# Week 8

## 1. 토픽 프로그래밍(파이썬)

- argument.py

```
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()
```

- rclpy.qos 모듈의 QoSProfile 클래스를 이용하여 토픽 퍼블리셔에서 사용할 QoS 설정을 해줌.
- self.create_publisher( 토픽 타입, 토픽 이름, QOS )
- self.create_timer( 초, 토픽 실행 )

```
    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))
```

- msg.stamp로 msg 발행

- calculator.py

```
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
```

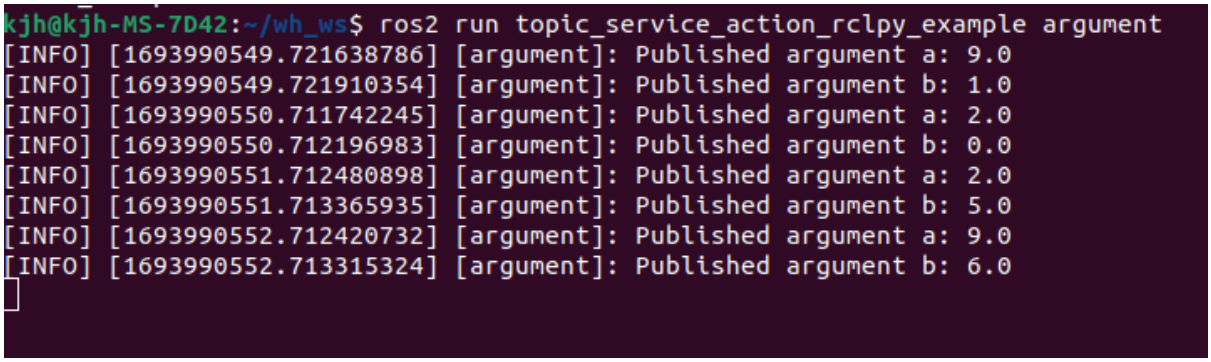
- subscriber, service, action 모두 포함

- subscriber 부분

```
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.create_subscription( 토픽 타입, 이름, QOS, 콜백함수 )

```
$ ros2 run topic_service_action_rclpy_example calculator
$ ros2 run topic_service_action_rclpy_example argument
```




# 2. 서비스 프로그래밍

- 서비스의 요청을 하는 쪽을 서비스 클라이언트, 요청받은 서비스를 수행한 후 응답하는쪽이 서비스 서버

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

- self.create_service( 서비스 타입, 서비스 이름, 콜백함수 )

- get_arithmetic_operator

```
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
```

- arithmetic_operator를 argument_operator 멤버 변수에 저장하고 이를 calculate_given_formula 함수의 인자로 넘겨줌