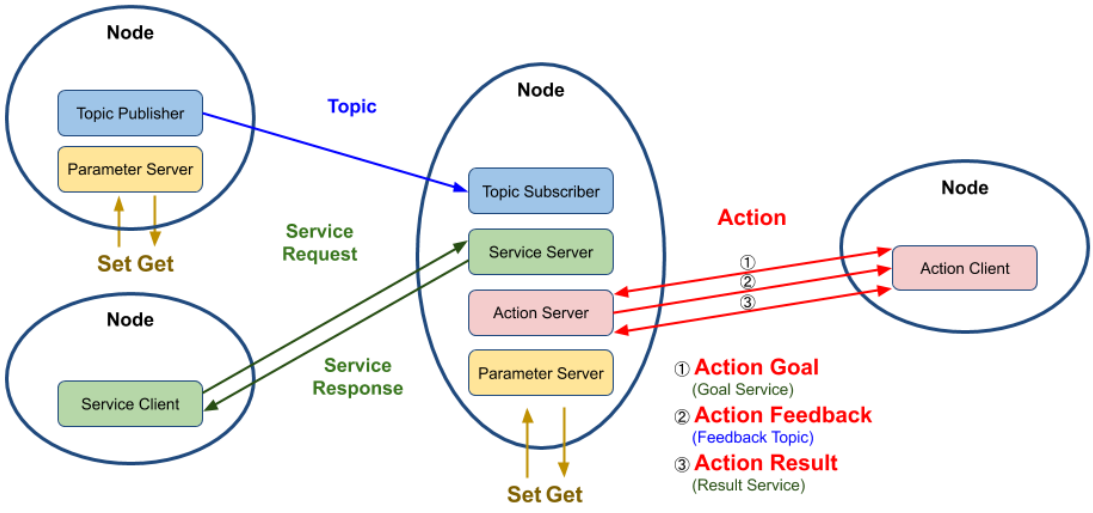
Week_8

6장 ROS2 패키지 설계(파이썬)

6.1 패키지 설계

- ROS 로봇 프로그램이
- 일반적인 로봇 프로그램과 다른 점은
    - ROS 로봇 프로그램은 프로세스를 **목적별로 나누어** 노드(Node)단위의 프로그램을 작성하고 노드와 노드간의 **데이터 통신을 고려하여 설계**해야 한다는 점이다.



- 예시에서 각 노드는 1개 **<u>이상</u>**의 publisher/subscribe 또는 server/client 요소를 포함하고 있다.

- 책에서 설명되어 있는 예시대로, 각 프로세스를 구성한다.

- argument_node
  - arithmetic_argument 토픽으로
    - 현재 시간
    - 변수 a와 b를 publish한다.

- calculator_node
  - arithmetic_argument 토픽으로
    - 생성된 시간
    - 변수 a와 b를 subscribe한다.

- operator_node
  - arithmetic_operator 서비스로
    - calculator노드에 연산자(+,-,*,/) 를 서비스로 요청한다.

- calculator_node
  - 저장하고 있던 변수 a와 b를

    operator_node로부터 받은 연산자와 조합하여 연산한 후에 operator_node에 결괏값을

    서비스의 응답으로 반환한다.
  - **액션 목푯값을 전달**받았다면, 연산 결과값을 누적한 후에 **액션 피드백**으로 연산식을 반환한다.

    누적된 연산 결과값이 전달받은 액션 목푯값보다 **커지면** 액션 결과값으로 **누적된 연산 결괏값과 진행된 연산식 전부를 보낸다.**
- checker_node
  - 연산 결과값의 누적 한계치를 액션 목표값으로 전달한다.

각 노드의 역활은 다음과 같다.

- argument_node (토픽 발행)
  - 현재 시간과 변수a와 b를 퍼블리시한다

- operator_node (서비스)
  - 연산자를 서비스 요청값으로 보낸다

- checker_node (액션)
  - Goal Service : 연산값의 한계를 전달한다
  - Feedback Topic : 연산 계산식을 checker_node에 반환한다.
  - Result Service : 연산값의 한계를 넘긴 경우(Goal Service가 충족된 경우) Action Result로 최종 연산 합계를 보낸다.

- calaulator_node
  - 각 노드로부터 받은 역활을 수행한다.
  - Topic Subscriber와
  - 다수의 Server로 구성되어 있다.

- 패키지 생성

```
kody@desktop:~/ros2_study/src$ ros2 pkg create tsa_rclpy_example --build-type ament_python --dependencies rclpy std_msgs msa_example
going to create a new package
package name: tsa_rclpy_example
destination directory: /home/kody/ros2_study/src
package format: 3
version: 0.0.0
description: TODO: Package description
maintainer: ['kody <pg01198@naver.com>']
licenses: ['TODO: License declaration']
build type: ament_python
dependencies: ['rclpy', 'std_msgs', 'msa_example']
creating folder ./tsa_rclpy_example
creating ./tsa_rclpy_example/package.xml
creating source folder
creating folder ./tsa_rclpy_example/tsa_rclpy_example
creating ./tsa_rclpy_example/setup.py
creating ./tsa_rclpy_example/setup.cfg
creating folder ./tsa_rclpy_example/resource
creating ./tsa_rclpy_example/resource/tsa_rclpy_example
creating ./tsa_rclpy_example/tsa_rclpy_example/__init__.py
creating folder ./tsa_rclpy_example/test
creating ./tsa_rclpy_example/test/test_copyright.py
creating ./tsa_rclpy_example/test/test_flake8.py
creating ./tsa_rclpy_example/test/test_pep257.py

[WARNING]: Unknown license 'TODO: License declaration'.  This has been set in the package.xml, but no LICENSE file has been created.
It is recommended to use one of the ament license identitifers:
Apache-2.0
BSL-1.0
BSD-2.0
BSD-2-Clause
BSD-3-Clause
GPL-3.0-only
LGPL-3.0-only
MIT
MIT-0
```

package.xml

- 생성 직후에 보니, 책에서 설명한 패키지들이 이미 설정되어 있다.
    - Lint 패키지와 build_type 패키지

```
<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>
```

setup.py

- glob
- 전역 변수에 대한 모듈이 아니다.

  globbing together files, 즉 파일을 하나로 묶는 역활을 한다.
  파일의 리스트를 만드는 작업을 한다.

- 파일이 있고

```
file1.txt
file2.txt
file3.txt
image1.png
image2.png
```

- glob를 사용하여

```
import glob

txt_files = glob.glob('*.txt')

print(txt_files)
```

- output을 만들 수 있다

```
['file1.txt', 'file2.txt', 'file3.txt']
```

- setup( )의 data_file= [ ] 부분 수정

```
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')))
        #만든 변수 share_dir을 활용한다.
    ],
```

- data_files=[ ]
  - data_files에 포함되는 리스트는 패키지가 설치될 때, 패키지에 포함되는 추가 파일을 명시하는 부분이다.

인수 부분

```
( 컴파일시 설치할 디렉토리 , 소스 파일의 대상 파일 )
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
```

- (share_dir + '/launch', glob.glob(os.path.join('launch', '*.launch.py')))은 어떻게 동작하는가?

  - os.path.join( ) 은 두개의 string을 현재 운영 체제(os)에 적합한 형태로 접합(join)한다.

    유닉스 기반이라면 **'launch/*.launch.py'**을 반환한다.

  - glob.glob( )는 파일에 대한 리스트를 만든다.

  반환하는 것은 다음과 같다.

  ```
  ['launch/launch1.launch.py', 'launch/launch2.launch.py',
   'launch/launch3.launch.py']
  ```

- entry_points - console_scripts를 수정한다

```
entry_points={
    # 패키지가 설치될 때 만들어져야 하는 executable script을(실행 파일을) 명시하는 부분을 말한다.
        'console_scripts': [
            # 형식은 다음과 같다.
            # 'name = module:callable'

            # name은 만들 실행파일의 이름이다.
            # - 실행파일을 실행하기 위해 터미널에 입력해야 하는 명령이기도 하다.

            # module은 실행할 함수가 포함된 파이썬 모듈의 import 경로이다.

            # callable은 스크립트가 실행 될 때에 불러올 함수의 이름이다.
            # 여기에서는 main을 사용했다.

            'argument = tsa_rclpy_example.arithmetic.argument:main',
            'operator = tsa_rclpy_example.arithmetic.operator:main',
            'calculator = tsa_rclpy_example.calculator:main',
            'checker = tsa_rclpy_example.checker:main',
        ],
    },
```

7.2 토픽 퍼블리셔 코드

arithmetic/argument.py

```
import random

from msa_example import ArithArgument
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')
```

```
        # argument 노드를 생성한다.
        self.declare_parameter('qos_depth',10)
        # qos_depth 파라미터를 10을 설정한다.
        qos_depth = self.set_parameter('qos_depth').value
        # 로컬변수 qos_depth에 10을 할당한다.

        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.man_random_num = self.get_parameter('max_random_num').value

        self.add_on_set_parameters_callback(self.update_parameter)
        # 콜백 함수로 update_parameter함수를 등록한다.

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

        self.arithmetic_argument_publisher = self.create_publisher(
            ArithmeticArgument,
            #토픽이름
            'arithmetic_argument',
            #메시지이름
            QOS_RKL10V)
            #사용할 QoS

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

    #앞의 self.timer로 1초마다 실행되게 설정한 함수이다.
    def publish_random_arithmetic_arguments(self):
        # 변수 msg에 ArithArgument()를 할당한다.
        msg = ArithArgument()

        # msg의 stamp필드를 현재 시간을 나타내는,  builtin_interfaces/Time형태로 발행한다.
        msg.stamp = self.get_clock().now().to_msg()

        # random int값을 생성하여 float값으로 반환한다.
        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))

        #arithmetic_argument_publisher를 호출하여 msg를 publish한다.
        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:
            # param.name에 따라
            # min값과 max값에 파라미터를 할당한다.
            if param.name == 'min_random_num' and param.type_ == Parameter.Type.INTEGER:
                self.min_random_num = param.value
            if 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()
```

7.3 토픽 서브스크라이버 코드

calculator/calculator.py

- 생략, 책에 포함된 예제만 살펴봄

```
calss Calculator(Node):

  def __init__ == '__main__':
    super().__init__('calculator')
    # argument를 0.0으로 초기화한다.
    self.argument_a = 0.0
    self.argument_b = 0.0
    #
    self.callback_group = ReentrantCallbackGroup()
#콜백 그룹에는
#MutuallyExclusive와
#Reentrant가 있다.

#여기에서는 ReentrantCallbackGroup을 선택하여 콜백을 병렬로 처리할 수 있게 하였다.
# 콜백 그룹을 선택하지 않으면 기본콜백 그룹이
# MultThreadedExcutor으로 설정되기 때문에 주의가 필요하다.

    (중략)
    QOS_RKL10V = QoSProfile(
      #QoS를 설정했다.
      reliability=QoSReliabilityPolicy.RELIABLE,
      history=QoSHistoryPolicy.KEEP_LAST,
      depth=qos_depth,
      durability=QoSDurabilityPolicy.VOLATILE)
```

```
    self.arithmetic_argument_subscriber = self.create.subscription(
      #토픽 이름
      ArithmeticArgument,
      #메시지 이름
      'arithmetic_argument',
      #콜백 함수
      self.get_arithmetic_argument,
      #QoS
      QOS_RKL10V,
      #콜백 그룹
      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('

    self.get_logger().info('
    self.get_logger().info('
```

7.4 노드 실행 코드

calculator/main.py

```
import rclpy
from rclpy.executors import MultiThreadedExecutor

from tsa_example.calculator.calculator import Calculator

def main(args=None):
    rclpy.init(args=args)
    try:
        calculator = Calculator()
        executor = MultiThreadedExecutor(num_thread=4)
        executor.add_node(calculator)
        try:
            executor.spin()
        except KeyboardInterrupt:
            calculator.get_logger().info('Keyboard Interrupt (SIGINT)')
```

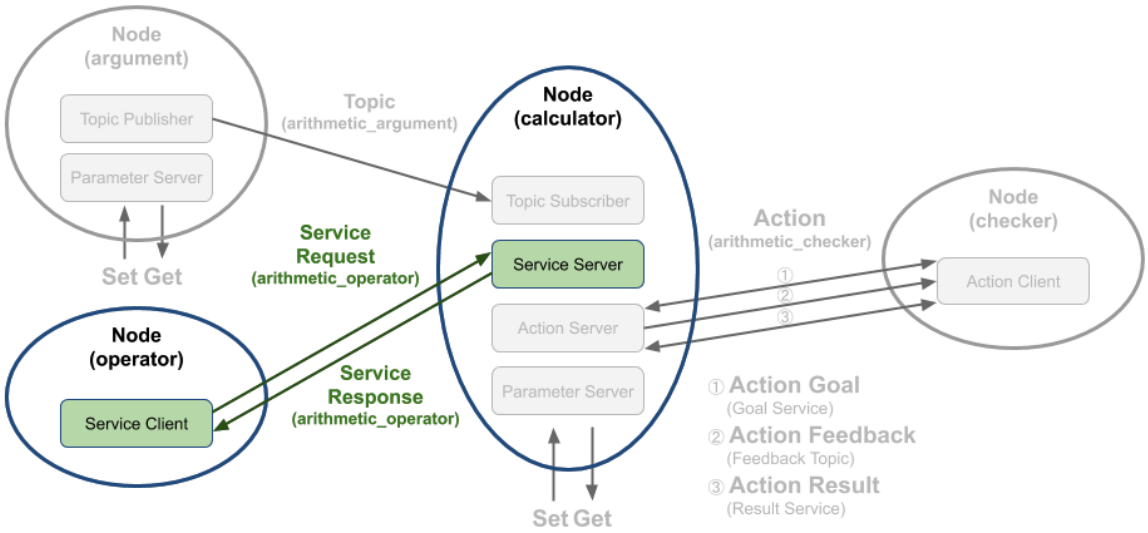
```
        finally:
            executor.shutdown()
            calculator.arithmetic_action_server.destroy()
            calculator.destroy_node()
    finally:
        rclpy.shutdown()

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

8장 서비스 프로그래밍(파이썬)

8.1 서비스

- 서비스는 **동기식 양방향 메시지 송수신** 방식이다.
  - 요청(Request)하는 측을 Client라고 하고
  - 응답(Response)하는 측을 Server라고 한다.



8.2 서비스 서버 코드

- 관련된 내용만 살펴본다.

calculator/calculator.py

create_service 메소드를 사용해서 service를 생성하는 부분

- part 1

```
self.arithmetic_service_server = self.create_service(
# 서비스 : 클라이언트 <-> 서버 에서
# 서버를 생성할 때에 create_service메소드를 사용한다.
  ArithmeticOperator,
  'arithmetic_operator',
  self.get_arithmetic_operator,
  callback_group=self.callback_group)
```

- part 2

```
def get_arithmetic_operator(self, request, response):
#get_arithmetic_operator의 argument_operator에
# requset의 arithmetic_operator를 할당한다.
  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

#식에 변수들을 할당함
  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
```

- part 3

```
  def calculate_given_formula(self, a, b, operator):
  #self로 포인터를 지정하고, a,b,operator를 전달하여
  # + - * / 연산을 진행한다.
    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
      #try문에서 에러가 발생하면 except문을 실행한다.
      except ZeroDivitionError:
        self.get_logger().error('ZeroDivitionError!')
        self.argument_result = 0.0
        return self.argument_result

      else:
        self.get_logger().error(
          'Plsase makes ure arithmetic operator(plus, minus, multiply, division).')
        self.argument_result = 0.0
      return self.argument_result
```

서비스 클라이언트 코드

- **arithmetic_operator 서비스 이름**으로 (calculator노드에)

  연산자(+ - * /)를 **서비스 요청**값으로 보낸다.

- part 1

arithmetic/operator.py

```
class Operator(Node):
  def __init__(self)
    super().__init__('operator')

    self.arithmetic_service_client = self.create_clinet(
    # 서비스: 서비스 <-> 클라이언트에서
    # 클라이언트를 생성하는 부분이다.
    # create_client 메소드를 사용한다.
```

```
  #서비스 타입을 전달한다
    ArithmeticOperator,
  #서비스 이름을 전달한다.
    'arithmetic_operator')

  while not self.arithmetic_service_client.wait_for_service(timeout_sec=0.1):
    self.get_logger().warning('The arithmetic_operator service not available.')
```

- part 2

```
def send_request(self):
  service_request = ArithmeticOperator.Request()
  service_request.arithmetic_operator = random.randint(1,4)
  futures = self.arithmetic_service_client.call_async(service_request)
  return futures
```

서비스 클라이언트 노트 실행 코드