument a: {0}'.format(self.argument_a))
        self.get_logger().info('Subscribed argument b: {0}'.format(self.argument_b))

    def get_arithmetic_operator(self, request, response):
        self.argument_operator = request.arithmetic_operator
        self.argument_result = self.calculate_given_formula(
            self.argument_a,
            self.argument_b,
            self.argument_operator)
        response.arithmetic_result = self.argument_result
        self.argument_formula = '{0} {1} {2} = {3}'.format(
                self.argument_a,
                self.operator[self.argument_operator-1],
                self.argument_b,
                self.argument_result)
        self.get_logger().info(self.argument_formula)
        return response

    def calculate_given_formula(self, a, b, operator):
        if operator == ArithmeticOperator.Request.PLUS:
            self.argument_result = a + b
        elif operator == ArithmeticOperator.Request.MINUS:
            self.argument_result = a - b
        elif operator == ArithmeticOperator.Request.MULTIPLY:
            self.argument_result = a * b
        elif operator == ArithmeticOperator.Request.DIVISION:
            try:
                self.argument_result = a / b
            except ZeroDivisionError:
                self.get_logger().error('ZeroDivisionError!')
                self.argument_result = 0.0
                return self.argument_result
        else:
            self.get_logger().error(
                'Please make sure arithmetic operator(plus, minus, multiply, division).')
            self.argument_result = 0.0
        return self.argument_result

    def execute_checker(self, goal_handle):
        self.get_logger().info('Execute arithmetic_checker action!')
        feedback_msg = ArithmeticChecker.Feedback()
        feedback_msg.formula = []
        total_sum = 0.0
        goal_sum = goal_handle.request.goal_sum
        while total_sum < goal_sum:
            total_sum += self.argument_result
            feedback_msg.formula.append(self.argument_formula)
            self.get_logger().info('Feedback: {0}'.format(feedback_msg.formula))
            goal_handle.publish_feedback(feedback_msg)
            time.sleep(1)
        goal_handle.succeed()
        result = ArithmeticChecker.Result()
        result.all_formula = feedback_msg.formula
        result.total_sum = total_sum
        return result
```

9장. 액션 프로그래밍 (파이썬)

액션은 `클라이언트`와 '서버' 노드간의 메시지 전송 방식 입니다. (동기식 양방향 송수신)

1. `클라이언트`는 액션의 목표를 전송 합니다. 서버는 액션 서비스를 처리 합니다.
2. `서버`는 `클라이언트`에게 피드백을 주기적으로 제공합니다.
3. `서버`는 `클라이언트`에게 액션 결과를 제공합니다.

```
  self.arithmetic_action_server = ActionServer(
      self,
      ArithmeticChecker,
      'arithmetic_checker',
      self.execute_checker,
      callback_group=self.callback_group)
```

goal_handle.request.goal_sum 를 통해 목표값을 불러옵니다.

```
def execute_checker(self, goal_handle):
        self.get_logger().info('Execute arithmetic_checker action!')
        feedback_msg = ArithmeticChecker.Feedback()
        feedback_msg.formula = []
        total_sum = 0.0
        goal_sum = goal_handle.request.goal_sum
        while total_sum < goal_sum:
            total_sum += self.argument_result
            feedback_msg.formula.append(self.argument_formula)
            self.get_logger().info('Feedback: {0}'.format(feedback_msg.formula))
            goal_handle.publish_feedback(feedback_msg)
            time.sleep(1)
        goal_handle.succeed()
        result = ArithmeticChecker.Result()
        result.all_formula = feedback_msg.formula
        result.total_sum = total_sum
        return result
```

액션 클라이언트 코드

```
from action_msgs.msg import GoalStatus
from msg_srv_action_interface_example.action import ArithmeticChecker
from rclpy.action import ActionClient
from rclpy.node import Node


class Checker(Node):

    def __init__(self):
        super().__init__('checker')
        self.arithmetic_action_client = ActionClient(
          self,
          ArithmeticChecker,
          'arithmetic_checker')

    def send_goal_total_sum(self, goal_sum):
        wait_count = 1
        while not self.arithmetic_action_client.wait_for_server(timeout_sec=0.1):
            if wait_count > 3:
                self.get_logger().warning('Arithmetic action server is not available.')
                return False
            wait_count += 1
        goal_msg = ArithmeticChecker.Goal()
        goal_msg.goal_sum = (float)(goal_sum)
        self.send_goal_future = self.arithmetic_action_client.send_goal_async(
            goal_msg,
            feedback_callback=self.get_arithmetic_action_feedback)
        self.send_goal_future.add_done_callback(self.get_arithmetic_action_goal)
        return True

    def get_arithmetic_action_goal(self, future):
        goal_handle = future.result()
        if not goal_handle.accepted:
            self.get_logger().warning('Action goal rejected.')
```

```
            return
        self.get_logger().info('Action goal accepted.')
        self.action_result_future = goal_handle.get_result_async()
        self.action_result_future.add_done_callback(self.get_arithmetic_action_result)

    def get_arithmetic_action_feedback(self, feedback_msg):
        action_feedback = feedback_msg.feedback.formula
        self.get_logger().info('Action feedback: {0}'.format(action_feedback))

    def get_arithmetic_action_result(self, future):
        action_status = future.result().status
        action_result = future.result().result
        if action_status == GoalStatus.STATUS_SUCCEEDED:
            self.get_logger().info('Action succeeded!')
            self.get_logger().info(
                'Action result(all formula): {0}'.format(action_result.all_formula))
            self.get_logger().info(
                'Action result(total sum): {0}'.format(action_result.total_sum))
        else:
            self.get_logger().warning(
                'Action failed with status: {0}'.format(action_status))
```

10장. 파라미터 프로그래밍 (파이썬)

파라미터는 서비스와 유사합니다. 파라미터는 서비스의 요청과 응답과는 달리, 노드 내부 또는 외부에서 매개변수 Set, Get이 주 목적입니다.

```
import random

from msg_srv_action_interface_example.msg import ArithmeticArgument
from rcl_interfaces.msg import SetParametersResult
import rclpy
from rclpy.node import Node
from rclpy.parameter import Parameter
from rclpy.qos import QoSDurabilityPolicy
from rclpy.qos import QoSHistoryPolicy
from rclpy.qos import QoSProfile
from rclpy.qos import QoSReliabilityPolicy


class Argument(Node):

    def __init__(self):
        super().__init__('argument')
        self.declare_parameter('qos_depth', 10)
        qos_depth = self.get_parameter('qos_depth').value
        self.declare_parameter('min_random_num', 0)
        self.min_random_num = self.get_parameter('min_random_num').value
        self.declare_parameter('max_random_num', 9)
        self.max_random_num = self.get_parameter('max_random_num').value
        self.add_on_set_parameters_callback(self.update_parameter)

        QOS_RKL10V = QoSProfile(
            reliability=QoSReliabilityPolicy.RELIABLE,
            history=QoSHistoryPolicy.KEEP_LAST,
            depth=qos_depth,
            durability=QoSDurabilityPolicy.VOLATILE)

        self.arithmetic_argument_publisher = self.create_publisher(
            ArithmeticArgument,
            'arithmetic_argument',
            QOS_RKL10V)

        self.timer = self.create_timer(1.0, self.publish_random_arithmetic_arguments)

    def publish_random_arithmetic_arguments(self):
        msg = ArithmeticArgument()
        msg.stamp = self.get_clock().now().to_msg()
        msg.argument_a = float(random.randint(self.min_random_num, self.max_random_num))
        msg.argument_b = float(random.randint(self.min_random_num, self.max_random_num))
        self.arithmetic_argument_publisher.publish(msg)
        self.get_logger().info('Published argument a: {0}'.format(msg.argument_a))
```

```
        self.get_logger().info('Published argument b: {0}'.format(msg.argument_b))

    def update_parameter(self, params):
        for param in params:
            if param.name == 'min_random_num' and param.type_ == Parameter.Type.INTEGER:
                self.min_random_num = param.value
            elif param.name == 'max_random_num' and param.type_ == Parameter.Type.INTEGER:
                self.max_random_num = param.value
        return SetParametersResult(successful=True)


def main(args=None):
    rclpy.init(args=args)
    try:
        argument = Argument()
        try:
            rclpy.spin(argument)
        except KeyboardInterrupt:
            argument.get_logger().info('Keyboard Interrupt (SIGINT)')
        finally:
            argument.destroy_node()
    finally:
        rclpy.shutdown()


if __name__ == '__main__':
    main()
```

11장. 실행 인자 프로그래밍 (파이썬)

프로그램을 실행할 때, 실행에 필요한 인자를 옵션으로 추가하여 실행하는 경우가 있습니다. 이때 사용되는 인자를 실행인자라고 합니다.

ROS2 에서 실행 인자 처리

```
def main(args=None):

    rclpy.init(args=args)
```

실행인자 구문 해석

```
import argparse
import sys

import rclpy

from topic_service_action_rclpy_example.checker.checker import Checker


def main(argv=sys.argv[1:]):
    parser = argparse.ArgumentParser(formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument(
        '-g',
        '--goal_total_sum',
        type=int,
        default=50,
        help='Target goal value of total sum')
    parser.add_argument(
        'argv', nargs=argparse.REMAINDER,
        help='Pass arbitrary arguments to the executable')
    args = parser.parse_args()

    rclpy.init(args=args.argv)
    try:
        checker = Checker()
        checker.send_goal_total_sum(args.goal_total_sum)
        try:
            rclpy.spin(checker)
```

```
        except KeyboardInterrupt:
            checker.get_logger().info('Keyboard Interrupt (SIGINT)')
        finally:
            checker.arithmetic_action_client.destroy()
            checker.destroy_node()
    finally:
        rclpy.shutdown()


if __name__ == '__main__':
    main()
```

1. 파서 만들기
2. 인자 추가하기
3. 인자 파싱하기
4. 인자 사용하기