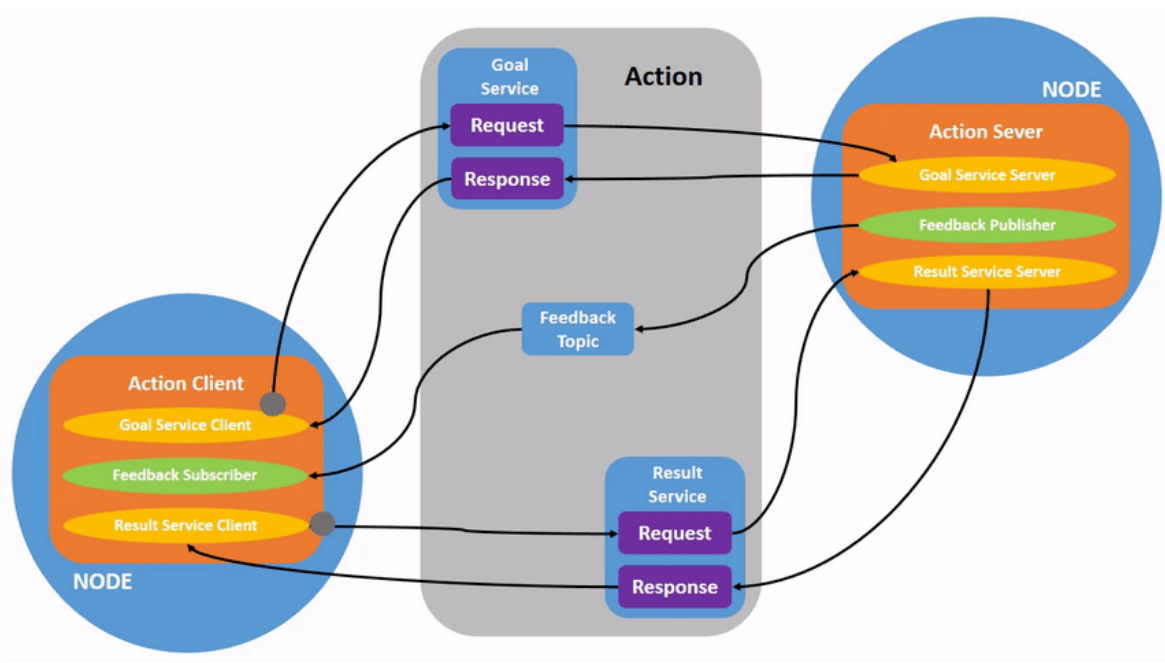
week4

▼ Week4

13. ROS2 액션

액션은 비동기식, 동기식 양방향 메시지 송수신 방식으로 액션 목표(Goal)을 지정하는 Action Client와 목표를 받아 특정 태스크를 수행하면서 중간 결과값을 전송하는 액션 피드백(Feedback) 그리고 최종 결괏값을 담은 액션 결과(Result)를 전송하는 Action Server간의 통신이다



액션은 토픽과 서비스의 결합이라고 볼 수 있는데 ROS 1의 액션이 토픽으로만 이루어진 것과 달리, ROS 2의 Action Client는 Service Client 3개와 Topic Subscriber 2개로 구성되어 있으며,

Action Server는 Service Server 3개와 Topic Publisher 2개로 구성됩니다. 이렇게 복잡하게 구성된 이유는, 이후 설명할 cancel_goal 등의 기능을 추가하기 위해 구성이 달라진 것이다.

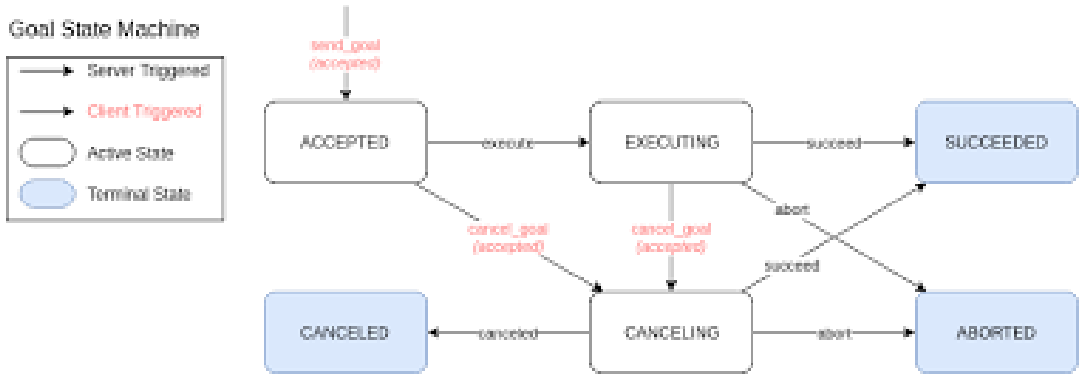
액션 목표, 피드백, 결과 데이터는 msg, serv 인터페이스의 변형인 **action 인터페이스**를 사용한다.

ROS 1에서 토픽만으로 액션을 구성하였을 때에는 동기식 방식을 사용하게 되지만, ROS 2 액션은

토픽+서비스로 이루어져 있기 때문에, 서버에 목표 전달(send_goal), 목표 취소(cancel_goal),

결과 받기(get_result)를 요청하여 새로운 기능을 이용할 수 있다.

비동기 방식의 약점은 실행 시점을 특정하기 어렵다는 것인데 이를 보완하기 위해 구현된 것이 ROS 2의 목표 상태(goal_state)입니다. 목표 상태는 목표 값의 상태 머신을 구동하여 액션의 프로세스를 쫓습니다. Goal State Machine은 액션 목표 전달 이후 액션의 상태 값을 액션 클라이언트에게 전달할 수 있어 비동기/동기 액션의 처리를 원활하게 할 수 있습니다.



노드 정보(ros2 node info /노드이름)

```
$ ros2 node info /turtlesim
/turtlesim
(생략)
  Action Servers:
    /turtle1/rotate_absolute: turtlesim/action/RotateAbsolute
  Action Clients:
```

- /turtlesim 노드는 turtle1/rotate_absolute 액션의 Action Server 역할
- turtlesim/action/RotateAbsoulute가 해당 액션의 인터페이스로 사용됨

```
$ ros2 node info /teleop_turtle
/teleop_turtle
(생략)
  Action Servers:

  Action Clients:
    /turtle1/rotate_absolute: turtlesim/action/RotateAbsolute
```

- /teleop_turtle 노드는 /turtle1/rotate_absolute을 요청하는 Action Client 역할

액션 목록(ros2 action list -t)

```
~$ ros2 action list -t
/turtle1/rotate_absolute [turtlesim/action/RotateAbsolute]
```

- 현재 개발환경에서 실행 중인 액션 목록 및 [액션 타입] 확인

액션 정보(ros2 action info /액션 이름)

```
~$ ros2 action info /turtle1/rotate_absolute
Action: /turtle1/rotate_absolute
Action clients: 1
    /teleop_turtle
Action servers: 1
    /turtlesim
```

- 사용되는 액션 이름과 해당 액션의 Action Server , Client 및 노드 이름, 갯수 확인

액션 목표(action goal) 전달

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

- 위의 형태 명령어로 액션 목표 전달 가능

```
~$ ros2 action send_goal /turtle1/rotate_absolute turtlesim/action/RotateAbsolute "{theta: 1.5708}"
Waiting for an action server to become available...
Sending goal:
     theta: 1.5708

Goal accepted with ID: c2e302b281bd41acb88af19089555def

Result:
    delta: 3.1200063228607178

Goal finished with status: SUCCEEDED
```

- 거북이가 12시 방향 향한다. 목표 값, UID, 변위 값(delta) 표시

```
$ ros2 action send_goal /turtle1/rotate_absolute turtlesim/action/RotateAbsolute "{theta: -1.5708}" --feedback
Waiting for an action server to become available...
Sending goal:
     theta: -1.5708

Goal accepted with ID: ad7dc695c07c499782d3ad024fa0f3d2

Feedback:
    remaining: -3.127622127532959

Feedback:
    remaining: -3.111621856689453

Feedback:
    remaining: -3.0956220626831055

(생략)

Feedback:
    remaining: -0.03962135314941406

Feedback:
    remaining: -0.023621320724487305

Feedback:
```

```
    remaining: -0.007621288299560547

Result:
    delta: 3.1040005683898926

Goal finished with status: SUCCEEDED
```

- 움직이는 중간과정들을 계속 피드백 받기위해선 끝에 —feedback 옵션을 덧붙인다.

14. ROS2 인터페이스

ROS 노드 간 데이터 주고 받기 위해서는 토픽, 서비스, 액션을 사용하는데 이 때 사용되는 데이터 형태를 ROS 2 인터페이스라고 한다.

토픽 - msg

서비스 - srv

액션 - action interface

IDL (Interface Definition Language)

1. 토픽 - 메시지 인터페이스(Message interface, msg)

turtlesim 패키지를 예로 들자면

```
~$ ros2 topic list -t
/parameter_events [rcl_interfaces/msg/ParameterEvent]
/rosout [rcl_interfaces/msg/Log]
/turtle1/cmd_vel [geometry_msgs/msg/Twist]
/turtle1/color_sensor [turtlesim/msg/Color]
/turtle1/pose [turtlesim/msg/Pose]/
```

- /turtle1/cmd_vel 토픽은 geometry_msgs/msgs/Twist 형태로, 기하학 관련 메시지를 모아둔 geometry_msgs 패키지의 msgs 분류의 Twist 데이터 형태다.

```
~$ ros2 interface show geometry_msgs/msg/Twist
# This expresses velocity in free space broken into its linear and angular parts.

Vector3  linear
Vector3  angular

~$ ros2 interface show geometry_msgs/msg/Vector3
# This represents a vector in free space.

# This is semantically different than a point.
# A vector is always anchored at the origin.
# When a transform is applied to a vector, only the rotational component is applied.

float64 x
float64 y
float64 z
```

- Twist 데이터형태는 Vector3 형태의 linear, angular 메시지가 있고 Vector3은 float64 형태로 x, y, z값을 가진다.

ros2 interface에는 show 외에도 list, package, packages, proto가 있다.

```
$ ros2 interface list
Messages:
    action_msgs/msg/GoalInfo
    action_msgs/msg/GoalStatus
    action_msgs/msg/GoalStatusArray
    actionlib_msgs/msg/GoalID
    actionlib_msgs/msg/GoalStatus
(생략)
Services:
    action_msgs/srv/CancelGoal
    composition_interfaces/srv/ListNodes
    composition_interfaces/srv/LoadNode
    composition_interfaces/srv/UnloadNode
    diagnostic_msgs/srv/AddDiagnostics
    diagnostic_msgs/srv/SelfTest
    example_interfaces/srv/AddTwoInts
    example_interfaces/srv/SetBool
(생략)
Actions:
    action_tutorials_interfaces/action/Fibonacci
    example_interfaces/action/Fibonacci
    tf2_msgs/action/LookupTransform
    turtlesim/action/RotateAbsolute
```

list는 현재 개발환경의 모든 msg, srv, action 메시지를 보여준다

```
$ ros2 interface packages
action_msgs
action_tutorials_interfaces
actionlib_msgs
builtin_interfaces
composition_interfaces
diagnostic_msgs
example_interfaces
geometry_msgs
(생략)
```

packages는 msg, srv, action 인터페이스를 담고 잇는 패키지 목록을 보여준다.

```
~$ ros2 interface package turtlesim
turtlesim/srv/TeleportRelative
turtlesim/srv/Spawn
turtlesim/msg/Color
turtlesim/srv/TeleportAbsolute
turtlesim/msg/Pose
turtlesim/action/RotateAbsolute
turtlesim/srv/Kill
turtlesim/srv/SetPen

~$ ros2 interface proto geometry_msgs/msg/Twist
"linear:
  x: 0.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
"
```

package 옵션에 패키지명을 입력하면 지정한 패키지에 포함된 인터페이스들을 보여준다.

proto에 특정 인터페이스 형태를 입력하면 그 인터페이스의 기본 형태를 표시한다.

2. 서비스 인터페이스(Service interface, srv)

/spawn 서비스를 예로 들자.

서비스 인터페이스는 메시지 인터페이스와 달리 서비스 요청 및 응답(Request/Response)형태로 구분됨.

요청 / 응답 데이터를 나누어 사용하기 위해 구분자(- - -)를 사용함

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

- x, y, theta, name 데이터는 서비스 요청에 해당되고 서비스 클라이언트 단에서 서비스 서버 단에 전송하는 데이터가 된다.
- 구분자(- - -)뒤의 name은 서비스 응답에 해당되고 서비스 서버 단에서 클라이언트의 요청에 응답하는 데이터가 된다.

3. 액션 인터페이스(Action interface, action)

메시지 및 서비스 인터페이스(msg, srv)의 확장형으로 /turtle1/rotate_absolute 서비스를 예로 들자

```
:$ ros2 interface show turtlesim/action/RotateAbsolute.action
# The desired heading in radians
float32 theta # 액션목표
---
# The angular displacement in radians to the starting position
float32 delta # 액션 결과
---
# The remaining rotation in radians
float32 remaining # 액션 피드백
```

- /turtle1/rotate_absolute 액션에 사용된 RotateAbsolute.action 인터페이스를 알아보자
- theta, delta, remaining의 3개 데이터를 포함하고 각각 구분자(- - -)로 액션 목표, 결과, 피드백에 해당된다.

15. ROS2 토픽/서비스/액션 정리 및 비교

	토픽	서비스	액션
연속성	연속성	일회성	토픽+서비스
방향성	단방향	양방향	양방향
동기성	비동기	동기	동기+비동기
다자간 연결	1:1, 1:N, N:1, N:N (publisher:subscriber)	1:1 (Server : Client)	1:1 (Server : Client)
노드 역할	Publisher, Subscriber	서버, 클라이언트	서버, 클라이언트
동작 트리거	Publisher	클라이언트	클라이언트
인터페이스	msg 인터페이스	srv 인터페이스	action 인터페이스
CLI 명령어	ros2 topic ros2 interface	ros2 service, ros2 interface	ros2 action, ros2 interface
사용 예	센서 데이터, 로봇 상태, 로봇 좌표, 로봇 속도 명령 등	LED 제어, 모터 토크 On/Off, IK/FK 계산, 이동 경로 계산 등	목적지로 이동, 물건 파지, 복합 태스크 등

인터페이스 비교

	msg	srv	action
데이터	토픽 데이터(data)	서비스 요청(request) ——— 서비스 응답(response)	액션 목표(goal) ———— 액션 결과(result) ————- 액션 피드백(feedback)
사용 예	geometry_msgs/msg/Twist	turtlesim/srv/Spawn.srv	turtlesim/action/RotateAbsolute.action

16. 파라미터

파라미터는 서비스 통신 방법을 사용해 노드 내부 또는 외부에서 노드 내 매개변수를 쉽게 지정(set)하거나 변경할 수 있고, 쉽게 가져(Get)와서 사용할 수 있게 하는 것이 목적이다.

모든 노드는 자신만의 '파라미터 서버'가 있고, '파라미터 클라이언트'를 가질 수 있어 자기 자신의 파라미터 및 다른 노드의 파라미터를 읽고 쓸 수 있다.

—> 각 노드의 다양한 매개변수를 글로벌 매개변수처럼 사용할 수 있어 추가 프로그래밍이나 컴파일 없이 변화 가능한 프로세스를 만들 수 있다.

파라미터 목록 확인(ros2 param list)

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

~$ ros2 param list
/teleop_turtle:
  scale_angular
  scale_linear
  use_sim_time
/turtlesim:
  background_b
  background_g
  background_r
  use_sim_time
```

- 현재 개발 환경의 실행 중인 노드의 파라미터 서버에 접근해 현재 사용 가능한 파라미터를 노드별로 목록화해 표시함
- use_sim_time 은 모든 노드의 기본 파라미터로 시뮬레이션할 때 사용하는 파라미터

파라미터 내용 확인(ros2 param describe)

```
~$ ros2 param describe /turtlesim background_b
Parameter name: background_b
  Type: integer
  Description: Blue channel of the background color
  Constraints:
    Min value: 0
    Max value: 255
    Step: 1
```

- tutlesim 노드의 background_b 파라미터의 데이터 형태는 integer
- 최소, 최대값은 0, 255

파라미터 읽기(ros2 param get)

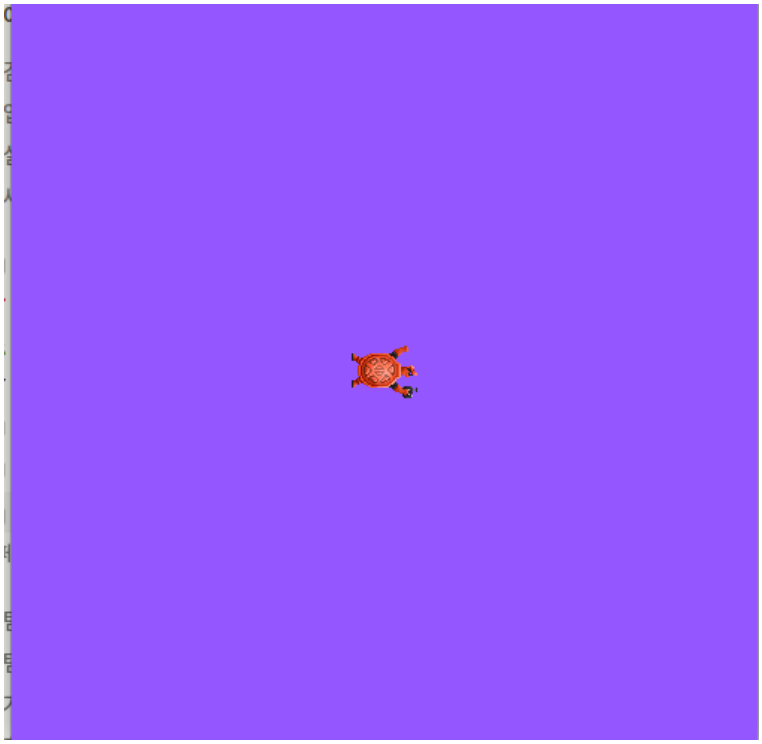
```
~$ ros2 param get <node_name> <parameter_name>
```

```
~$ ros2 param get /turtlesim background_b
Integer value is: 255
~$ ros2 param get /turtlesim background_g
Integer value is: 86
~$ ros2 param get /turtlesim background_r
Integer value is: 69
```

파라미터 쓰기(ros2 param set)

```
~$ ros2 param set <node_name> <param_name> <value>
```

```
~$ ros2 param set /turtlesim background_r 148
Set parameter successful
```



- 기본 배경색(background)이 바뀐다
- 파라미터는 별도의 추가 프로그래밍 없이 정해놓은 파라미터를 통해 외부에서 읽고, 쓸 수 있어 프로그램 수정과 컴파일 없이도 다른 결과 도출 가능

파라미터 저장(ros2 param dump)

앞에서 파라미터 값을 변경했지만 turtlesim 노드를 종료하고 다시 실행하면 다시 초기값으로 설정된다. 따라서 저장하고 이후 저장된 파라미터 값을 포함한 명령어로 실행해야함

```
~$ ros2 param dump /turtlesim
Saving to:  ./turtlesim.yaml # yaml형태로 저장된다.
```

```
~$ cat ./turtlesim.yaml
/turtlesim:
  ros__parameters:
    background_b: 255
```

```
    background_g: 86
    background_r: 148 # 변경된 부분이 잘 저장되어 있음
    use_sim_time: false
```

노드 실행 시 저장된 파라미터 값을 사용하려면

```
~$ ros2 run turtlesim turtlesim_node --ros-args --params-file ./turtlesim.yaml
[INFO] [1690448230.516725749] [turtlesim]: Starting turtlesim with node name /turtlesim
[INFO] [1690448230.518105204] [turtlesim]: Spawning turtle [turtle1] at x=[5.544445], y=[5.544445], theta=[0.000000]
--
```

파라미터 삭제(ros2 param delete)

```
~$ ros2 run param delete /turtlesim background_b
Deleted parameter successfully
```