Service 프로그래밍 - Python

해당 실습 자료는 한양대학교 Road Balance - ROS 2 for G Camp와 ROS 2 Documentation: Foxy, 표윤석, 임태훈 <ROS 2로 시작하는 로봇 프로그래밍> 루피페이퍼(2022) 를 참고하여 작성하였습니다.

> 앞선 장에서 Service에 대해 배워보았습니다. 간단하게 정리하자면 node가 Service를 사용하여 통신할 때 데이터 요청(`request`)을 보내는 node를 **Client Node**라고 하고 요청에 응답(`response`)하는 **Server Node**라고 합니다. 요청과 응답의 구조는 `.srv` 파일에 의해 결정됩니다.

Service 프로그래밍

- 이번 장에서는 간단한 service 프로그래밍을 위해 간단한 정수 덧셈 시스템을 구현해봅니다.
- **Client Node**에서는 두 정수의 합을 `request` 하고, **Server Node**에서는 연산 결과를 `response` 합니다.

패키지 생성

- 작성한 워크스페이스인 `ros2_ws/src` 디렉토리에 이동하신 다음 새로운 패키지를 생성합니다.

```
$ ros2 pkg create py_srvcli --build-type ament_python  --depe
```

- 새로운 패키지 이름은 `py_srvcli` 으로 동명의 디렉토리에 패키지 기본 구성이 생성된 것을 확인 할 수 있을 겁니다.

- 추가로 `--dependencies` 인수를 통해 패키지 환경 설정 파일 `package.xml` 에 필요한 종속성 패키지인 `rclpy` , `example_interfaces` 가 자동으로 추가됩니다.
- `example_interfaces` 는 `request`와 `response` 을 구성하는 데에 필요한 `.srv` 파일이 포함된 패키지 입니다.

```
int64 a
int64 b
---
int64 sum
```

인터페이스

- 노드 간의 데이터를 주고받을 때 사용되는 데이터의 형태를 **인터페이스(interface)**라 하며, 유저가 원하는 형태로 구성된 인터페이스를 생성할 수 있습니다.
  (
  단순 자료형을 기본으로하며 메시지를 포함하는 간단한 데이터 구조 및 메시지들이 나열된 배열 구조로 사용할 수 있습니다.)
  - 메시지를 포함하는 간단한 데이터 구조 및 메시지들이 나열된 배열 구조는 단순 자료형을 기반으로하며 정의시 아래와 같이 기술합니다.
    - fieldtype은 메시지 자료형, fieldname은 메시지 이름에 해당합니다.

	msg 인터페이스	srv 인터페이스	action 인터페이스
확장자	*.msg	*.srv	*.action
데이터	토픽 데이터 (data)	서비스 요청 (request) --- 서비스 응답 (response)	액션 목표 (goal) --- 액션 결과 (result) --- 액션 피드백 (feedback)
형식	fieldtype1 fieldname1 fieldtype2 fieldname2 fieldtype3 fieldname3	fieldtype1 fieldname1 fieldtype2 fieldname2 --- fieldtype3 fieldname3 fieldtype4 fieldname4	fieldtype1 fieldname1 fieldtype2 fieldname2 --- fieldtype3 fieldname3 fieldtype4 fieldname4 --- fieldtype5 fieldname5 fieldtype6 fieldname6

▼ [참고]기본 자료형과 언어별 자료형 매칭

Type name	Python	C++	DDS type
bool	builtins.bool	bool	boolean
byte	builtins.bytes*	uint8_t	octet
char	builtins.str*	char	char
float32	builtins.float*	float	float
float64	builtins.float*	double	double
int8	builtins.int*	int8_t	octet
uint8	builtins.int*	uint8_t	octet
int16	builtins.int*	int16_t	short
uint16	builtins.int*	uint16_t	unsigned short
int32	builtins.int*	int32_t	long
uint32	builtins.int*	uint32_t	unsigned long
int64	builtins.int*	int64_t	long long
uint64	builtins.int*	uint64_t	unsigned long long
string	builtins.str	std::string	string
wstring	builtins.str	std::u16string	wstring
static array	builtins.list*	std::array<T, N>	T[N]
unbounded dynamic array	builtins.list	std::vector	sequence
bounded dynamic array	builtins.list*	custom_class<T, N>	sequence<T, N>
bounded string	builtins.str*	std::string	string

[출처] 016 ROS 2 인터페이스 (interface) (오픈소스 소프트웨어 & 하드웨어: 로봇 기술 공유 카페 (오로카)) | 작성자 표윤석

Server Node 작성

- `ros2_ws/src/py_srvcli/py_srvcli` 경로에 새로운 파이썬 스크립트 `service_member_function.py` 생성하여 아래의 코드를 작성해주세요

```
from example_interfaces.srv import AddTwoInts

import rclpy
from rclpy.node import Node


class MinimalService(Node):

    def __init__(self):
        super().__init__('minimal_service')

        ### ============= 서버 설정 =============
        self.srv = self.create_service(
            AddTwoInts,
```

```
                        ## srv 타입: 해당 클래스의 인터페이스로 서비
            'add_two_ints',
                        ## 서비스명
            self.add_two_ints_callback) ## 콜백 함수
        ### ========================================

    def add_two_ints_callback(self, request, response):
        ## Client Node로부터 클래스로 생성된 인터페이스로 서비스 요청에
             request 부분과 응답에 해당되는 response으로 구분
        response.sum = request.a + request.b
                ## 위의 AddTwoInts 인터페이스 정보 확인
        self.get_logger().info('Incoming request\na: %d b: %d
                # cmd 창 출력

        return response ## 응답값 반환


def main(args=None):
    rclpy.init(args=args) # 초기화
    node = MinimalService() # MinimalService를 node라는 이름으로
    try:
        rclpy.spin(node) # 생성한 노드를 spin하여 지정된 콜백 함수 실
    except KeyboardInterrupt:
        node.get_logger().info('Keyboard Interrupt (SIGINT)')
    finally:# 종료시 (ex `Ctrl + c`)
        node.destroy_node()  # 노드 소멸
        rclpy.shutdown() # rclpy.shutdown 함수로 노드 종료

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

- 먼저 `import` 구문을 살펴보겠습니다.

```
from example_interfaces.srv import AddTwoInts
```

```
import rclpy
from rclpy.node import Node
```

- Topic과 굉장히 유사한 형태를 가지고 있는 것을 볼 수 있습니다. 조금 다른 점은 `from example_interfaces.srv import AddTwoInts` 메세지를 불러올 때, `msg`가 아닌 `srv` 인 것을 볼 수 있습니다.

- Topic과 동일하게 부모 클래스(Node)를 상속하고 노드 명을 지정해줍니다. 해당 예시에서는 노드 명을 `'minimal_service'`로 지정했습니다.

```
class MinimalService(Node):
    def __init__(self):
        super().__init__('minimal_service')
        self.srv = self.create_service(AddTwoInts, 'add_two_i
```

- Topic의 `create_publisher` 와 유사하게 Service에서는 `create_service(<srv 타입>, <서비스 서버 명>, <콜백 함수>)` 를 통해 node를 생성합니다.

- 콜백 함수인 `self.add_two_ints_callback` 을 살펴볼까요?

```
def add_two_ints_callback(self, request, response):
    response.sum = request.a + request.b
    self.get_logger().info('Incoming request\na: %d b: %d' %

    return response
```

- Service Client Node로부터 `request` 로 받은 정수 `a, b` 에 대한 합에 대한 결과를 `response.sum` 에 담아 return을 수행합니다.

- Service Server Node는 Topic과 동일하게 `spin` 으로 구성되어 무한정 대기하다가 Service Client Node로부터 새로운 `request` 를 받으면 위 과정을 수행하여 `response` 보내는 과정을 반복합니다.

```
def main(args=None):
    rclpy.init(args=args) # 초기화
    node = MinimalService() # MinimalService를 node라는 이름으로
```

```
    try:
        rclpy.spin(node) # 생성한 노드를 spin하여 지정된 콜백 함수 실
    except KeyboardInterrupt:
        node.get_logger().info('Keyboard Interrupt (SIGINT)')
    finally:# 종료시 (ex `Ctrl + c`)
        node.destroy_node()  # 노드 소멸
        rclpy.shutdown() # rclpy.shutdown 함수로 노드 종료


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

Clinet Node 작성

- `ros2_ws/src/py_srvcli/py_srvcli` 경로에 새로운 파이썬 스크립트 `client_member_function.py` 생성하여 아래의 코드를 작성해주세요

```
import sys ## 터미널 창으로부터 두 정수를 입력 받기

from example_interfaces.srv import AddTwoInts
import rclpy
from rclpy.node import Node


class MinimalClientAsync(Node):

    def __init__(self):
        super().__init__('minimal_client_async')

        ### ============= 클라이언트 설정 =============
        self.cli = self.create_client(
            AddTwoInts, ## 서비스 타입
            'add_two_ints') ## 서비스 명
        ### ==========================================

        while not self.cli.wait_for_service(timeout_sec=1.0):
```

```
                ## 일치하는 service client self.cli가 사용 가능한
            self.get_logger().info('service not available, wa



    ### ============= request를 보내는 함수 =============
    def send_request(self, a, b):
        self.req = AddTwoInts.Request()
        self.req.a = a
        self.req.b = b
        self.future = self.cli.call_async(self.req)
        rclpy.spin_until_future_complete(self, self.future)
        return self.future.result()
    ### ==========================================


def main(args=None):
    rclpy.init(args=args) # 초기화
    minimal_client= MinimalClientAsync()
        # MinimalClientAsync를 node라는 이름으로 생성
    try:
        response = minimal_client.send_request(int(sys.argv[1
        minimal_client.get_logger().info(
                'Result of add_two_ints: for %d + %d = %d' %
                (int(sys.argv[1]), int(sys.argv[2]), response
    except KeyboardInterrupt:
        minimal_client.get_logger().info('Keyboard Interrupt
    finally:# 종료시 (ex `Ctrl + c`)
        minimal_client.destroy_node()  # 노드 소멸
        rclpy.shutdown() # rclpy.shutdown 함수로 노드 종료


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

- `import`문의 전반적인 구조는 동일합니다.

```
import sys

from example_interfaces.srv import AddTwoInts
import rclpy
from rclpy.node import Node
```

- 한 가지 다른 점은 `import sys`를 통해 터미널 창으로부터 두 정수를 입력 받기 위해 코드가 추가되었습니다.
- 동일하게 Node를 상속 받으며 해당 노드 명의 이름은 `'minimal_client_async'` 로 설정해주었습니다.

```
class MinimalClientAsync(Node):

    def __init__(self):
        super().__init__('minimal_client_async')
        self.cli = self.create_client(AddTwoInts, 'add_two_in
        while not self.cli.wait_for_service(timeout_sec=1.0):
            self.get_logger().info('service not available, wa
```

- `create_client(<서비스 타입>, <서비스 명>)`
- service server와 <서비스 타입>, <서비스 명>이 같아야 합니다.
- `while` 에서는 일치하는 service client `self.cli`가 사용 가능한지 1초에 한번씩 확인합니다.

- `send_request` 문은 실질적으로 `request`를 보내는 함수에 해당합니다.

```
    def send_request(self, a, b):
        self.req.a = a
        self.req.b = b
        self.future = self.cli.call_async(self.req)
        rclpy.spin_until_future_complete(self, self.future)
        return self.future.result()
```

- `future` 는 특정 작업에 대한 약속을 받아내는 것으로 service의 효율성을 결정하는 부분입니다.

- `main` 문

```
def main(args=None):
    rclpy.init(args=args) # 초기화
    minimal_client= MinimalClientAsync() # MinimalClientAsync
    try:
        response = minimal_client.send_request(int(sys.argv[1
        minimal_client.get_logger().info(
                'Result of add_two_ints: for %d + %d = %d' %
                (int(sys.argv[1]), int(sys.argv[2]), response
    except KeyboardInterrupt:
        minimal_client.get_logger().info('Keyboard Interrupt
    finally:# 종료시 (ex `Ctrl + c`)
        minimal_client.destroy_node()  # 노드 소멸
        rclpy.shutdown() # rclpy.shutdown 함수로 노드 종료


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

Add an entry point

- `ros2 run` 커맨드를 통해 작성한 Service node 실행시키기 위해서는 `setup.py` 속의 `entry_points` 구역에 아래의 내용을 추가해야 합니다.

```
entry_points={
    'console_scripts': [
        'service = py_srvcli.service_member_function:main',
        'client = py_srvcli.client_member_function:main',
    ],
},
```

Build and run

- 지금까지 코드 작성하시느라 고생하셨습니다. 그럼 이제 실행해볼까요!
- 실행 과정(`ros2 run` 실행 전에 수행해야 하는 코드)
  1. 먼저 실행을 위한 경로로 이동하여 ROS2 실행 환경을 실행합니다.

     ```
     $ cd ~/ros2_ws
     $ source /opt/ros/foxy/setup.bash
     ```

  2. 그 다음에 빌드를 수행합니다.

     ```
     $ colcon build --symlink-install --packages-select py_s
     ```

  3. 마지막으로 로컬에 위치한 패키지의 환경 변수를 설정하기 위해서 setup file을 source 합니다!

     ```
     $ source install/local_setup.bash
     ```

     > 💡 install 디렉토리에 위치한 `local_setup`과 `setup`은 뭐가 다른 걸까요?
     >
     > - `local_setup` 은 내가 설치한 패키지의 환경 변수를 source 하기 위한 파일!
     > - `setup` 은 `/opt/ros/foxy` 와 같이 글로벌하게 사용되는 환경 변수도 source 합니다.
     >   즉,
     >   `source /opt/ros/foxy/setup.bash & source install/setup.bash` 과 동일합니다.

- 그럼 이제 실행해볼까요!

```
##  터미널 1

$ ros2 run py_srvcli service

[INFO] [minimal_service]: Incoming request
a: 2 b: 3
```

```
##  터미널 2

$ ros2 run py_srvcli client 2 3

[INFO] [minimal_client_async]: Result of add_two_ints: for 2 
```