

ROS2 통신 - Service

경희대학교 기계공학과
로봇 제어 지능 연구실
김상현 교수



01

ROS2 Service

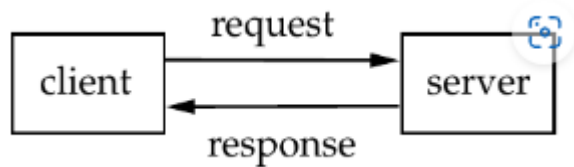
ROS2 Service란?

- ▶ ROS2 Service는 Topic과 다르게 동기식 통신

- Bi-directional (양 노드간 통신), one-to-one (다수의 익명 노드가 아니라 요청/응답하는 2개의 노드가 있음)

Client Node는 일부 데이터를 요청(Request)하여 서버 노드에 보냄.

Server Node는 클라이언트 노드에서 요청받은 작업 (ex. 앞으로 가라, 연산을 해라 등)을 수행하고 이를 다시 클라이언트에 보냄.



ROS2 Topic

- ◉ Service이란 Node 간의 메시지를 주고받는 통신의 한 방법
- ◉ Service을 요청하는 쪽을 Client, 응답하는 쪽을 Server라고 한다.

- ◉ Service의 주요 특징

- ◉ **구성**

- Request와 Response로 구성되어 있다. Client가 노드에 Request를 보내고 Server 노드가 요청을 처리하고 Response를 보냄
 - 토픽과는 다르게 한 번 통신 후 연결이 종료되는 방식

- ◉ **통신 방법**

- 클라이언트는 서버에 정보를 요청하는 메시지를 보내고 서버는 그에 대한 정보를 응답 메시지로 제공한다.
 - 일회성 통신으로 클라이언트의 요청이 완료되면 두 노드는 연결이 끊어진다.

- ◉ **목표**

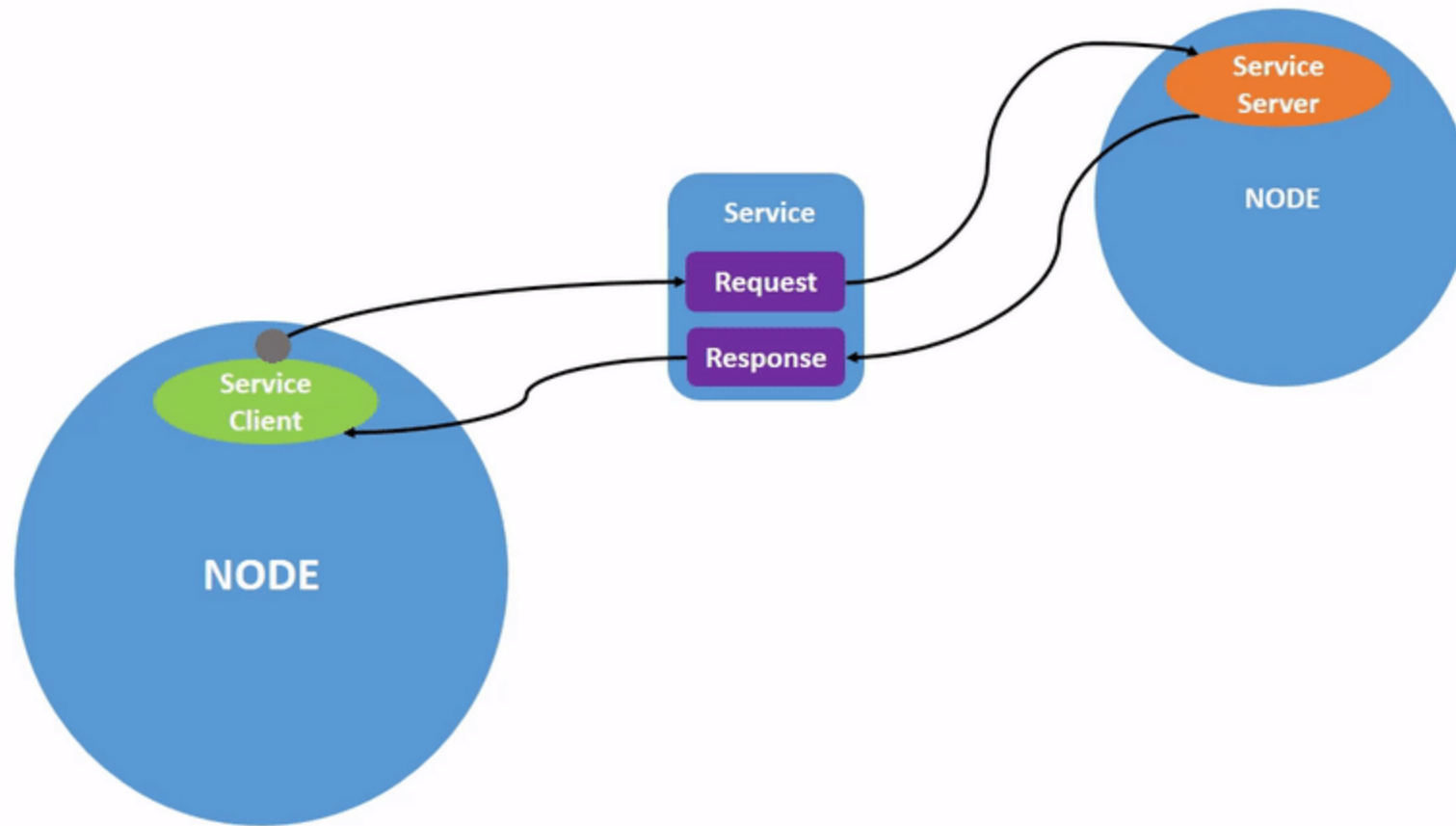
- 특정 조건에서 이벤트를 처리하거나 요청된 명령에 따라 특정 행동을 수행

- ◉ **특징**

- 하나의 서비스에 많은 클라이언트가 있을 수 있지만 서버는 오로지 하나만 존재할 수 있다.

ROS2 Service

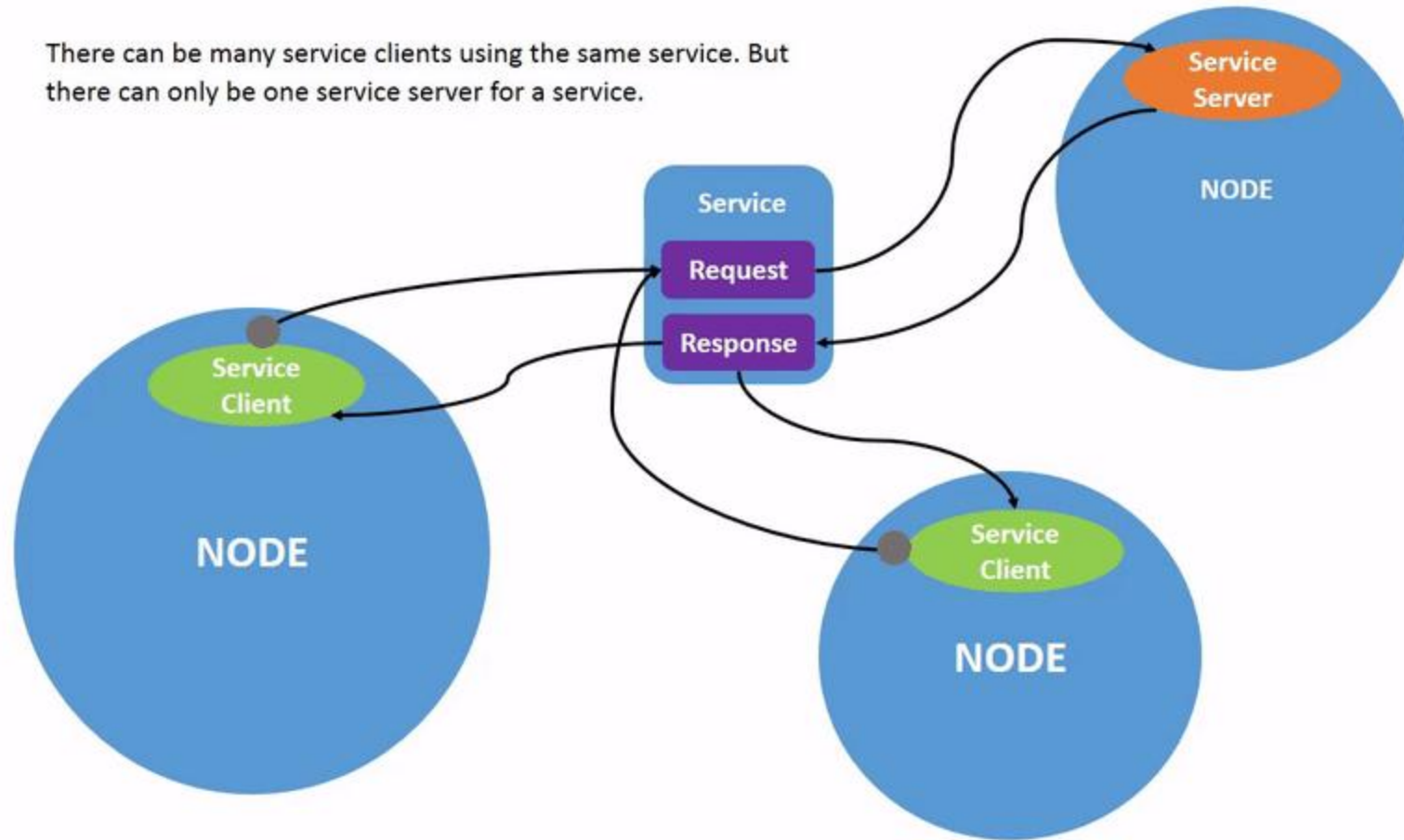
예시 – 1:1 통신



ROS2 Service

예시 – 1:N 통신

There can be many service clients using the same service. But there can only be one service server for a service.



ROS2 Service란?

▶ ROS2 Service 명령어

Node 실행 (Terminal 1)

```
$ ros2 run turtlesim turtlesim_node
```

Node 실행 (Terminal 2)

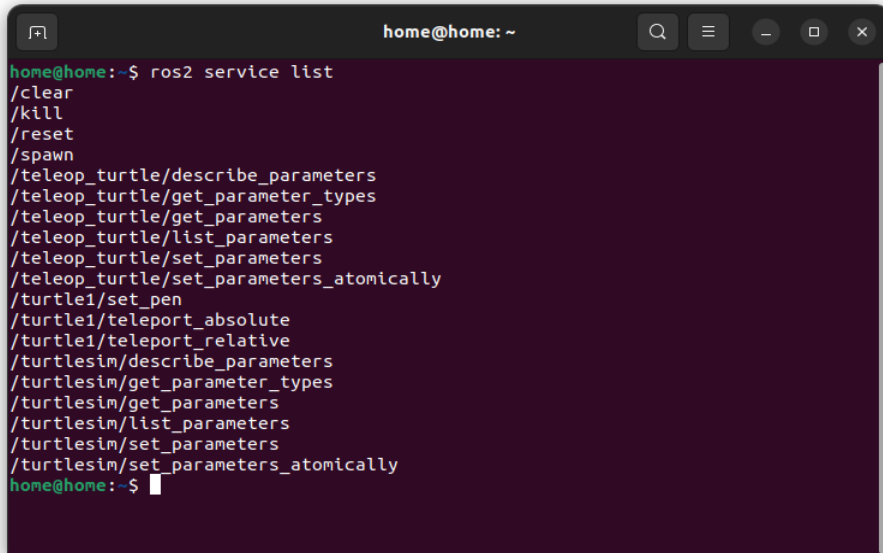
```
$ ros2 run turtlesim turtle_teleop_key
```

Node 실행 (Terminal 3) 사용가능한 Service 목록을 확인

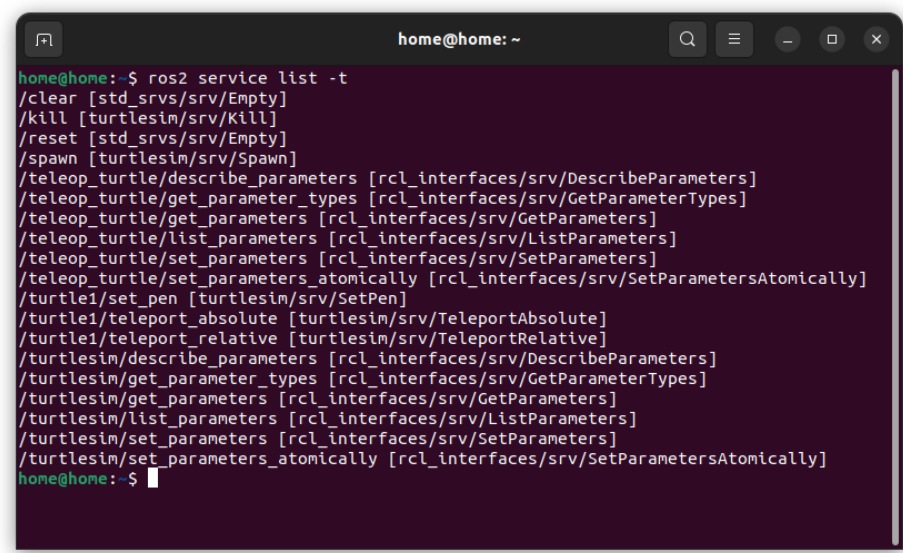
```
$ ros2 service list
```

Node 실행 (Terminal 3) 사용가능한 Service 목록 및 유형 확인.

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



```
home@home: ~  
home@home:~$ ros2 service list  
/clear  
/kill  
/reset  
/spawn  
/teleop_turtle/describe_parameters  
/teleop_turtle/get_parameter_types  
/teleop_turtle/get_parameters  
/teleop_turtle/list_parameters  
/teleop_turtle/set_parameters  
/teleop_turtle/set_parameters_atomically  
/turtle1/set_pen  
/turtle1/teleport_absolute  
/turtle1/teleport_relative  
/turtlesim/describe_parameters  
/turtlesim/get_parameter_types  
/turtlesim/get_parameters  
/turtlesim/list_parameters  
/turtlesim/set_parameters  
/turtlesim/set_parameters_atomically  
home@home:~$
```



```
home@home: ~  
home@home:~$ ros2 service list -t  
/clear [std_srvs/srv/Empty]  
/kill [turtlesim/srv/Kill]  
/reset [std_srvs/srv/Empty]  
/spawn [turtlesim/srv/Spawn]  
/teleop_turtle/describe_parameters [rcl_interfaces/srv/DescribeParameters]  
/teleop_turtle/get_parameter_types [rcl_interfaces/srv/GetParameterTypes]  
/teleop_turtle/get_parameters [rcl_interfaces/srv/GetParameters]  
/teleop_turtle/list_parameters [rcl_interfaces/srv/ListParameters]  
/teleop_turtle/set_parameters [rcl_interfaces/srv/SetParameters]  
/teleop_turtle/set_parameters_atomically [rcl_interfaces/srv/SetParametersAtomically]  
/turtle1/set_pen [turtlesim/srv/SetPen]  
/turtle1/teleport_absolute [turtlesim/srv/TeleportAbsolute]  
/turtle1/teleport_relative [turtlesim/srv/TeleportRelative]  
/turtlesim/describe_parameters [rcl_interfaces/srv/DescribeParameters]  
/turtlesim/get_parameter_types [rcl_interfaces/srv/GetParameterTypes]  
/turtlesim/get_parameters [rcl_interfaces/srv/GetParameters]  
/turtlesim/list_parameters [rcl_interfaces/srv/ListParameters]  
/turtlesim/set_parameters [rcl_interfaces/srv/SetParameters]  
/turtlesim/set_parameters_atomically [rcl_interfaces/srv/SetParametersAtomically]  
home@home:~$
```

ROS2 Service란?

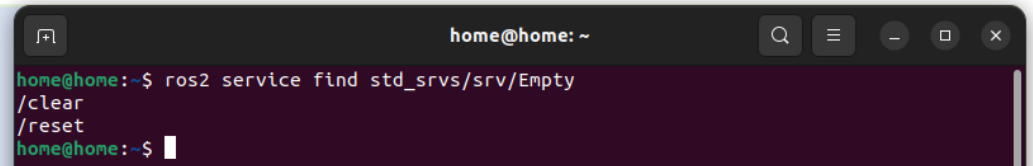
▶ ROS2 Service 명령어

특정 유형의 모든 서비스를 찾는 명령어

\$ ros2 service find <type_name>

Node 실행 (Terminal 3)

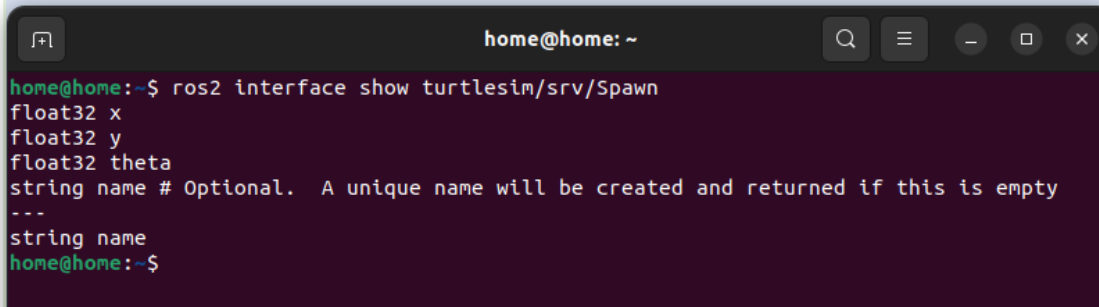
\$ ros2 service find std_srvs/srv/Empty



```
home@home: ~  
home@home:~$ ros2 service find std_srvs/srv/Empty  
/clear  
/reset  
home@home:~$
```

Node 실행 (Terminal 3) Spawn 서비스에 대한 정보 확인

\$ ros2 interface show turtlesim/srv/Spawn



```
home@home:~$ ros2 interface show turtlesim/srv/Spawn  
float32 x  
float32 y  
float32 theta  
string name # Optional. A unique name will be created and returned if this is empty  
---  
string name  
home@home:~$
```

위 - Request

--- 를 기준으로

아래 - Response

ROS2 Service란?

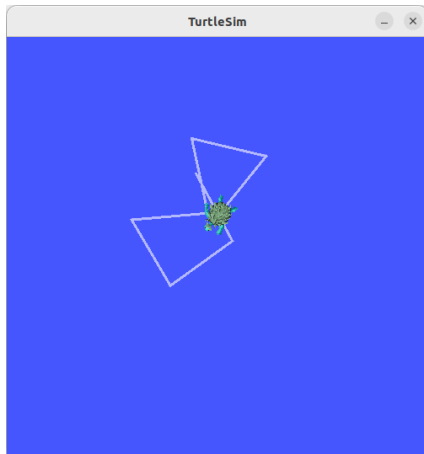
▶ ROS2 Service 명령어

서비스 실행 명령어

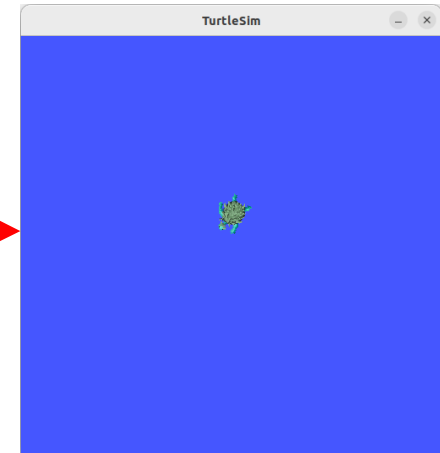
```
$ ros2 service call <Service Name> <Service Type> <Request>
```

Terminal 3

```
$ ros2 service call /clear std_srvs/srv/Empty
```



```
home@home: ~  
home@home:~$ ros2 service call /clear std_srvs/srv/Empty  
waiting for service to become available...  
requester: making request: std_srvs.srv.Empty_Request()  
  
response:  
std_srvs.srv.Empty_Response()  
home@home:~$
```



ROS2 Service

▶ (실습) 커스텀 서비스 만들기

my_srv 패키지 만들기

```
$ ros2 pkg create my_srv --build-type ament_cmake
```

Custom Service 및 Message는 **Cmake**에서만 만들 수 있다

```
# ros2_ws/src폴더 안에 만들 것
```

Package.xml 수정 (2차시 예제 참조)

```
<buildtool_depend>rosidl_default_generators</buildtool_depend>
<exec_depend>rosidl_default_runtime</exec_depend>
<member_of_group>rosidl_interface_packages</member_of_group>
```

CMakeLists.txt 수정 (2차시 예제 참조)

```
find_package(rosidl_default_generators REQUIRED)
```

```
rosidl_generate_interfaces(${PROJECT_NAME}
  "srv/Changerate.srv"
)
```

Changerate.srv 만들기

```
$ mkdir srv
```

```
▼ my_srv
  ▼ srv
    ≡ Changerate.srv
```

```
float64 newrate
---
bool ret
```

} Request

} Response

ROS2 Service (Python)

- ▶ (실습) 터틀봇을 특정 위치에 spawn하는 노드를 작성

service_tutorial_py 패키지 만들기

```
$ ros2 pkg create service_tutorial_py --build-type ament_python --dependencies rclpy turtlesim std_srvs geometry_msgs my_srv  
# ros2_ws/src폴더 안에 만들 것
```

setup.py 수정 (2차시 예제 참조)

```
entry_points={  
    'console_scripts': [  
        'spawn_turtle = service_tutorial_py.turtle_spawn:main',  
        'velocity_toggle_service = service_tutorial_py.velocity_toggle:main',  
    ],  
}
```

ROS2 Service (Python)

▶ 클라이언트 코드 설명 (Spawn)

- 서비스 호출

리퀘스트/리스폰스 타입: 각 서비스들은 정해진 서비스 규격을 따른다.

```
1) from turtlesim.srv import Spawn
```

클라이언트 오브젝트: 각 서비스들은 지정된 서비스 오브젝트를 통해 “호출(call)” 된다.

```
1) self.client_ = self.create_client( <Service_type>,<Service_name>)
```

- service_type: part inside the angle brackets — formally called the template parameter — is the data type for the service
(본 예제에서는 turtlesim service의 spawn 타입)
- service_name: a string containing the name of the service on which we want to publish
(본 예제에서는 spawn)

서비스 오브젝트

```
self.request = Spawn.Request()
```

```
self.request.x
```

```
self.request.y
```

```
self.request.name
```

} 객체 안의 변수에 값 대입

```
future = self.client.call_async(self.request)
```

```
rclpy.spin_until_future_complete(self,future)
```

} Service 객체 생성

} Service 요청 (request 객체 전달)
Service의 결과 기다리기

ROS2 Service (Python)

▶ 서버 코드 설명 (toggle)

– 서비스 서버

서비스 콜백: 토픽의 subscriber와 마찬가지로 콜백함수가 존재한다.

```
def function_name(self, request, response):  
    ...
```

서버 오브젝트: 각 서비스들은 지정된 서비스 오브젝트를 통해 콜백함수와 결합된다.

```
self.service = self.create_service(<service_type>,<service_name>,<callback_function>)
```

표준 서비스: https://index.ros.org/p/std_srvs/#humble

std_srvs

This package provides several service definitions for standard but simple ROS services.

For more information about ROS 2 interfaces, see [index.ros2.org](https://index.ros.org/p/index.ros2.org)

Services (.srv)

- [Empty.srv](#): A service containing an empty request and response.
- [SetBool.srv](#): Service to set a boolean state to true or false, for enabling or disabling hardware for example.
- [Trigger.srv](#): Service with an empty request header used for triggering the activation or start of a service.

ROS2 Service (Python)

▶ (실습) 터틀봇을 특정 위치에 spawn하는 노드를 작성

turtle_spawn.py

```
import rclpy
from rclpy.node import Node
from turtlesim.srv import Spawn

class SpawnTurtleNode(Node):

    def __init__(self):
        super().__init__('spawn_turtle_client')
        self.client_ = self.create_client(Spawn, 'spawn')
        while not self.client_.wait_for_service(timeout_sec=1.0):
            self.get_logger().info('Service not available, waiting again...')

        self.request_ = Spawn.Request()

    def send_request(self):
        self.request_.x = 5.0
        self.request_.y = 2.0
        self.request_.name = 'new_turtle'

        future = self.client_.call_async(self.request_)
        rclpy.spin_until_future_complete(self, future)

        if future.result() is not None:
            self.get_logger().info(f"Successfully spawned a new turtle: {future.result().name}")
        else:
            self.get_logger().error('Failed to spawn a new turtle')

def main(args=None):
    rclpy.init(args=args)
    node = SpawnTurtleNode()
    node.send_request()
    rclpy.shutdown()

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

Service_client node 및 client object 생성

Service에 Parameter 전달
새로운 turtle을 x=5, y=2에 생성

ROS2 Service (Python)

▷ (실습) 터틀봇의 속도를 toggle하는 노드를 작성

velocity_toggle.py

```
from std_srvs.srv import Empty
from geometry_msgs.msg import Twist

class PubVelToggle(Node):

    def __init__(self):
        super().__init__('pubvel_toggle')
        self.forward = True
        self.cnt = 0

        # 서비스 생성
        self.server_ = self.create_service(Empty, 'toggle_forward', self.toggle_forward_callback)

        # 퍼블리셔 생성
        self.pub_ = self.create_publisher(Twist, 'turtle1/cmd_vel', 10)

        # 타이머 생성 (500ms 주기)
        self.timer_ = self.create_timer(0.5, self.publish_velocity)

        self.get_logger().info("PubVelToggle node has been started.")

    def toggle_forward_callback(self, request, response):
        self.cnt += 1
        self.forward = not self.forward
        self.get_logger().info(f"Now sending {'forward' if self.forward else 'rotate'} commands.")
        return response

    def publish_velocity(self):
        if self.cnt == 0:
            return

        msg = Twist()
        msg.linear.x = 1.0 if self.forward else 0.0
        msg.angular.z = 0.0 if self.forward else 1.0

        self.pub_.publish(msg)

        self.get_logger().info(f"Published velocity command - linear.x: {msg.linear.x}, angular.z: {msg.angular.z}")

def main(args=None):
    rclpy.init(args=args)

    node = PubVelToggle()
    rclpy.spin(node)

    rclpy.shutdown()
```

Service Server 생성

Service Call시 실행되는 함수

ROS2 Service (Python)

- (실습) 터틀봇을 특정 위치에 spawn하는 노드 및 속도를 toggle하는 노드를 작성

Terminal 1

```
$ ros2 run turtlesim turtlesim_node
```

Spawn Service Node 실행 (Terminal 2)

```
$ ros2 run service_tutorial_py spawn_turtle
```

-ros2 service call /spawn turtlesim/srv/Spawn "{x:2.0, y:2.0, theta: 0.0, name: 'myturtle'}" 와 결과 비교

Velocity Toggle Node 실행 (Terminal 2)

```
$ ros2 run service_tutorial_py velocity_toggle_service
```

Toggle Service Call(Terminal 3)

```
$ ros2 service call /toggle_forward std_srvs/srv/Empty
```

} Call 할 때마다 toggle적용

ROS2 Service (C++)

클라이언트 코드 설명 (Spawn)

- 서비스 호출

리퀘스트/리스폰스 타입: 각 서비스들은 정해진 서비스 규격을 따른다.

1) `#include "turtlesim/srv/spawn.hpp"`

클라이언트 오브젝트: 각 서비스들은 지정된 서비스 오브젝트를 통해 "호출(call)" 된다.

1) `node->create_client<ServiceType>("Servicename")`

- service_type: part inside the angle brackets — formally called the template parameter — is the data type for the service
(본 예제에서는 turtlesim::srv::Spawn)
- service_name: a string containing the name of the service on which we want to publish
(본 예제에서는 spawn)

서비스 오브젝트

```
auto request = std::make_shared<turtlesim::srv::Spawn::Request>() } Service 객체 생성
request->x
request->y } 객체 안의 변수에 값 대입
request->name

auto result = client->async_send_request(request) } Service 요청 (request 객체 전달)
```

ROS2 Service (C++)

▶ 서버 코드 설명 (toggle)

– 서비스 서버

서비스 콜백: 토픽의 subscrib와 마찬가지로 콜백함수가 존재한다.

```
void function_name(const std::shared_ptr<package_name::service_type::Request> request,
                  std::shared_ptr<package_name::service_type::Response> response)
{
    ...
}
```

서버 오브젝트: 각 서비스들은 지정된 서비스 오브젝트를 통해 콜백함수와 결합된다.

- 1) `rclcpp::Service<package_name::srv::Service>::SharedPtr service = node->create_service<package_name::srv::Service>("servicename", &callback)`

표준 서비스: https://index.ros.org/p/std_srvs/#humble

std_srvs

This package provides several service definitions for standard but simple ROS services.

For more information about ROS 2 interfaces, see [index.ros2.org](https://index.ros.org)

Services (.srv)

- [Empty.srv](#): A service containing an empty request and response.
- [SetBool.srv](#): Service to set a boolean state to true or false, for enabling or disabling hardware for example.
- [Trigger.srv](#): Service with an empty request header used for triggering the activation or start of a service.

ROS2 Service (C++)

▷ (실습) 터틀봇을 특정 위치에 spawn하는 노드를 작성

turtle_spawn.cpp

```
#include "rclcpp/rclcpp.hpp"
#include "turtlesim/srv/spawn.hpp"

#include <chrono>
#include <cstdlib>
#include <memory>

class SpawnTurtleNode: public rclcpp::Node
{
public:
    SpawnTurtleNode() : Node("spawn_turtle_client")
    {
        client_ = this -> create_client<turtlesim::srv::Spawn>("spawn");
        request_ = std::make_shared<turtlesim::srv::Spawn::Request>();
    }

    void send_request()
    {
        request_ -> x = 5.0;
        request_ -> y = 2.0;
        request_ -> name = "new_turtle";

        while(!client_ -> wait_for_service(std::chrono::seconds(1)))
        {
            if(!rclcpp::ok())
            {
                RCLCPP_INFO(this -> get_logger(), "Interrupted while waiting for the service. Exiting ...");
                return;
            }
            RCLCPP_INFO(this -> get_logger(), "Service not available, waiting again ...");
        }

        auto result = client_ -> async_send_request(request_);

        if (rclcpp::spin_until_future_complete(this -> get_node_base_interface(), result) ==
            rclcpp::FutureReturnCode::SUCCESS)
        {
            auto response = result.get();
            RCLCPP_INFO(this -> get_logger(), "Successfully spawned a new turtle: %s", response -> name.c_str());
        }
        else
        {
            RCLCPP_ERROR(this -> get_logger(), "Failed to spawn a new turtle");
        }
    }

private:
    rclcpp::Client<turtlesim::srv::Spawn>::SharedPtr client_;
    turtlesim::srv::Spawn::Request::SharedPtr request_;
};

int main(int argc, char * argv[])
{
    rclcpp::init(argc, argv);
    auto node = std::make_shared<SpawnTurtleNode>();
    node -> send_request();
    rclcpp::shutdown();
}
```

Service_client node 및 client object 생성

Service에 Parameter 전달

ROS2 Service (C++)

◉ (실습) 터틀봇의 속도를 toggle하는 노드를 작성

velocity_toggle.cpp

```
#include "rclcpp/rclcpp.hpp"
#include "std_srvs/srv/empty.hpp"
#include "geometry_msgs/msg/twist.hpp"

class PubVelToggle : public rclcpp::Node
{
public:
    PubVelToggle() : Node("pubvel_toggle"), forward_(true), cnt_(0)
    {
        server_ = this->create_service<std_srvs::srv::Empty>(
            "toggle_forward",
            std::bind(&PubVelToggle::toggle_forward_callback, this, std::placeholders::_1, std::placeholders::_2)
        );

        pub_ = this->create_publisher<geometry_msgs::msg::Twist>("turtle1/cmd_vel", 10);

        timer_ = this->create_wall_timer(
            std::chrono::milliseconds(500),
            std::bind(&PubVelToggle::publish_velocity, this)
        );

        RCLCPP_INFO(this->get_logger(), "PubVelToggle node has been started.");
    }

private:
    rclcpp::Service<std_srvs::srv::Empty>::SharedPtr server_;
    rclcpp::Publisher<geometry_msgs::msg::Twist>::SharedPtr pub_;
    rclcpp::TimerBase::SharedPtr timer_;
    bool forward_;
    int cnt_;

    bool toggle_forward_callback(
        const std::shared_ptr<std_srvs::srv::Empty::Request> request,
        std::shared_ptr<std_srvs::srv::Empty::Response> response)
    {
        cnt_++;
        forward_ = !forward_;
        RCLCPP_INFO(this->get_logger(), "Now sending %s commands.", forward_ ? "forward" : "rotate");
        return true;
    }

    void publish_velocity()
    {
        if (cnt_ == 0) {
            return;
        }

        auto msg = std::make_shared<geometry_msgs::msg::Twist>();

        msg->linear.x = forward_ ? 1.0 : 0.0;
        msg->angular.z = forward_ ? 0.0 : 1.0;

        pub_->publish(*msg);

        RCLCPP_INFO(this->get_logger(), "Published velocity command - linear.x: %.2f, angular.z: %.2f", msg->linear.x, msg->angular.z);
    }
};

int main(int argc, char **argv)
{
    rclcpp::init(argc, argv);

    auto node = std::make_shared<PubVelToggle>();
    rclcpp::spin(node);

    rclcpp::shutdown();
    return 0;
}
```

Service Server 생성

Service Call시 실행되는 함수

ROS2 Service

- 실습 2: change_rate.srv 를 만들어 float64를 이용하여, 제어 주기를 바꿔주는 srv를 만들기

명령어

```
ros2 run turtlesim turtlesim_node
```

```
ros2 run service_tutorial_py velocity_toggle_plus_service  
-ros2 service call /change_rate my_srv/srv/Changerate newrate:\ value\
```

Callback

```
def change_rate_callback(self, request, response):  
    self.get_logger().info(f"Changing rate to {request.newrate}")  
    self.frequency = request.newrate  
    self.rate_changed = True  
    return response
```

Service

```
self.change_rate_service = self.create_service(Changerate, 'change_rate', self.change_rate_callback)
```

감사합니다

