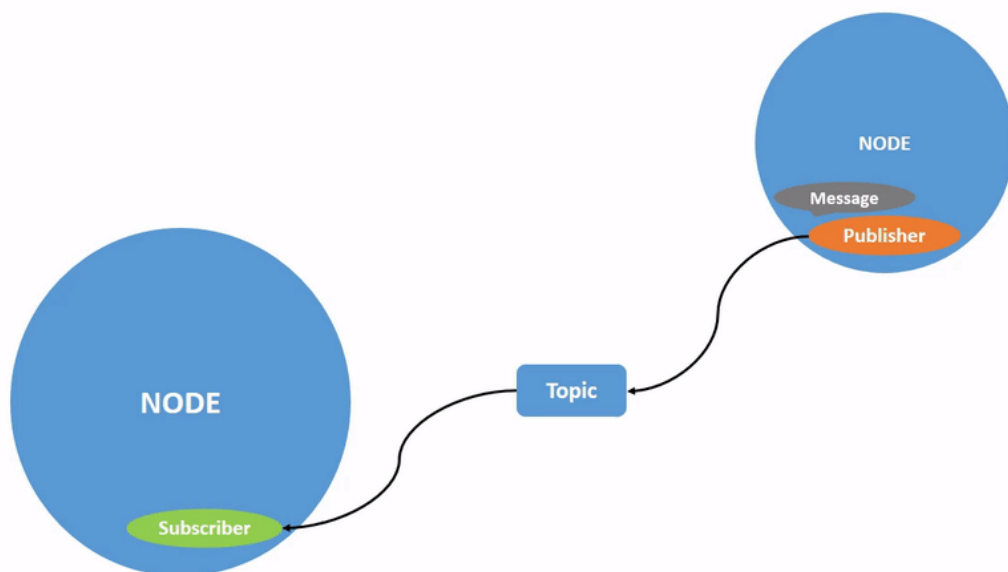




Topic 프로그래밍 - Python

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

이번 장에서는 **Publisher Node**와 **Subscriber Node** 간의 메시지 통신 **Topic**을 구현해볼 예정입니다. 해당 예제에서는 ROS 프로그래밍: Package - Python에서 제작한 Package를 활용하여 아주 간단한 예제인 ROS2 판 'Hello World'를 제작해볼 예정입니다.



- 비동기식 단방향 메시지 송수신 방식
 - msg인터페이스 형태의 메시지를 송신하는 **Publisher**와 수신하는 **Subscriber**로 구성
 - 이번 프로젝트에서는 간단한 topic 실습을 진행합니다.
 - Publisher: 1초마다 숫자 카운트한 정보를 송신
 - Subscriber: 발행된 정보를 수신하여 출력
- ▼ 결과 미리보기

```
# 터미널1 Subscriber
```

```
$ ros2 run topic_helloworld helloworld_subscriber
[INFO]: Received message: Hello World: 0
[INFO]: Received message: Hello World: 1
[INFO]: Received message: Hello World: 2
[INFO]: Received message: Hello World: 3
```

```
# 터미널 2 Publisher
```

```
$ ros2 run topic_helloworld helloworld_publisher
[INFO]: Published message: Hello World: 0
[INFO]: Published message: Hello World: 1
[INFO]: Published message: Hello World: 2
[INFO]: Published message: Hello World: 3
```

package 만들기

```
$ cd ~/ros2_ws/src
$ ros2 pkg create topic_helloworld --build-type ament_python --dependencies r
```

인터페이스

	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

Publisher Node 작성

- Publisher Node의 파이썬 스크립트는 `~/ros2_ws/src/topic_helloworld/topic_helloworld/`helloworld_publisher.py`` 라는 이름으로 소스 코드 파일을 저장하시면 됩니다.

```
$ cd ~/ros2_ws/src/topic_helloworld/topic_helloworld
```

```
$ gedit helloworld_publisher.py (VSCode를 이용하여 코드 작성하셔도 됩니다!)
```

```
#####
##### 필요한 라이브러리 호출 #####
#####
import rclpy ## 클라이언트 라이브러리 호출
from rclpy.node import Node
from rclpy.qos import QoSProfile # QoS 설정을 위해
from std_msgs.msg import String # String 메시지 인터페이스를 사용하기 위해 import

#####
##### Publisher 클래스 정의 #####
#####

class HelloworldPublisher(Node): # Node 클래스를 상속
    def __init__(self):
        super().__init__('helloworld_publisher') # 부모 클래스(Node)의 생성자를 호출
        qos_profile = QoSProfile(depth=10) # 통신상태가 원활하지 못할 경우 퍼블리시
        self.helloworld_publisher = self.create_publisher(# create_publisher함수
            String, # 토픽 메시지 타입: String,
            'helloworld', # 토픽 이름: helloworld
            qos_profile)#, QoS설정
        self.timer = self.create_timer(1, self.publish_helloworld_msg) # 콜백
        self.count = 0

    def publish_helloworld_msg(self): # 앞서 지정한 콜백 함수
        msg = String() # 메시지 타입 String으로 지정
        msg.data = 'Hello World: {0}'.format(self.count) # 표준 문자열 메시지 구조
        self.helloworld_publisher.publish(msg) # __init__ 에서 정의한 helloworld
        self.get_logger().info('Published message: {0}'.format(msg.data)) # E
        self.count += 1

#####
##### main 함수로, publisher 노드를 반복 실행 #####
#####

def main(args=None):
    rclpy.init(args=args) # 초기화
    node = HelloworldPublisher() # HelloworldPublisher를 node라는 이름으로 생성
    try:
        rclpy.spin(node) # rclpy에게 이 Node를 반복해서 실행 (=spin) 하라고 전달
    except KeyboardInterrupt: # `Ctrl + c`가 동작했을 때
        node.get_logger().info('Keyboard Interrupt (SIGINT)')
```

```

finally:
    node.destroy_node() # 노드 소멸
    rclpy.shutdown() # rclpy.shutdown 함수로 노드 종료

if __name__ == '__main__':
    main()

```

- 코드 내용에 대해서는 하나씩 알아보도록 하겠습니다.
- Python ROS2 프로그래밍을 위한 `rclpy`
- `rclpy` 의 `Node` 클래스
- 퍼블리셔의 QoS 설정을 위한 `QoSProfile` 클래스
- ROS 표준 메시지 타입 `std_msgs.msg` 모듈의 `String` 클래스

```

import rclpy
from rclpy.node import Node
from rclpy.qos import QoSProfile
from std_msgs.msg import String

```

- QoS(Quality of Service)는 ROS 2에서 표준화된 메시지 통신을 위해 사용되는 DDS(Data Distribution Service)의 “데이터 통신 옵션”에 해당합니다.
-

- 먼저 `main` 문을 먼저 살펴보겠습니다.

```

def main(args=None):
    # ROS Node가 동작하기 위해서는 현 상태에 대한 정보를 알아내는 등 초기 작업이 필요합니다.
    rclpy.init(args=args)

    # 아직 살펴보지 않았지만, 새로운 Node에 해당하는 무언가를 만들었습니다.
    node = HelloWorldPublisher()
    try:
        # rclpy에게 이 Node를 반복해서 실행 (=spin) 하라고 전달하고 있습니다.
        rclpy.spin(node)
    except KeyboardInterrupt:
        # Node의 실행 중 여러 상태들에 대한 로그
        # 해당 예제에서는 KeyboardInterrupt -> ctrl + c가 동작했을 때, 로그를 출력합니다.
        node.get_logger().info('Keyboard Interrupt (SIGINT)')
    finally:
        # 사용을 마친 node를 종료합니다.
        node.destroy_node()
        # 마찬가지로 사용을 마친 Node는 종료됩니다.
        rclpy.shutdown()

```

- `rclpy.spin` 의 기본 동작 방식은 **반복적으로 Node를 동작** 시키는 것입니다.

▼ 하여 Node를 특정 시간 만큼 반복 하도록 설계하고 싶다면 아래와 같이 코드를 짤 수 있습니다.

```
start_time = node .get_clock().now().to_msg().sec
clock_now  = start_time
time_delta = 0

# 5초 동안만 실행
while (clock_now - start_time) < 5:
    # 단 한 번만 spin시키기 위해 spin_once가 사용되었습니다.
    rclpy.spin_once(node )
    clock_now = cmd_vel_publisher.get_clock().now().to_msg().sec

time_delta = clock_now - start_time
print(f'{time_delta} seconds passed')
```

- 해당 노드에서는 `HelloWorldPublisher` 로 `Node` 클래스를 상속하여 사용할 예정입니다.

```
class HelloWorldPublisher(Node):

    def __init__(self):
        super().__init__('helloworld_publisher')
        qos_profile = QoSProfile(depth=10)
        self.helloworld_publisher = self.create_publisher(
            String,
            'helloworld',
            qos_profile)
        self.timer = self.create_timer(1, self.publish_helloworld_msg)
        self.count = 0
```

- `super().__init__('helloworld_publisher')` 를 이용하여 Node 클래스의 생성자를 상속 받고 노드 이름을 'helloworld_publisher'로 지정합니다.
 - 상속이란, 마치 유산을 상속하듯이 **상위 Class가 구현해 둔 모든 기능을 추가 개발 없이 동일하게 사용할 수 있다**는 특징과 더불어, 상속 받은 Class 외에 **자신만의 기능을 추가할 수 있다**는 뜻입니다.
- `qos_profile = QoSProfile(depth=10)` 는 버퍼를 10개 까지 저장하라는 설정 입니다.
- `self.helloworld_publisher = self.create_publisher(String, 'helloworld', qos_profile)`
 - `self.create_publisher(<메세지 타입>, <토픽 이름>, <QoS 타입>)`
- `self.timer = self.create_timer(1, self.publish_helloworld_msg)`
 - 어느 정도의 주기로 publish 할 것 인지를 선택합니다.
 - 해당 예제에서는 1초 마다 콜백 함수 `self.publish_helloworld_msg` 를 퍼블리시 합니다.

- 1초 간격을 갖고 반복적으로 실행되는 콜백 함수 `self.publish_helloworld_msg` 를 살펴볼까요?

```
def publish_helloworld_msg(self):
    msg = String()
    msg.data = 'Hello World: {0}'.format(self.count)
    self.helloworld_publisher.publish(msg)
    self.get_logger().info('Published message: {0}'.format(msg.data))
    self.count += 1
```

- `msg = String()` 표준 문자열 메시지 구조에 따라 `msg.data` 에 문자열을 담아줍니다.
▼ `std_msgs/msg/String`

`std_msgs/msg/String` Message

File: `std_msgs/msg/String.msg`

Raw Message Definition

```
# This was originally provided as an example message.
# It is deprecated as of Foxy
# It is recommended to create your own semantically meaningful message.
# However if you would like to continue using this please use the equivalent in example_msgs.

string data
```

Compact Message Definition

```
string data
```

- `self.helloworld_publisher.publish(msg)` 을 통해 메시지를 퍼블리시 합니다.

Subscriber Node작성

- Subscriber Node의 파이썬 스크립트는 `~/ros2_ws/src/topci_helloworld/topci_helloworld/`` 폴더에 ``helloworld_subscriber.py`` 라는 이름으로 소스 코드 파일을 저장하시면 됩니다.

```
#####
##### 필요한 라이브러리 호출 #####
#####
import rclpy
from rclpy.node import Node
from rclpy.qos import QoSProfile # QoS 설정을 위해
from std_msgs.msg import String # String 메시지 인터페이스를 사용하기 위해 import

#####
##### Subscriber 클래스 정의 #####
#####
```

```

class HelloworldSubscriber(Node): # Node 클래스를 상속
    def __init__(self):
        super().__init__('Helloworld_subscriber') # 부모 클래스(Node)의 생성자를
        qos_profile = QoSProfile(depth=10) # 통신상태가 원활하지 못할 경우 퍼블리시
        self.helloworld_subscriber = self.create_subscription( # create_subsc
            String, # 토픽 메시지 타입
            'helloworld', # 토픽 이름
            self.subscribe_topic_message, # 수신 받은 메시지를 처리할 콜백함수 **
            qos_profile) # QoS: qos_profile

    def subscribe_topic_message(self, msg): # 앞서 지정한 콜백 함수 정의.
        self.get_logger().info('Received message: {0}'.format(msg.data)) # E

#####
##### main 함수로, Subscriber 노드를 반복 실행 #####
#####
def main(args=None):
    rclpy.init(args=args) # 초기화
    node = HelloworldSubscriber() # HelloworldSubscriber를 node라는 이름으로 생성
    try:
        rclpy.spin(node) # rclpy에게 이 Node를 반복해서 실행 (=spin) 하라고 전달
    except KeyboardInterrupt: # `Ctrl + c`가 동작했을 때
        node.get_logger().info('Keyboard Interrupt (SIGINT)')
    finally:
        node.destroy_node() # 노드 소멸
        rclpy.shutdown() # rclpy.shutdown 함수로 노드 종료

if __name__ == '__main__':
    main()

```

- 서브스크라이버 노드에서도 `Node` 클래스를 상속 받습니다.

```

class HelloworldSubscriber(Node):

    def __init__(self):
        super().__init__('Helloworld_subscriber')
        qos_profile = QoSProfile(depth=10)
        self.helloworld_subscriber = self.create_subscription(
            String,
            'helloworld',
            self.subscribe_topic_message,
            qos_profile)

```

- `super().__init__('Helloworld_subscriber')` 서브스크라이버 노드의 이름은 `'Helloworld_subscriber'` 로 지정합니다.
- `create_subscription(String, 'helloworld', self.subscribe_topic_message, qos_profile)`
 - `create_subscription(<메세지 타입>, <토픽 이름>, <수신 받은 메시지를 처리할 콜백함수>, <QoS 설정>`
- 다음은 위에서 지정한 콜백 함수인 `subscribe_topic_message` 함수입니다.

```
def subscribe_topic_message(self, msg):
    self.get_logger().info('Received message: {0}'.format(msg.data))
```

- 서브스크라이브한 메시지는 String타입으로 msg라는 이름을 사용하며 받은 메시지는 msg.data에 저장하게 합니다.
- 해당 예제 코드는 'Hello World: 1'과 같은 메시지를 서브스크라이브한 msg.data에게 get_logger의 info 함수를 이용하여 서브스크라이브 된 메시지를 콘솔창에 출력하는 것입니다.
- 마지막으로 `main` 함수에서는 `HelloworldSubscriber` 을 node로 선언하여 사용하는 것 외에는 퍼블리셔 노드의 `main` 함수와 동일합니다.

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

Add an entry point

- `ros2 run` 커맨드를 통해 작성한 service node 실행시키기 위해서는 `setup.py` 속의 `entry_points` 구역에 아래의 내용을 추가해야합니다.

```
entry_points={
    'console_scripts': [
        'helloworld_subscriber = topic_helloworld.helloworld_subscriber:main'
        'helloworld_publisher = topic_helloworld.helloworld_publisher:main'
    ],
},
```


Build and run

- 이제 패키지를 build하고 실행해보도록 하겠습니다
- 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 topic_helloworld
Starting >>> topic_helloworld
Finished <<< topic_helloworld[0.66s]

Summary: 1 package finished [0.87s]
```

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.

실행

터미널1

```
$ ros2 run topic_helloworld helloworld_subscriber
[INFO]: Received message: Hello World: 0
[INFO]: Received message: Hello World: 1
[INFO]: Received message: Hello World: 2
[INFO]: Received message: Hello World: 3
[INFO]: Received message: Hello World: 4
[INFO]: Received message: Hello World: 5
```

터미널 2

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

```
$ ros2 run topic_helloworld helloworld_publisher
[INFO]: Published message: Hello World: 0
[INFO]: Published message: Hello World: 1
[INFO]: Published message: Hello World: 2
[INFO]: Published message: Hello World: 3
[INFO]: Published message: Hello World: 4
[INFO]: Published message: Hello World: 5
```

이번 장에서는 Topic을 이용하여 ROS2 판 'Hello world'를 작성해보았습니다. 다음 장에서는 단발적인 통신에 활용되는 동기식 양방향 메세지 통신 **Service**에 대해 다뤄볼 예정입니다.