

ROS DAY3

과제 1번 보고서

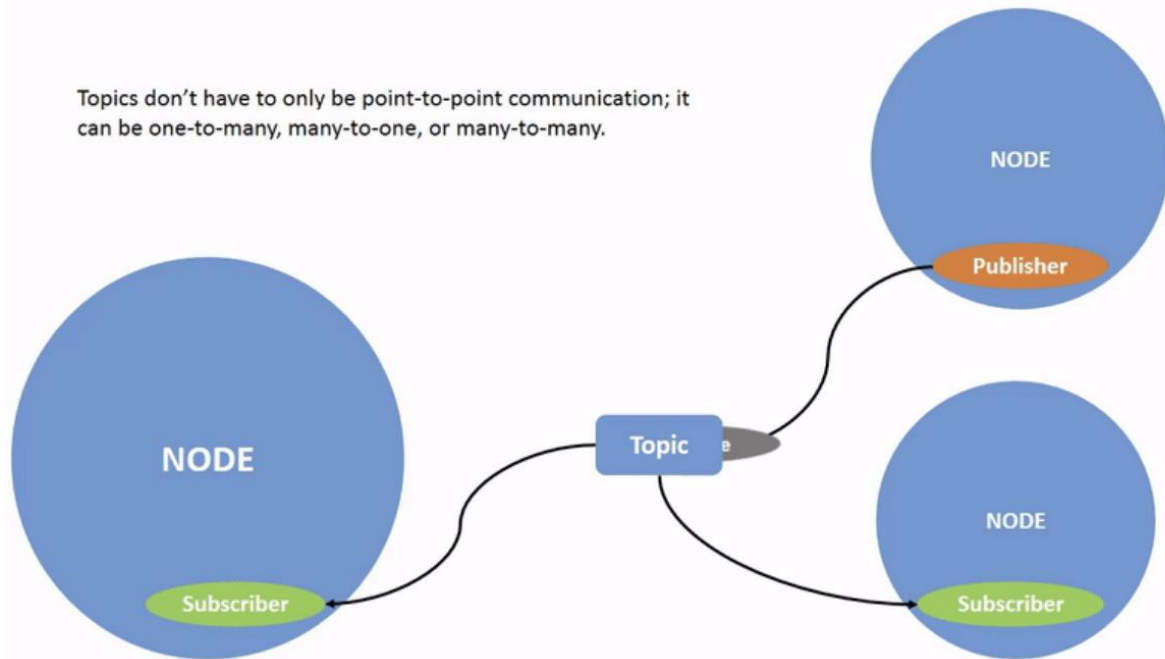
2023741024
로봇학부 박건후

목차

- (1) 토픽 통신
- (2) 서비스 통신
- (3) parameters
- (4) actions
- (5) 기본적인 publish, subscribe 복습
- (6) 서비스 통신 코드 학습

(1) 토픽 통신

토픽 통신은 정말로 메시지를 교환하는데 통로 같은 역할을 합니다. 노드는 여러 토픽에 메시지를 다양하게 보낼 수 있고 이렇게 publish한 것을 메시지를 받는 서브스크라이버들은 동시에 이 메시지를 받을 수 있습니다.



그래서 위의 사진처럼 토픽 통신은 일대일 통신으로 보아야 할 것이 아닌 1대 다수의 통신인 것입니다. 마치 카카오톡의 단체방처럼 한 명이 문자를 올리면 모두가 이 메시지 정보를 받을 수 있는 것과 같습니다. 이처럼 토픽이라는 것이 버스 역할을 해주고 있는 것입니다. 그리고 이 토픽 통신이 이루어지는 것을 직접 시각화하며 확인하기 위해서는 `rqt_graph`라는 것을 사용할 수 있습니다. `rqt_graph`는 토픽명, 노드명 등을 적어놓고 화살표 연결을 통해 데이터가 오고 가는 방향을 시각화해줍니다. 아래 사진이 토픽 통신에서의 그래프의 예시입니다.

위 사진에서는 `/turtlesim`노드와 `/teleop_turtle`간의 토픽을 통해 통신하는 것을 나타낸 그래프입니다. 여기서 `teleop_turtle`이 `turtle1/cmd_vel`이라는 토픽으로 데이터를 publish하고 `/turtlesim` 노드가 그 데이터를 토픽으로부터 받는 것입니다. 이처럼 `rqt_graph`는 나중에 더 복잡한 노드 간의 통신에

서도 쉽게 관계를 확인할 수 있게 해주는 도구인 것입니다.

이제 터미널에 토픽 관련 명령어가 뭐가 있는지 보겠습니다. `ros2 topic list` 혹은 `ros2 topic list -t`를 터미널에 입력하면 그에 해당하는 토픽명들 그리고 `-t`를 추가적으로 입력했다면 토픽 타입까지 알 수 있습니다.

`ros2 topic echo` 토픽이름을 작성하게 되면 발행된 데이터들을 볼 수 있습니다.

`ros2 topic info` 토픽이름을 작성할 경우 토픽명에서 사용하는 토픽 타입이 나오게 되고 여기서 publish하는 노드와 subscribe 하는 노드의 개수 또한 알 수 있습니다.

`ros2 interface show` 토픽 타입을 작성하게 되면 입력해야 할 매개변수들을 알려줍니다. `teleop`을 사용하여 정보를 보내고자 할 때 linear, angle 값을 보내야 하는데 이 때 어느 정도 값으로 보낼지 어떤 형식으로 보내야 할지 알려줍니다. 아래 사진이 그 해당 부분입니다.

```
# This expresses velocity in free space broken into its linear and angular parts.
Vector3 linear
  float64 x
  float64 y
  float64 z
Vector3 angular
  float64 x
  float64 y
  float64 z
```

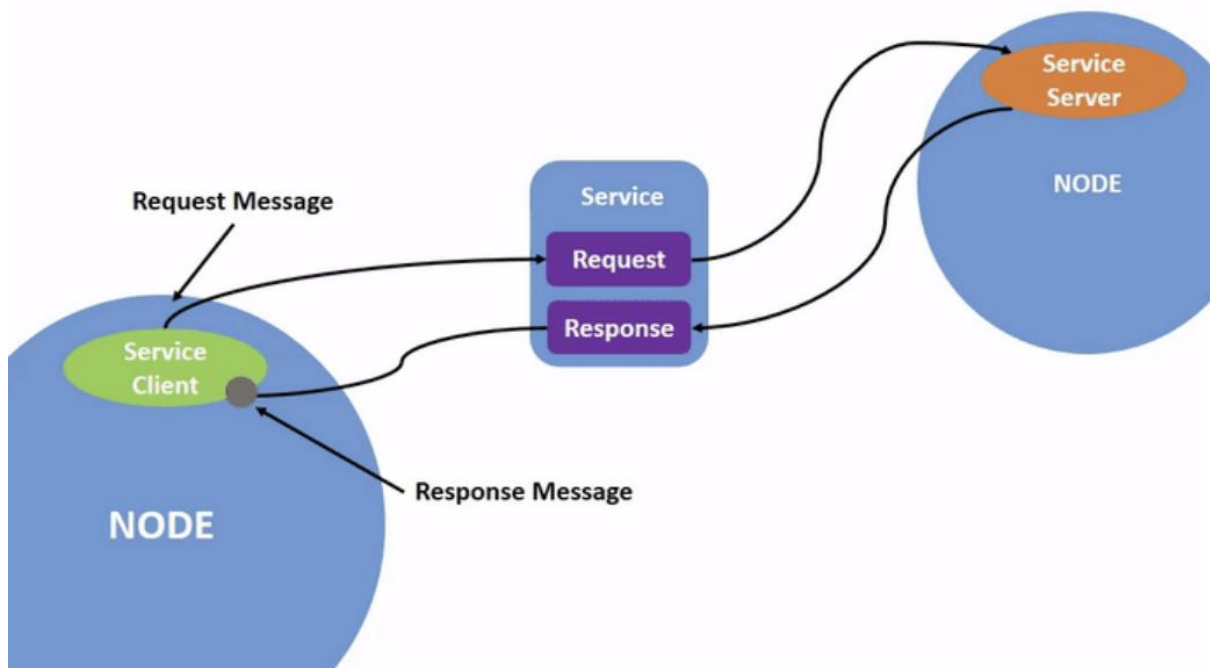
`ros2 topic hz` 토픽명을 작성하면 특정 토픽이 얼마나 자주 발행되는지 확인할 수 있습니다.

```
average rate: 59.354
min: 0.005s max: 0.027s std dev: 0.00284s window: 58
```

위 사진을 보시면 평균적으로 초당 59번 메시지가 발행되고 메시지 간격의 최대, 최소 시간, 표준편차, 샘플 개수로 정보를 나타냄을 볼 수 있습니다.

(2) 서비스 통신

서비스 통신은 다른 방식의 통신 방식입니다. 서비스 통신은 call and response 모델을 기반으로 합니다. 여기서 call and response란 하나의 노드가 다른 노드에게 요청을 보내면, 그 노드는 해당 요청을 처리한 후 응답을 돌려주는 방식입니다. 이는 클라이언트-서버 구조와 유사하며 요청이 있을 때만 통신이 발생합니다. 토픽은 데이터를 지속적으로 발행하고 이를 구독하는 노드들이 데이터를 받는 것인 반면, 서비스 통신은 요청하여 그 요청된 것을 처리한 후 관련 데이터를 제공해주는 것입니다.



그래서 위 사진을 보시면 노드끼리 통신을 할 때 서버, 클라이언트라는 개념이 생기게 되고 클라이언트는 서비스라는 버스에게 요청을 보내고 서비스는 그 요청을 서비스 라는 버스로부터 받고 요청을 처리한 후 다시 버스를 통해 전달하는 형태입니다. 이 또한 일대일 통신이 아닌 클라이언트가 여러이고 서버가 하나로써 1대 다수 통신을 할 수 있습니다.

서비스 또한 토픽처럼 타입이 있습니다. 그 타입을 통해 데이터를 구성해서 전달하는 것입니다. 터미널에서의 서비스 관련 명령어 또한 토픽에서의 관련 명령어와 같습니다. 터미널에서 토픽 통신을 보낼 때 `ros2 topic pub` 토픽이름 토픽타입 매개변수를 입력한 것처럼 서비스 통신할 때도 이와 유사한 방식으로, `ros2 service call` 서비스이름 서비스타입 매개변수로 입력하

면 됩니다.

제가 이번에 배운 명령어를 말씀드리겠습니다.

```
ros2 service find <type_name>
```

위 명령어 토픽/서비스이름을 입력하면 그 토픽/서비스에서 사용하는 토픽/서비스 타입을 알 수 있게 됩니다. 그렇게 되면 위에서 말씀드린 interface show를 사용하면 토픽/서비스 타입의 매개변수를 구하는 명령어를 통해 통신 때 보내야 할 모든 값들에 대해 얻을 수 있습니다. 반대로 토픽/서비스 타입을 입력하면 그 토픽/서비스 타입을 사용하는 토픽/서비스 이름을 알 수 있습니다. 이렇게 반대 방향으로 정보를 또한 알 수 있습니다.

(3) 파라미터

각 노드는 자신의 파라미터를 갖고 있습니다. 이는 노드가 시작될 때나 실행 중에 사용할 수 있는 설정 값입니다. 이 매개변수의 값은 int, float 등의 타입을 가집니다. 예를 들어 터틀심의 백그라운드 색의 r, g, b 값 설정 등이 있습니다.

```
ros2 param dump <node_name>
```

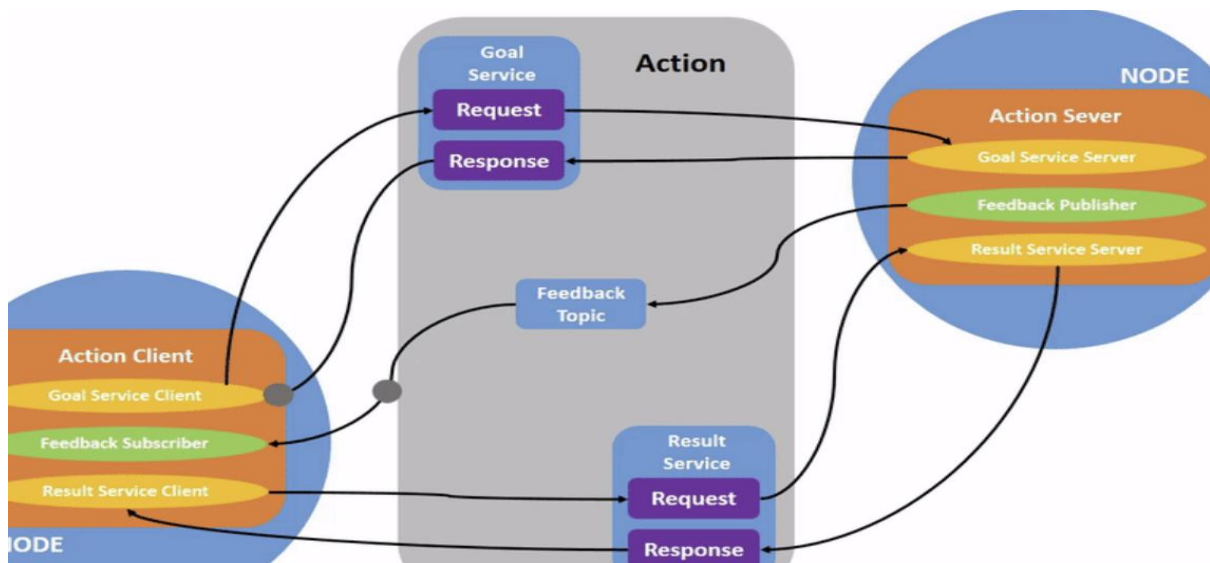
이 코드를 통해 해당 노드의 모든 파라미터 변수를 볼 수 있고

```
$ ros2 param get <node_name> <parameter_name>
```

위 사진의 명령어로 param get 명령어를 받아올 수도 있고 get 자리에 set을 넣어서 파라미터 변수 값을 설정할 수도 있습니다.

(4) 액션

액션은 ros2에서 하나의 통신 방식입니다. 그리고 이는 goal, feedback, result로 나뉩니다. 이 기능은 통신 중 차단될 수 있다는 것을 제외하면 서비스 통신과 유사합니다. goal, result가 그에 해당됩니다. 그리고 feedback 부분에서는 토픽 통신 방식을 사용하여 연속적인 데이터 교환을 가능하게 한 구조입니다.



위 사진처럼 액션에도 클라이언트와 서버가 있습니다. 클라이언트는 액션 서버에게 goal을 보냅니다. 그리고 서버는 goal을 받고 그에 대한 feedback과 result를 제공합니다. 이에 대한 예시로 터틀심의 teleop_key가 있습니다. teleop_key 노드를 실행시키면

```
Use arrow keys to move the turtle.  
Use G|B|V|C|D|E|R|T keys to rotate to absolute orientations. 'F' to cancel a rotation.
```

이런 문구가 뜨게 되는데 해당 키보드를 입력하면 goal을 action server에서 보내게 되고 서버에서 이를 처리하면 그 처리한 결과를 띄우게 됩니다. 그렇게 될 시 아래 사진과 같은 문구가 뜨게 됩니다.

```
[INFO] [turtlesim]: Rotation goal canceled
```

아래의 사진은 제가 직접 사용해보고 액션의 통신을 한 번 확인해본 사진입니다.

```
geonhu@PGH:~$ ros2 run turtlesim turtlesim_node
INFO [1758008160.953399067] [turtlesim]: Starting turtlesim with node name /turtlesim
INFO [1758008160.955951248] [turtlesim]: Spawning turtle [turtle1] at x=[5.54445], y=[5.544445], theta=[0.000000]
INFO [1758008227.746907355] [turtlesim]: Rotation goal completed successfully
INFO [1758008232.546501033] [turtlesim]: Rotation goal completed successfully
INFO [1758008237.091100763] [turtlesim]: Rotation goal completed successfully
INFO [1758008246.802872397] [turtlesim]: Rotation goal completed successfully

geonhu@PGH:~$ ros2 run turtlesim turtle_teleop_key
Reading from keyboard
.....
Use arrow keys to move the turtle.
Use G|B|V|C|D|E|R|T keys to rotate to absolute orientations. 'F' to cancel a rotation.
'Q' to quit.
```

이처럼 액션 클라이언트인 teleop과 액션 서버인 turtlesim이 액션 통신을 하는 것을 확인할 수 있습니다.

ros2 topic info처럼 action info도 가능한데

```
Action: /turtle1/rotate_absolute
Action clients: 1
    /teleop_turtle
Action servers: 1
    /turtlesim
```

turtlesim에서 ros2 action info를 실행한 경우 클라이언트, 서버 각각 하나씩 있음을 확인할 수 있습니다.

```
ros2 action send_goal <action_name> <action_type> <values>
```

위의 문장은 터미널에서 goal을 보낼 때 사용하는 명령어입니다. 이전 토픽, 서비스와 같이 이름, 타입, 매개변수로 이루어져 있음을 재차 확인할 수 있었습니다

```
Goal accepted with ID: f8db8f44410849eaa93d3feb747dd444
```

```
Result:
delta: -1.568000316619873
```

그렇게 되면 goal을 받았다는 문구와 함께 result값이 나온 것을 확인할 수 있습니다.

그리고 --feedback을 ros2 action send_goal 명령어 맨 마지막에 추가하게 되면 이 goal의 피드백도 실시간으로 토픽 통신의 방식으로 받아볼 수 있습니다.

(5) 기본적인 publish, subscribe 복습

```
rc1cpp::Publisher<std_msgs::msg::String>::SharedPtr publisher_;
```


이렇게 Publisher의 SharedPtr 포인터 멤버를 사용하여 publish 해줄 포인터를 선언하고

```
publisher_ = this->create_publisher<std_msgs::msg::String>("topic", 10);
```

그리고 생성자에서는 create_publisher라는 헬퍼함수를 사용하여 이렇게 Publisher객체를 생성하고 이에 대한 포인터를 초기화해줍니다. 이 때 토픽명을 헬퍼함수에서 정하게 됩니다. 그리고 전할 정보, 데이터에 대한 타입을 템플릿 매개변수에 넣어줘서 명시해줘야 합니다. 그 후 아래 사진처럼 Publisher 클래스의 멤버함수를 사용하여 message라는 std_msgs::msg::String이라는 객체를 인수에 넣어 전달하는 것입니다.

```
publisher_->publish(message);
```

```
rclcpp::Subscription<std_msgs::msg::String>::SharedPtr subscription_;
```

서브스크라이버를 만들 때에도 Subscription 클래스를 사용하여 포인터를 생성하고

```
subscription_ = this->create_subscription<std_msgs::msg::String>("topic", 10, std::bind(&MinimalSubscriber::topic_callback, this, _1));
```

위 사진처럼 헬퍼함수로 생성함과 동시에 토픽이름을 동일하게 작성해주고 메시지를 받았을 때 그 때마다 해야 할 처리를 사용자 정의할 수 있도록 바인딩을 하여 콜백 함수 내에서 원하는 방식으로 데이터를 다룰 수 있습니다.

```
find_package(rclcpp REQUIRED)
find_package(std_msgs REQUIRED)
```

그리고 CMakeList 작성하는 방법을 보겠습니다. rclcpp, std_msgs의 내용을 사용하므로 이를 cmake가 기본적으로 찾는 것부터 해줘야 합니다.

```
add_executable(talker src/publisher_member_function.cpp)
ament_target_dependencies(talker rclcpp std_msgs)
```

이 부분에서는 talker라는 실행파일 이름을 설정하는 부분입니다. 실행 파일 이름을 설정하고 그 실행파일에 실행시킬 파일들을 여기에 작성해주면 됩니다. 위의 사진의 경우에는 하나의 cpp파일밖에 없지만 구현 파일, main파일 등등을 같이 사용하여 실행파일을 구성해야 할 경우 띄어쓰기로 구분을 지어서 여러 개를 작성하면 됩니다. 이 add_executable을 여러 개 사용하게 되는 경우도 있는데 하나의 패키지에 여러 실행 파일, 실행 노드들이 있기 때문입니다. 주로 노드 단위로 프로그램을 구성하기 때문에 하나의 패키지에 여러 publish하는 노드, subscribe하는 노드, 작업, 연산을 처리하는 노드 등

등끼리 묶어서 사용하고, 실행시키기 위해서는 각각의 실행파일을 만들고 각각의 터미널에서 이를 만드는 것이 좋습니다.

ament_target_dependencies에도 talker라는 실행파일 이름을 똑같이 써줘야 합니다. 이때 이 실행 파일이 어떤 패키지를 필요로 하는지에 대한 의존성을 작성하는 부분입니다. 이 부분은 말그대로 의존성이라는 것으로 rclcpp, std_msgs를 사용하고 그 기능에 의존하는 파일이기에 이 의존성을 작성하게 될 때 사용하며 의존하지 않는데도 추가로 작성하면 에러가 나는 경우가 많습니다.

```
install(TARGETS
  talker
  DESTINATION lib/${PROJECT_NAME})
```

install 설정하는 부분에서도 talker라는 실행 파일명을 이번에도 똑같이 작성해줘야 합니다. 그리고 이 부분에서 실제 실행파일이 생성됩니다.

(6) 서비스 통신 코드 학습

```
void add(const std::shared_ptr<example_interfaces::srv::AddTwoInts::Request> request,
```

이 부분은 서비스 통신을 하기 위해 msg파일에서 생성한 것들을 hpp로 인클루드하고 인클루드한 것으로부터 나온 타입에 대해 스코프 지정 연산자(::)를 사용하여 Request 타입을 생성한 것입니다. 이렇게 하면 shared_ptr이기에 request를 '->'연산자를 통해 Request 객체를 가리켜서 해당 멤버에 접근할 수 있게 되는 것입니다. 이를 터틀심에 적용하게 되면 이전에 수행했던 과제 3번의 경우에도 거북이가 그리는 선의 색깔, 두께를 바꾸는 것의 서비스 타입이 turtlesim/srv/set_pen이어서

turtlesim::srv::SetPen::Request으로 make_shared를 하여 해당 요청하는 객체를 생성해서 파라미터에 접근하여 값을 바꾸는 작업을 할 수 있었습니다. 이렇게 파라미터를 설정한 후에는 이전에 설정한 rclcpp::Client 객체를 통해 async_send_request(요청 객체)를 호출하여 해당 요청을 받고 서버가 이에 응답하여 파라미터 값을 바꿔 그림을 그리게 되는 것입니다.

```
rcldcpp::Client<example_interfaces::srv::AddTwoInts>::SharedPtr client =
    node->create_client<example_interfaces::srv::AddTwoInts>("add_two_ints");
```

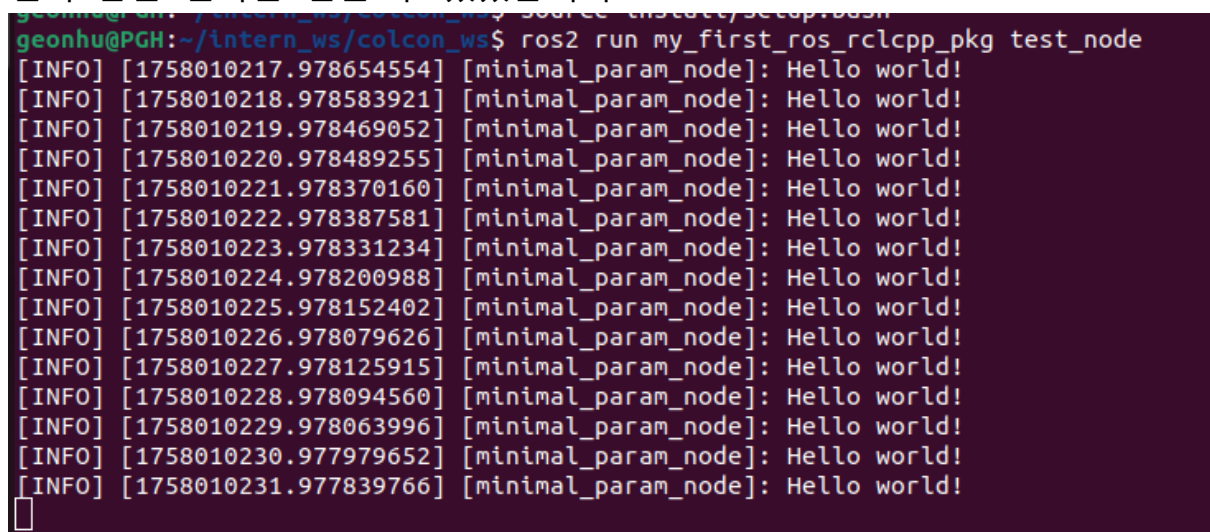
이처럼 Client클래스의 SharedPtr타입을 사용하고, create_client 헬퍼함수를 사용하여 객체를 만들고 포인터를 초기화합니다.

```
auto result = client->async_send_request(request);
```

그리고 async_send_request 메서드를 사용하여 요청을 보내서 설정한 멤버대로 처리되도록 하는 것입니다.

(7) Using parameters in a class 따라하기

이 부분은 과제 2번과도 비슷한 내용입니다. Node 클래스에 있는 declare_parameter와 get_parameter를 사용하여 값을 받아올 수 있으면 get_parameter를 통해 값을 받아오고 값을 받아올 수 없다면 디폴트로 설정된 값을 가져오게 됩니다. 이를 직접 실행해보아 터미널에서 아래의 사진과 같은 결과를 얻을 수 있었습니다.



```
geonhu@PGH:~/intern_ws/colcon_ws$ ros2 run my_first_ros_rclcpp_pkg test_node
[INFO] [1758010217.978654554] [minimal_param_node]: Hello world!
[INFO] [1758010218.978583921] [minimal_param_node]: Hello world!
[INFO] [1758010219.978469052] [minimal_param_node]: Hello world!
[INFO] [1758010220.978489255] [minimal_param_node]: Hello world!
[INFO] [1758010221.978370160] [minimal_param_node]: Hello world!
[INFO] [1758010222.978387581] [minimal_param_node]: Hello world!
[INFO] [1758010223.978331234] [minimal_param_node]: Hello world!
[INFO] [1758010224.978200988] [minimal_param_node]: Hello world!
[INFO] [1758010225.978152402] [minimal_param_node]: Hello world!
[INFO] [1758010226.978079626] [minimal_param_node]: Hello world!
[INFO] [1758010227.978125915] [minimal_param_node]: Hello world!
[INFO] [1758010228.978094560] [minimal_param_node]: Hello world!
[INFO] [1758010229.978063996] [minimal_param_node]: Hello world!
[INFO] [1758010230.977979652] [minimal_param_node]: Hello world!
[INFO] [1758010231.977839766] [minimal_param_node]: Hello world!
```

```
this->declare_parameter("my_parameter", "world");
```

코드에서는 이 부분을 사용하여 my_parameter를 써서 매개변수의 이름을 적어줘서 선언을 한 후 만약 my_parameter라는 이름을 찾을 수 없거나 값이 안 적혀 있다면 world를 디폴트로 가져올 수 있게 설정한 것입니다.

```
std::string my_param = this->get_parameter("my_parameter").as_string();
```

위 사진의 코드는 1초마다 호출되는 메서드의 내용 중 일부로써 `get_parameter`로 `my_parameter`라는 이름에 해당되는 값을 가져오는데 이 때 값이 없다면 `world`를 반환하게 됩니다. 이 때 `get_parameter` 반환값의 타입의 메서드를 통해 실제 스트링을 리턴하도록 하여 `world`문자열을 정상적으로 출력되게 됩니다. 실제 값이 있다면 그 값을 반환하겠지만 관련된 파일이 없는 상태에서 이를 선언하고 값을 얻어올 때는 디폴트 값을 반환함을 볼 수 있었습니다.