Week10

1. Log

프로그래밍 언어에서 로그는 프로그램을 개발하는 과정에서 개발자가 프로그램을 검토하는 용도로 사용되고, 간단한 디버깅 툴로도 활용된다. Ros2 에서는 logger와 logging 라이브러리를 사용

한다.

```
# 로그가 저장되는 디렉터리
$ ls ~/.ros/log
```

Foxy 버전에서는 로그가 저장되는 경로를 변경할 수 없지만, 다음 버전인 Galactic부터는 이 기능을 지원한다.

1.2.2. 로그 수준

Ros2 에서 로그 수준은 총 5가지(DEBUG, INFO, WARN, ERROR, FATAL)이다. 로그 수준을 나누는 기준은 개발자마다 다르다. 구딩 나눠보자면

1. Programmatically

: 가장 기본적인 방법은 코드에 명시하는 것이다.

2. Externallly

rqt_logger_level 노드를 통해 서비스 통신으로 특정 노드의 로그 레벨을 런타임에서 변경할 수 있는 기능을 제공한다.

3. Command line

노드를 실행할 때 인자를 통해 해당 노드의 로그 수준을 지정할 수 있다. 다음과 같이 데모 코드를 실행하는 명령어 뒤에 실행 인자로 원하는 로그 수준을 적어주면된다.

```
$ ros2 run logging_demo logging_demo_main --ros-args --log-level debug
```

rclpy 예제를보자. DEBUG 수준의 로그를 보기 위해 실행 시 인자를 추가해보자.

```
$ ros2 run logging_rclpy_example logging_example --ros-args --log-level debug
```

2. ROS2 CLI

ROS2 CLI(Command Line Interface) 명령어, 사용법, 종류와 각 verbs, sub-verbs, options에 대한 내용의 예제를 보자

ros2 run

run은 특정 패키지의 특정 노드를 실행하는 명령어이다. 일반적으로 1개의 노드를 의미하지만 executable에 따라 복수 노드도 실행 가능

```
$ ros2 run turtlesim turtlesim_node
$ ros2 run turtlesim turtle_teleop_key
```

ros2 launch

launch는 특정 패키지의 특정 런치 파일을 실행한다. 이를 통해 설정만 변경하여 노드를 실행시킬 수있고, 복수 개의 노드를 실행시키거나 런치에서 또 다른 패키지의 다른 런치파일을 불러와 실행시킬 수도 있다.

```
$ ros2 launch demo_nodes_cpp talker_listener.launch.py
[INFO] [launch]: All log files can be found below /home/rodel/.ros/log/2023-09-16-14-21-53-611248-rodel-24792
```

```
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [talker-1]: process started with pid [24794]
[INFO] [listener-2]: process started with pid [24796]
[talker-1] [INFO] [1694841714.681130878] [talker]: Publishing: 'Hello World: 1'
[listener-2] [INFO] [1694841714.681967800] [listener]: I heard: [Hello World: 1]
[talker-1] [INFO] [1694841715.681125691] [talker]: Publishing: 'Hello World: 2'
[listener-2] [INFO] [1694841715.681373891] [listener]: I heard: [Hello World: 2]
[talker-1] [INFO] [1694841716.681172727] [talker]: Publishing: 'Hello World: 3'
```

ros2 pkg

지정 패키지의 정보를 얻거나 패키지를 생성하는데 사용

```
$ ros2 pkg create my_first_ros_rclpy_pkg --build-type ament_python --dependencies rclpy std_msg
: ament_python 빌드 형태에 rclpy, std_msgs 패키지에 의존성을 가진 패키지를 생성
```

ros2 topic

토픽의 구성, 대역폭, 지연시간, 인터페이스 형태 등의 정보를 얻거나 특정 토픽을 송신 및 수신하는데 사용되는 명령어다

```
$ ros2 topic echo /turtle1/cmd_vel
```

ros2 service

서비스의 정보를 얻거나 직접 서비스 요청을 테스트해볼 수 있는 명령어다

```
$ ros2 service list -t
```

ros2 action

액션의 정보를 얻거나 직접 액션 목표 전달 테스트

```
$ ros2 action info /turtle1/rotate_absolute
$ ros2 action list
```

ros2 interface

토픽/서비스/액션에서 사용하는 인터페이스의 정보를 얻는 데 사용되는 명령어

```
$ ros2 interface list
$ ros2 interface package turtlesim
$ ros2 interface packages
```

ros2 param

파라미터의 정보를 확인하고 설정하거나 읽어오는 등의 명령어

```
$ ros2 param delete /turtlesim background_b
: turtlesim 노드의 background-B 파라미터를 삭제한다
$ ros2 param list
: 사용가능한 파라미터 목록 확인
$ ros2 param set /turtlesim background_b 148
: turtlesim 노드의 background_b 파라미터를 148이라는 값으로 설정
```

ros2 bag

토픽을 저장하거나 재생할 때 사용한다

```
$ ros2 bag record /turtle1/cmd_vel
:원하는 토픽 저장
$ ros2 bag info rosbag2_2021_01_25-08_03_24
저장된 rosbag 파일의 정보를 확인한다
```

```
$ ros2 bag play rosbag2_2021_01_25-08_03_24
: 지정한 rosbag 파일 재생한다
```

ros2 daemon

데몬은 Ros2 도구들의 빠른 실행을 위해 도입된 툴이다. 이를 사용해 Ros2 그래프에 대한 정보를 저장하면 매번 쿼리를 요청하는 것보다 빠르게 응답받을 수 있다.

ros2 doctor

ROs2 설정 및 네트웤,. 패키지 버전, RMW 등과 같은 Ros2 개발환경의 잠재적 문제를 확인하는 명령어

```
$ ros2 doctor hello
MULTIMACHINE COMMUNICATION SUMMARY
Topic: /canyouhearme, Published Msg Count: 10
Subscribed from:
                Hostname            Msg Count /1.0s
Multicast Group/Port: 225.0.0.1/49150, Sent Msg Count: 10
Received from:
                Hostname            Msg Count /1.0s
------------------------------------------------------------
```

2.3 ROS2 CLI의 | 빠른 실행

홈폴더의 .bashrc 파일에 자주사용되는 ROS2 CLI 명령어를 단축 명령어로 지정해두면 편리하다

```
$ source ~/.bashrc
```

2.4 CLI 명령어에서 ROS argument 사용하기

주로 run 또는 launch 명령어와 함께 사용된다. —ros-args 옵션을 통해 지정한다

```
$ ros2 run 패키지이름 노드이름 --ros-args (ROS argument)
$ ros2 param dump /turtlesim
```

3. Intra-process communication

일반적으로 ROS를 사용하여 개발된 로봇은 복수 개의 노드를 이용한다. 예를 들어 라이다 데이터를 다루는 노드, 모터를 제어하는 노드, 로봇의 위치를 추종하는 노드, 경로를 생성하는 노드 등이 한 번에 실행된다. 그러나 복수 개의 노드를 실행하는 건 성능 저하로 이어질 수 있다.

Intra-process communication

Ros2 에서는 복수 개의 노드 사용 시 시스템의 성능 저하를 해결하기 위해 IPC를 사용한다. 복수개의 노드를 단일 프로세스에서 동작되도록 하고, 고정된 메모리 공간에 접근해서 메모리 복사가 이루어지지않는다.

IPC 는 rclcpp만 지원한다. unique_ptr을 통해 전달하고 싶은 메시지의 소유권을 유일하게 만들고 executor를 통해 단일 프로세스에서 두 노드가 동작되도록 하는 zero-copy 통신 방법을 사용한다.

만약 신호처리 알고리즘을 위한 노드가 여러 개로 나뉘어져 있다면 해당 데이터는 복수개의 퍼블리셔와 서브스크라이버를 거치면서 여러 번 메모리 복사가 일어난다. 하지만 IPC를 이용하면 매우 효율적으로 고정된 메모리 공간을 사용할 수 있다.

4. QoS(Quality of Service)

Qos 옵션 6가지(History, Reliability, Durability, Deadline, Lifespan, Liveliness)

1. History : 데이터를 몇개나 보관할지 결정하는 옵션
2. Reliability : 데이터 전송에 있어 속도를 우선시 하는지 신뢰성을 우선시 하는지 결정
3. Durability : 데이터를 수신하는 서브스크라이버가 생성되기 전의 데이터를 사용할지 폐기할지에 대한 옵션

4. Deadline : 정해진 주기 안에데이터가 발신 및 수신되지 않을 경우 EventCallback을 실행
5. LifeSpan : 정해진 주기 안에서 수신되는 데이터만 유효판정하고 나머지는 삭제
6. Liveliness : 정해진 주기 안에서 노드 혹은 토픽의 생사확인하는 QoS 옵션

Topic

Topic의 기본 QoS 설정은 rclpy와 rclcpp에서 특별한 설정이 없는 한 RMW QoS Profile의 Default 설정을 사용한다. 즉 Reliability은 Reliable이고 History 는 Keep_last에 Depth는 10이며, Durability는 Voliatile이다.

Service

특별한 경우 제외하고는 기본 QoS를 사용한다.

Action

액션은 토픽과 서비스를 모두 사용하는 복합 형태다.

4.3 실습

History

helloworld_publisher와 hellloworld_subscriber를 보자

```
class HelloWorldPublisher(Node):
    def __init__(self):
        super().__init__('helloworld_publisher')   # 부모 클래스(Node)의 생성자를 호출하고 노드 이름을 'helloworld_publisher'로 지정한다
        qos_profile = QoSProfile(depth = 10)     # 퍼블리셔의 Qos 설정 위해 QoSProfile호출하고 기본 depth를 10으로 설정 : 이는 통신상태가 원할하?
```

History를 Keep_last로 변경하면 다음과 같다. 지정한 depth인 10개의 메시지만 보관된다.

```
qos_profile = QoSProfile(
  reliability = QoSReliabilityPolicy, RELIABLE,
history = QoSHistoryPolicy.KEEP_Last, depth = 10,
durabiltity = QoSDurabiltiyPolicy.TRANSIENT_LOCAL)
```