

ROS2 day3 hw1 ros2 이해 과제 보고서

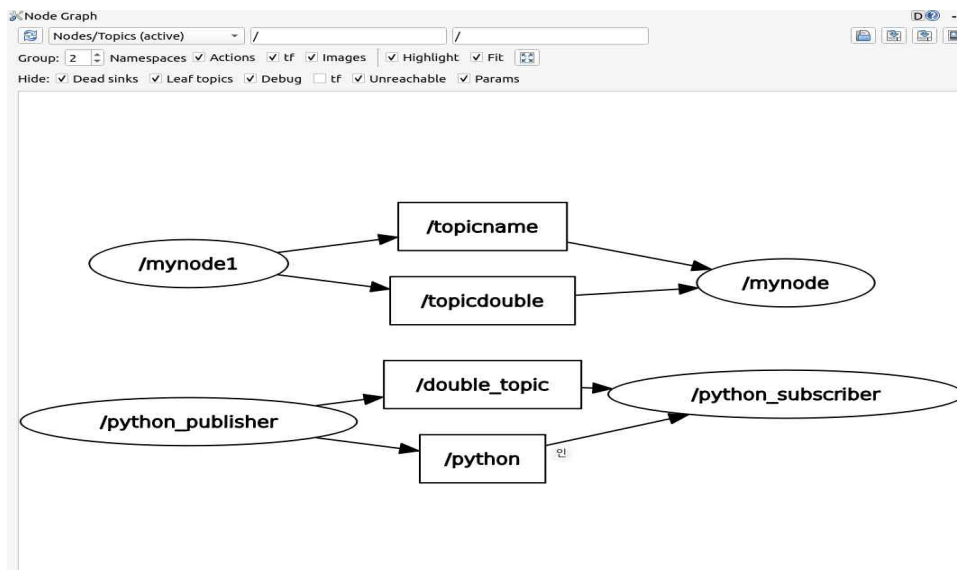
로봇 20기 인턴 현창석

목차

- beginner topic 요약
- beginner service 요약
- beginner parameter 요약
- beginner action 요약
- parameter 사용 예시

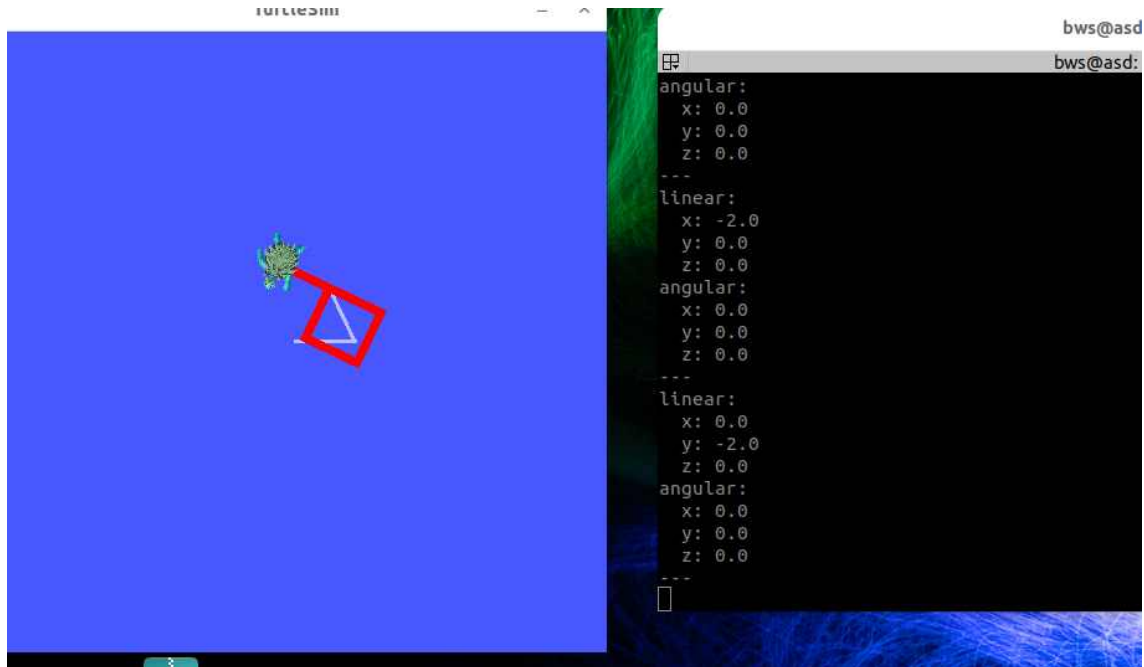
1. - beginner topic 요약

ROS2는 시스템을 노드로 나눈다. 토픽은 이 노드들이 메시지를 교환하는 데에 있어서 가장 중요한 역할을 한다. 노드는 여러 토픽에 데이터를 전송할 수 있고 동시에 여러 주제에 대한 subscribe 할 수 있다.



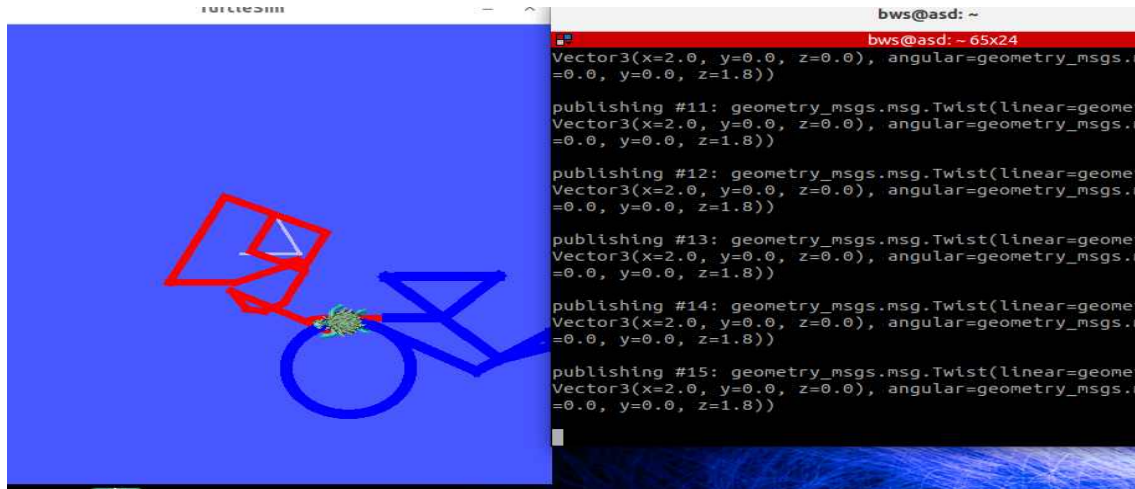
rqt_graph는 노드와 토픽을 포함한 모든 연결을 위 사진과 같이 시각화할 수 있다. 다음은 파이썬과 C++로 구현한 노드 4개를 실행 시킨 후 graph 명령을 실행시켰을 때의 모습이다. 퍼블리시한 데이터가 두 노드를 거쳐 섹스크라이프된 모습이다 .토픽에 퍼블리시되는 데이터를 보기 위해선 `ros2 topic echo <topic_name>` 이 명령어를 입력하면 된다.

`ros2 topic echo /turtle1/cmd_vel` 하여 토픽이름을 cmd_vel로 했을 때 터틀로 도형을 그렸던 과제에서의 노드를 실행시키면



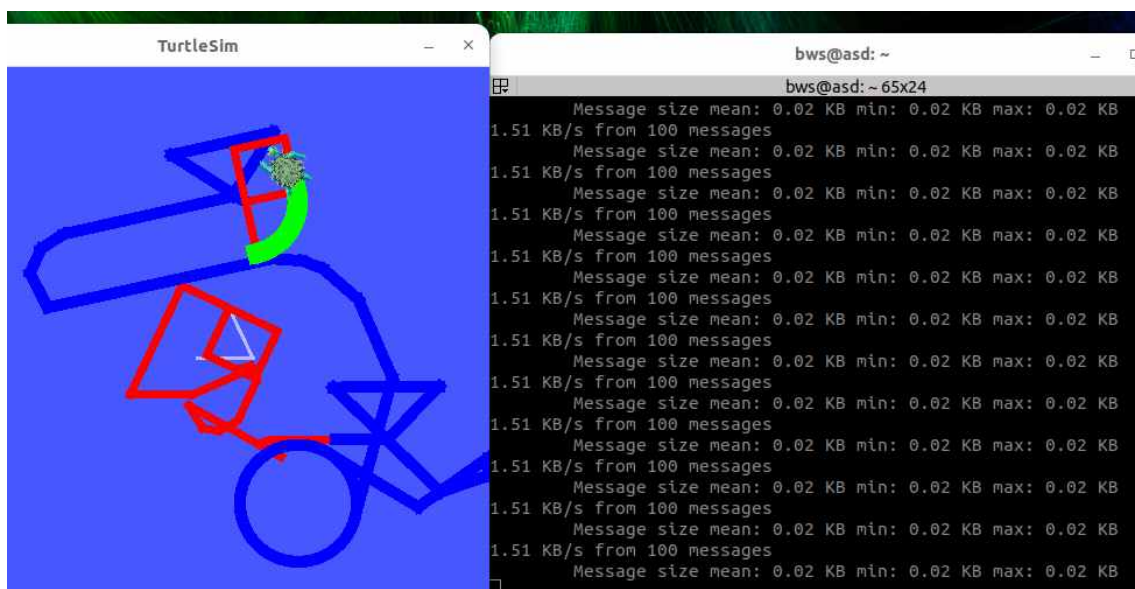
해당 결과물이 출력됨을 알 수 있다. 직접 토픽에 값을 명령하기 위해서는 `ros2 topic pub <topic_name> <msg_type> '<args>'` 해당 명령줄을 이용한다. args에는 토픽에 전달할 실제 데이터 값이다. 터틀을 움직이기 위해서 토픽 이름란에 `/turtle1/cmd_vel geometry_msgs/msg/Twist`를 넣고 `"{linear: {x: 2.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 1.8}}"`를 YAML 구문에 입력을 하면

해당 결과가 출력되는 것을 볼 수 있다. 여기서 YAML이란 포맷에 대한 규칙이다. 시스템과 시스템 사이의 데이터를 주고 받을 경우 데이터의 연동과 호환성을 위해 포맷의 규칙이 요구된다. 기존 웹에서는 XML과 JSON을 이용하였지만 XML은 사용하기 복잡하고 가독성이 떨어지며 json 또한 주석을 달 수 없는 단점과 중괄호와 대괄호로 코드의



길이가 매우 길어지는 단점이 존재하였다. 이러한 단점을 보완하기 위해 사용한 포매팅 방식이 YAMAL이다 . json과 달리 주석도 쓸 수 있으며 주로 Docker Compose, Kubernetes, Flutter, Spring boot 프로젝트에서 설정파일을 정의할 때 주로 사용된다.

토픽에 의해 프로그램이 사용되는 대역폭은 `ros2 topic bw /turtle1/pose` 명령어로 볼 수 있다.



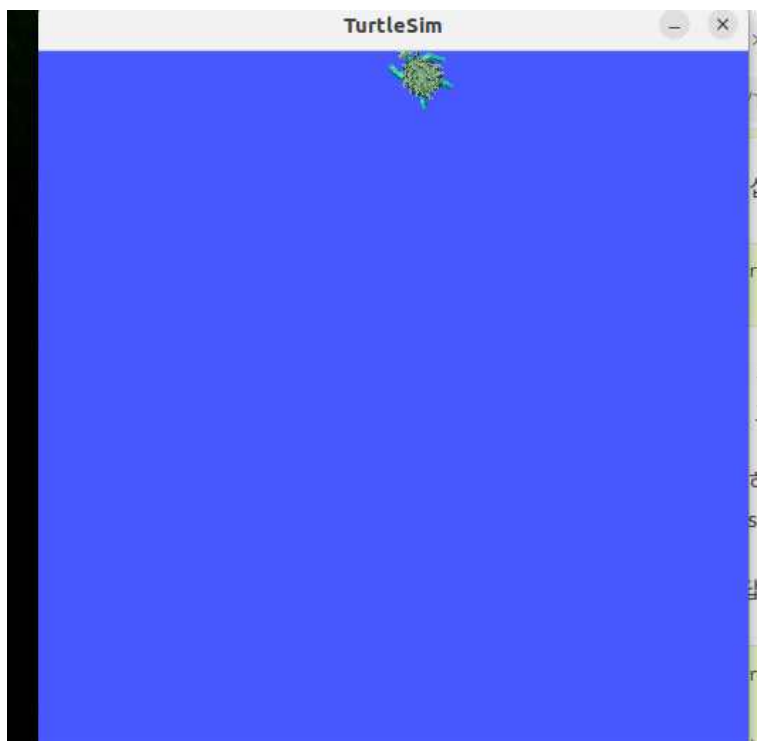
그러면 다음과 같은 결과물을 확인할 수 있다 . 대역폭 과 메시지의 개

수를 위와 같이 확인할 수 있다 .

- beginner service 요약

이번엔 ros2 call 서비스를 사용해볼 것이다.

ros2 service call <service_name> <service_type> <arguments> 형식으로 서비스를 호출할 수 있다. 예를 들어 ros2 service call /clear std_srvs/srv/Empty 이렇게 호출하면

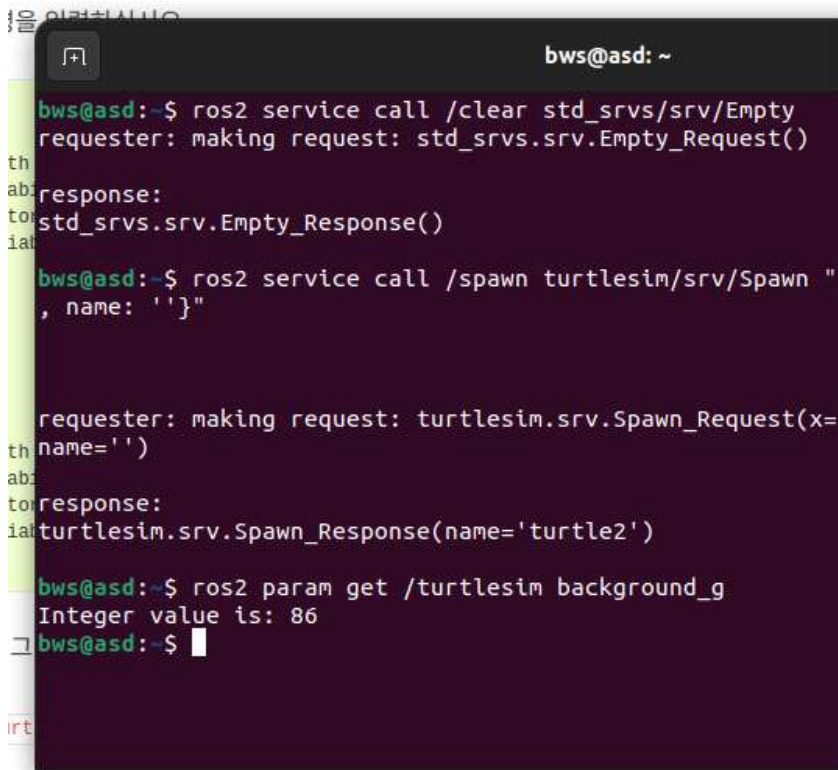


이렇게 아까 터틀이 그린 그림이 모두 지워진 것을 확인할 수 있다.

이렇게 하여 서비스는 단일성 요청에 적합하다는 것을 알 수 있고 지속적으로 통신을 해야한다면 토픽을 사용해야 한다.

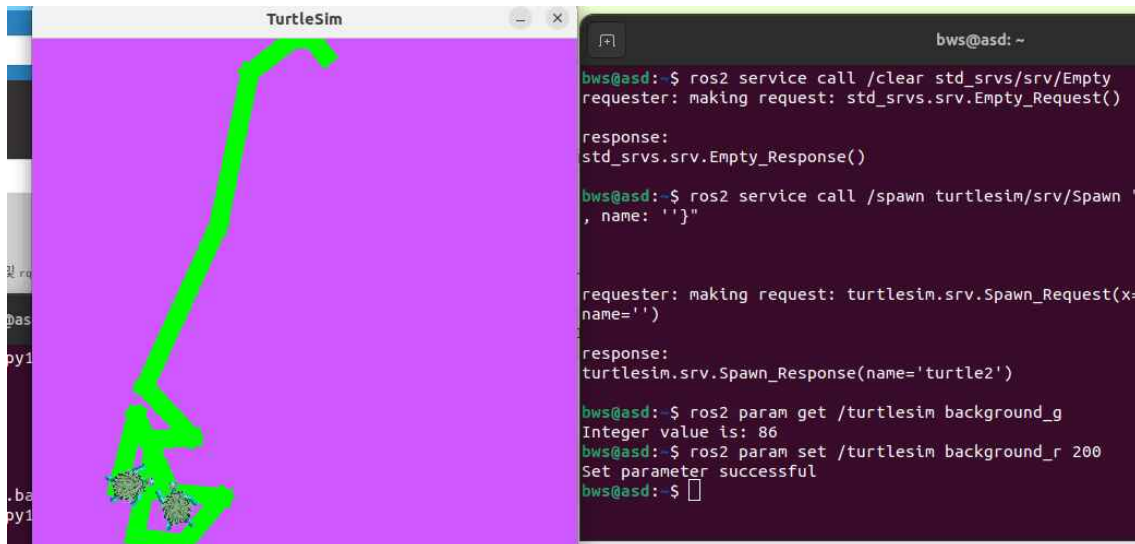
- beginner parameter 요약

노드를 구성하고있는 것은 매개변수들이다. 이 매개변수 값을 변경하여 노드를 설정할 수 있다. `ros2 param get <node_name> <parameter_name>`를 통해 매개 변수의 형식과 현재의 값을 알 수 있다

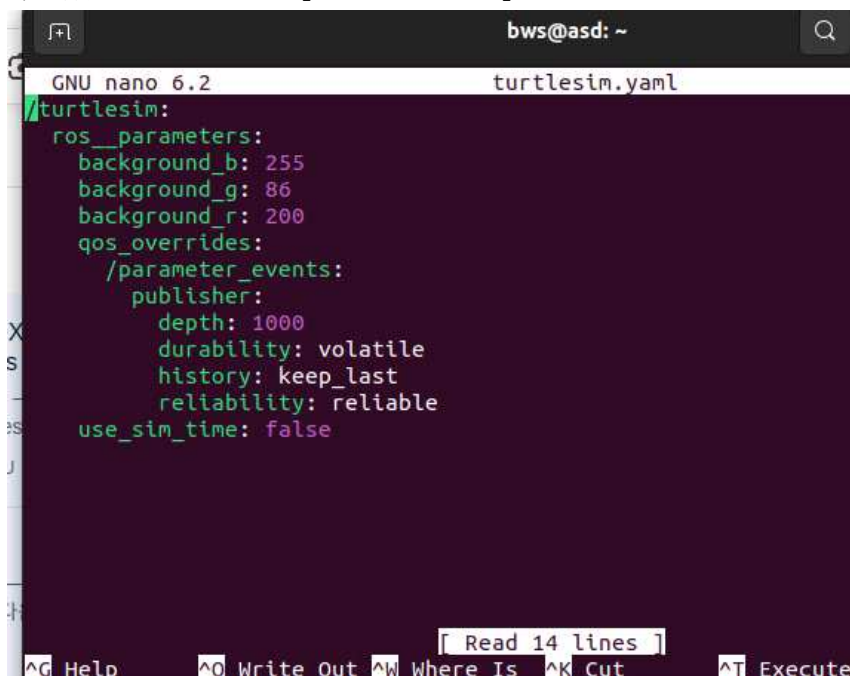


```
bws@asd: ~  
bws@asd:~$ ros2 service call /clear std_srvs/srv/Empty  
requester: making request: std_srvs.srv.Empty_Request()  
response:  
std_srvs.srv.Empty_Response()  
bws@asd:~$ ros2 service call /spawn turtlesim/srv/Spawn "  
, name: ''}"  
requester: making request: turtlesim.srv.Spawn_Request(x=  
name='')  
response:  
turtlesim.srv.Spawn_Response(name='turtle2')  
bws@asd:~$ ros2 param get /turtlesim background_g  
Integer value is: 86  
bws@asd:~$
```

이를 통해 현재 `background_g` 변수 값이 86이라는 것을 알 수 있다. 실행 중에 매개변수의 값을 변경하기 위해선 `ros2 param set <node_name> <parameter_name> <value>` 이 명령을 사용한다.



이렇게 실행 중 명령을 입력하여 배경 색을 변경해보았다. 매개변수 값을 파일로 저장하고 다시 불러올 수도 있다 . 현재 터틀심을 실행할 수 있으므로 `ros2 param dump /turtlesim > turtlesim.yaml` 으로 매



개변수 값을 파일로 저장해보겠다 .

이렇게 저장된 것을 확인할 수 있다 . `ros2 param load <node_name> <parameter_file>` 명령으로 다시 로드 해보면

```
Set parameter background_b successful
Set parameter background_g successful
Set parameter background_r successful
Set parameter qos_overrides./parameter_events.publisher.depth 1
'qos_overrides./parameter_events.publisher.depth' cannot be set
ad-only
Set parameter qos_overrides./parameter_events.publisher.durability 1
eter 'qos_overrides./parameter_events.publisher.durability' can
e it is read-only
Set parameter qos_overrides./parameter_events.publisher.history 1
r 'qos_overrides./parameter_events.publisher.history' cannot be
s read-only
Set parameter qos_overrides./parameter_events.publisher.reliability 1
meter 'qos_overrides./parameter_events.publisher.reliability' c
use it is read-only
Set parameter use_sim_time successful
bws@asd:~$
bws@asd:~$
```

이러한 결과가 출력되는 것을 확인 할 수 있다.

즉 노드에는 기본 구성 값을 정의할 수 있는 매개 변수가 존재한다.
이 매개변수를 설정한 후 yaml 파일에 저장할 수 있고 다시 로드하여
노드에 적용할 수도 있다.

- beginner action 요약

액션은 양방향 메시지 송수신 방식으로 액션 목표 (goal)를 지정하는
Action Client과 액션 목표를 받아 특정 태스크를 수행하면서 중간 결과
값에 해당되는 액션 피드백(feedback)과 최종 결과값에 해당되는 액션
결과(result)를 전송하는 Action Server 간의 통신이라고 볼 수 있다.

토픽과 서비스 방식을 혼합하여 사용하는 방식이라고 할 수 있다.

Action Client는 Service Client 3개와 Topic Subscriber 2개로 구성되어있으며, Action Server는 Service Server 3개와 Topic Publisher 2개로 구성된다. 액션 목표/피드백/결과(goal/feedback/result) 데이터는 msg 및 srv 인터페이스의 변형으로 action 인터페이스라고 한다

토픽에 대해서도 자세히 살펴보면 토픽은 비동기식 단방향 메시지 송수신 방식으로 msg 메시지 형태의 메시지를 발행하는 `Publisher`와 메시지를 구독하는 `Subscriber` 간의 통신이다.

하나의 이상의 토픽을 발행할 수 있을 뿐만이 아니라 `Publisher` 기능과 동시에 토픽(예: Topic D)을 구독하는 `Subscriber` 역할도 동시에 수행할 수 있다. 원한다면 자신이 발행한 토픽을 셀프 구독할 수 있게 구성할 수도 있다.

마지막으로 서비스에 대해서도 자세히 살펴보면 서비스는 양방향 메시지 송수신 방식으로 서비스의 요청(Request)을 하는 쪽을 Service Client라고 하며 요청받은 서비스를 수행한 후 서비스의 응답(Response)을 하는 쪽을 Service Server라고 한다. 복수의 클라이언트를 가질 수 있도록 설계되었다. 단, 서비스 응답은 서비스 요청이 있었던 서비스 클라이언트에 대해서만 응답을 하는 형태이다.

- parameter 사용 예시

기존 퍼블리시 과제의 코드에 파라미터 클래스를 추가로 정의하여 파라미터를 받을 수 있게 하였다.

mynode.hpp 파일에서

```
class MinimalParam : public rclcpp::Node
```

로 ros2가 제공하는 Node를 상속하는 파라미터 클래스를 생성해주었다.

```
MinimalParam();
```

public 부분에는 생성자와

```
~MinimalParam();
```

소멸자를 선언하였다. 그리고

```
void timer_callback();
```

파라미터가 들어오면 실행할 콜백함수를 선언해주었고

마지막으로

```
rclcpp::TimerBase::SharedPtr timer_;
```

지속적으로 로그를 출력해주기 위해 스마트 포인터 타이머를 선언해주었다.

cpp_param.cpp 파일에서는 헤더 파일에 선언한 것들을 구현해주었다.

먼저 MiniMalParam 매개변수의 생성자를 호출하고 ,

```
MinimalParam::MinimalParam() : Node("minimal_param_node")
```

이름을 minimal_param_node 로 정의하였다.

이 생성자 안에는

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

my_parameter의 기본값을 world 문자열로 선언해주고

```
timer_ = this -> create_wall_timer ( 1000ms, std::  
bind(&MinimalParam::timer_callback,this));
```

1000ms의 시간마다 timer_callback함수를 실행하는 타이머를 생성하였다.

```
MinimalParam::timer_callback ()
```

타이머 콜백함수에서는

```
std::string my_param = this ->  
get_parameter("my_parameter").as_string(); my_parameter노드  
에서 파라미터를 저장할 수 있도록 하였고
```

```
RCLCPP_INFO (this -> get_logger(), "Hello %s!",  
my_param.c_str()); my_param에 저장된 문자열 로그를 출력하도록  
하였다.
```

```
또한      std::vector<rcldcpp::Parameter>
all_new_parameters{rcldcpp::Parameter("my_parameter","world")};
```

부분은 "my_parameter"라는 이름을 가진 파라미터를 생성 후 "world"라는 string으로 기본값 설정한다는 의미이다. 여러 개를 한 번에 세팅할 수 있도록 std::vector 형태로 파라미터를 모은다.

마지막으로

```
this->set_parameters(all_new_parameters); 해주어
all_new_parameters 벡터 안에 들어있는 파라미터들을 노드에 한꺼번
에 반영한다.
```

메인 함수

```
int main (int argc, char **argv) {
```

에서는

```
rcldcpp::init(argc,argv);
ros2를 초기화 먼저 해주고
```

```
rcldcpp::spin(std::make_shared<MinimalParam>());
```

MinimalParam클래스 구성하여 rcldcpp::spin노드에서 데이터를 처리하기 시작도록 만든다.