ROS2 프로그래밍 입문 (9차시)

9. ROS2 복습_3

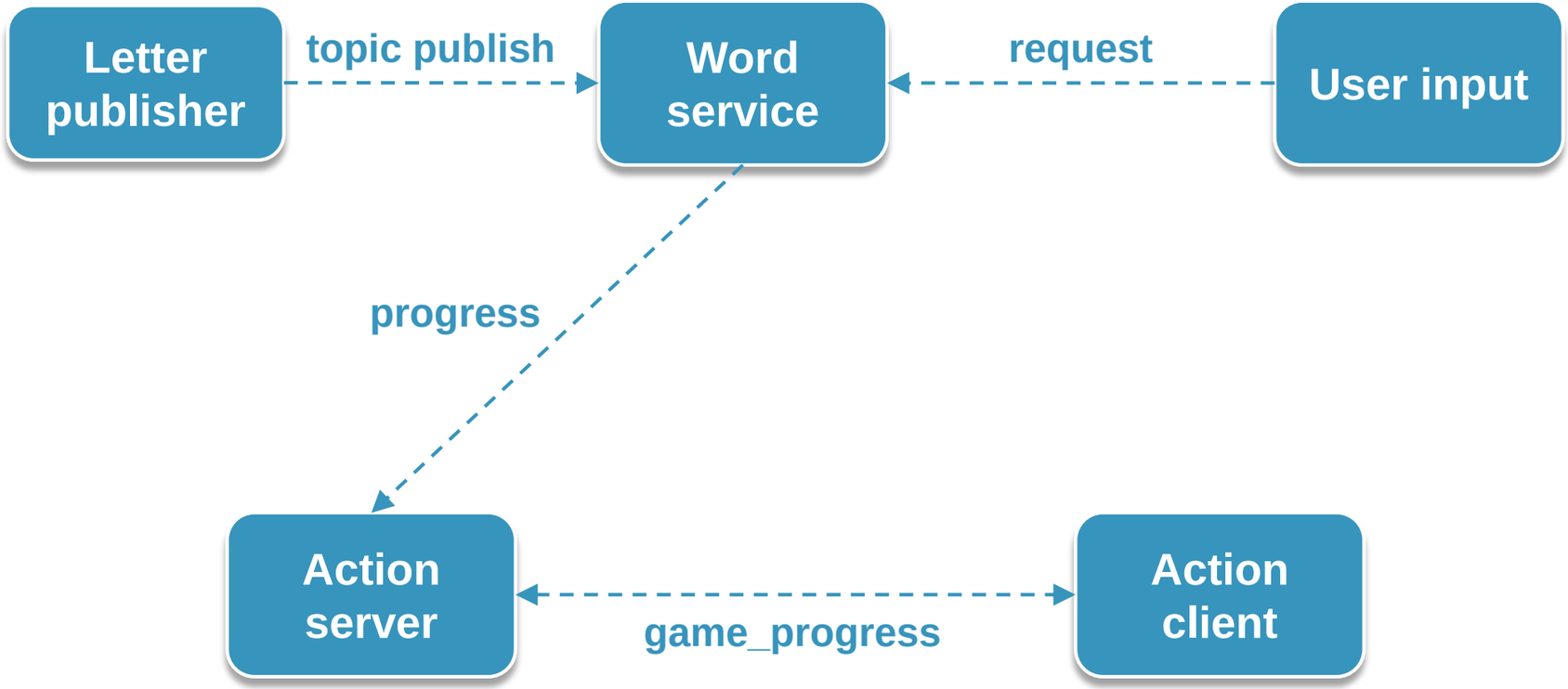
00

Contents

ROS2 복습_3

Hang-man - 실습

Hang-man 구조도



Hang-man - 실습

코드 구조

- **전체 코드의 구조는 다음과 같음**
  - 일부 파일은 지금부터 생성 예정

```
.
├── hangman_game
│   ├── hangman_game
│   │   ├── __init__.py
│   │   ├── letter_publisher.py
│   │   ├── progress_action_client.py
│   │   ├── progress_action_server.py
│   │   ├── user_input.py
│   │   └── word_service.py
│   ├── package.xml
│   ├── resource
│   │   └── hangman_game
│   ├── setup.cfg
│   └── setup.py
└── hangman_interfaces
    ├── action
    │   └── GameProgress.action
    ├── CMakeLists.txt
    ├── msg
    │   └── Progress.msg
    ├── package.xml
    └── srv
        └── CheckLetter.srv
```

hangman_interfaces

- **hangman_interfaces/msg/Progress.msg**
  - current_state : 현재 상태 ex) p y _ _ o n
  - attempts_left : 목숨(남은 시도 횟수)
  - game_over : 목숨이 다 소진 되었는지
  - won : 게임에서 이겼는지(목숨 소진 이전에 정답을 맞추었는지)

```
1   string current_state
2   int32 attempts_left
3   bool game_over
4   bool won
```

Hang-man - 실습

hangman_interfaces

- **hangman_interfaces/srv/CheckLetter.srv**
  - updated_word_state: 현재 상태 ex) p y _ _ o n
  - is_correct : 현재 user input으로 들어온 글자가 선택된 단어 내에 존재하는지에 대한 bool 타입 자료형
  - message : 맞았으면 "Correct"를 띄우고 틀리면 "WRONG"을 띄움

```
# Empty request
---
string updated_word_state
bool is_correct
string message
```

Hang-man - 실습

hangman_interfaces

- **hangman_interfaces/action/GameProgress.action**
  - game_over : 목숨이 다 소진 되었는지
  - won : 게임에서 이겼는지(목숨 소진 이전에 정답을 맞추었는지)

```
# Goal
# Empty since the client doesn't need to send any data
---
# Result
bool game_over
bool won
---
# Feedback
bool game_over
```

Hang-man - 실습

hangman_game

- hangman_game/hangman_game/letter_publisher.py의 전체 코드

```python
import rclpy
from rclpy.node import Node
from std_msgs.msg import String

class LetterPublisher(Node):

    def __init__(self):
        super().__init__('letter_publisher')
        self.publisher_ = self.create_publisher(String, 'letter_topic', 10)
        self.timer = self.create_timer(1.0, self.publish_letter)
        self.current_letter = ord('a')

    def publish_letter(self):
        msg = String()
        msg.data = chr(self.current_letter)
        self.publisher_.publish(msg)
        self.get_logger().info(f'Publishing: {msg.data}')
        self.current_letter += 1
        if self.current_letter > ord('z'):
            self.current_letter = ord('a')

def main(args=None):
    rclpy.init(args=args)
    letter_publisher = LetterPublisher()
    rclpy.spin(letter_publisher)
    letter_publisher.destroy_node()
    rclpy.shutdown(

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

Hang-man - 실습

hangman_game

- hangman_game/hangman_game/word_service.py의 전체 코드

```python
import rclpy
from rclpy.node import Node
from hangman_interfaces.srv import CheckLetter
from hangman_interfaces.msg import Progress
from std_msgs.msg import String
import random


class WordService(Node):

    def __init__(self):
        super().__init__("word_service")
        self.service = self.create_service(
            CheckLetter, "check_letter", self.check_letter_callback
        )
        self.subscription = self.create_subscription(
            String, "letter_topic", self.letter_callback, 10
        )
        self.progress_publisher = self.create_publisher(Progress, "progress", 10)

        data = Progress()
        data.current_state = ""
        data.attempts_left = 20
        data.game_over = False
        data.won = False
        self.progress_publisher.publish(data)

        self.current_letter = ""
        self.word_list = ["python", "hangman", "robot", "ros", "interface"]
        self.word = random.choice(self.word_list)
        self.word_state = ["_"] * len(self.word)
        self.get_logger().info(f"The word has {len(self.word)} letters.")
        self.attempts_left = 20  # Max attempts
```

Hang-man - 실습

hangman_game

- hangman_game/hangman_game/word_service.py의 전체 코드

```python
def letter_callback(self, msg):
    self.current_letter = msg.data

def check_letter_callback(self, request, response):
    letter = self.current_letter
    if letter in self.word:
        for idx, char in enumerate(self.word):
            if char == letter:
                self.word_state[idx] = letter
        response.is_correct = True
        response.message = "Correct!"
    else:
        self.attempts_left -= 1
        response.is_correct = False
        response.message = "WRONG"
    response.updated_word_state = "".join(self.word_state)
    self.get_logger().info(f"Received letter: {letter}")
    self.get_logger().info(f"Current word state: {response.updated_word_state}")

    # Publish progress
    progress_msg = Progress()
    progress_msg.current_state = response.updated_word_state
    progress_msg.attempts_left = self.attempts_left
    progress_msg.game_over = "_" not in self.word_state or self.attempts_left <= 0
    progress_msg.won = "_" not in self.word_state
    self.progress_publisher.publish(progress_msg)
    return response
```

Hang-man - 실습

hangman_game

- hangman_game/hangman_game/word_service.py의 전체 코드

```
def main(args=None):
    rclpy.init(args=args)
    letter_publisher = LetterPublisher()
    rclpy.spin(letter_publisher)
    letter_publisher.destroy_node()
    rclpy.shutdown(

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

Hang-man - 실습

hangman_game

- hangman_game/hangman_game/user_input.py 전체 코드

```python
# hangman_game/user_input.py
import rclpy
from rclpy.node import Node
from hangman_interfaces.srv import CheckLetter
import threading

class UserInput(Node):

    def __init__(self):
        super().__init__("user_input")
        self.cli = self.create_client(CheckLetter, "check_letter")
        while not self.cli.wait_for_service(timeout_sec=1.0):
            self.get_logger().info("Service not available, waiting...")
        self.req = CheckLetter.Request()
        self.get_logger().info("Press Enter to check the current letter.")
        threading.Thread(target=self.input_thread, daemon=True).start()

    def input_thread(self):
        while True:
            input("Press Enter to input the current letter.")
            self.send_request()

    def send_request(self):
        future = self.cli.call_async(self.req)

def main(args=None):
    rclpy.init(args=args)
    user_input = UserInput()
    rclpy.spin(user_input)
    user_input.destroy_node()
    rclpy.shutdown()

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

Hang-man - 실습

hangman_game

- hangman_game/hangman_game/progress_action_client.py 전체 코드

```python
import rclpy
from rclpy.node import Node
from hangman_interfaces.action import GameProgress
from rclpy.action import ActionClient


class ProgressActionClient(Node):

    def __init__(self):
        super().__init__("progress_action_client")
        self._action_client = ActionClient(self, GameProgress, "game_progress")
        self.result_received = False
        self.send_goal()

    def send_goal(self):
        self.get_logger().info("Action Client: Waiting for action server...")
        self._action_client.wait_for_server()
        goal_msg = GameProgress.Goal()
        self.get_logger().info("Action Client: Sending goal request...")
        self._send_goal_future = self._action_client.send_goal_async(
            goal_msg, feedback_callback=self.feedback_callback
        )
        self._send_goal_future.add_done_callback(self.goal_response_callback)

    def feedback_callback(self, feedback_msg):
        feedback = feedback_msg.feedback
        if feedback.game_over:
            self.get_logger().info("Action Client: Game over detected in feedback")
```

Hang-man - 실습

hangman_game

- hangman_game/hangman_game/progress_action_client.py 전체 코드

```python
    def goal_response_callback(self, future):
        goal_handle = future.result()
        if not goal_handle.accepted:
            self.get_logger().info("Action Client: Goal rejected")
            self.result_received = True
            return

        self.get_logger().info("Action Client: Goal accepted")
        self._get_result_future = goal_handle.get_result_async()
        self._get_result_future.add_done_callback(self.get_result_callback)

    def get_result_callback(self, future):
        result = future.result().result
        if result.won:
            self.get_logger().info("Action Client: Congratulations! You won!")
        else:
            self.get_logger().info("Action Client: Game Over. You lost.")
        self.result_received = True


def main(args=None):
    rclpy.init(args=args)
    action_client = ProgressActionClient()
    while rclpy.ok():
        rclpy.spin_once(action_client)
        if action_client.result_received:
            break
    action_client.destroy_node()
    rclpy.shutdown()


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

Hang-man - 실습

hangman_game

- hangman_game/hangman_game/progress_action_server.py 전체 코드

```python
import rclpy
from rclpy.node import Node
from hangman_interfaces.action import GameProgress
from hangman_interfaces.msg import Progress
from rclpy.action import ActionServer
from rclpy.executors import MultiThreadedExecutor
import time
import threading


class ProgressActionServer(Node):

    def __init__(self):
        super().__init__("progress_action_server")
        self._action_server = ActionServer(
            self, GameProgress, "game_progress", self.execute_callback
        )
        self.current_progress = Progress()
        self.progress_received_event = threading.Event()

        # Subscribe to the 'progress' topic to get game updates
        self.subscription = self.create_subscription(
            Progress, "progress", self.progress_callback, 10
        )
        self.subscription

        self.get_logger().info("Action Server Initialized")
        self.get_logger().info(f"GAME OVER: {self.current_progress.game_over}")
        self.get_logger().info(f"WON: {self.current_progress.won}")
```

Hang-man - 실습

hangman_game

- hangman_game/hangman_game/progress_action_server.py 전체 코드

```python
    def progress_callback(self, msg):
        self.current_progress = msg
        self.get_logger().info(
            f"Progress updated: {self.current_progress.current_state}"
        )

    def execute_callback(self, goal_handle):
        self.get_logger().info("Action Server: Received goal request")
        feedback_msg = GameProgress.Feedback()
        update_rate = 1.0  # seconds

        while not self.current_progress.game_over:
            # Publish feedback
            feedback_msg.game_over = self.current_progress.game_over
            goal_handle.publish_feedback(feedback_msg)
            self.get_logger().info(
                f"Current State: {self.current_progress.current_state}"
            )
            self.get_logger().info(
                f"Attempts Left: {self.current_progress.attempts_left}"
            )

            time.sleep(update_rate)

        # Game is over
        result = GameProgress.Result()
        result.game_over = self.current_progress.game_over
        result.won = self.current_progress.won

        if self.current_progress.won:
            self.get_logger().info("Action Server: Congratulations! You won!")
        else:
            self.get_logger().info("Action Server: Game Over. You lost.")

        goal_handle.succeed()
        self.get_logger().info("Action Server: Goal succeeded")
        return result
```

Hang-man - 실습

hangman_game

- hangman_game/hangman_game/progress_action_server.py 전체 코드

```python
def main(args=None):
    rclpy.init(args=args)
    action_server = ProgressActionServer()

    executor = MultiThreadedExecutor()
    executor.add_node(action_server)
    try:
        executor.spin()
    except KeyboardInterrupt:
        pass
    finally:
        action_server.destroy_node()
        rclpy.shutdown()


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

Hang-man - 실습

hangman_game

- **hangman_game/setup.py**
  - entry_points를 다음과 같이 변경

```
entry_points={
        'console_scripts': [
            'letter_publisher = hangman_game.letter_publisher:main',
            'word_service = hangman_game.word_service:main',
            'user_input = hangman_game.user_input:main',
            'progress_action_server = hangman_game.progress_action_server:main',
            'progress_action_client = hangman_game.progress_action_client:main',
        ],
    },
```

Hang-man - 실습

hangman_game

- **hangman_interfaces/CMakeLists.txt**

```
cmake_minimum_required(VERSION 3.8)
project(hangman_interfaces)

if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
  add_compile_options(-Wall -Wextra -Wpedantic)
endif()

# find dependencies
find_package(ament_cmake REQUIRED)
find_package(builtin_interfaces REQUIRED)
find_package(rosidl_default_generators REQUIRED)

ament_export_dependencies(rosidl_default_runtime)

set(msg_files
  "msg/Progress.msg"
)

set(srv_files
  "srv/CheckLetter.srv"
)

set(action_files
  "action/GameProgress.action"
)

rosidl_generate_interfaces(${PROJECT_NAME}
  ${msg_files}
  ${srv_files}
  ${action_files}
  DEPENDENCIES builtin_interfaces
)

if(BUILD_TESTING)
  find_package(ament_lint_auto REQUIRED)
  set(ament_cmake_copyright_FOUND TRUE)
  set(ament_cmake_cpplint_FOUND TRUE)
  ament_lint_auto_find_test_dependencies()
endif()

ament_package()
```

Hang-man - 실습

hangman_game

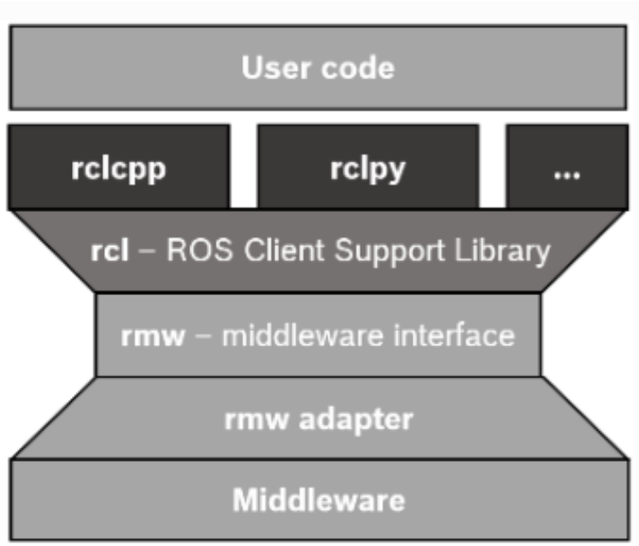
- hangman_interfaces/package.xml

```xml
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd"
schematypens="http://www.w3.org/2001/XMLSchema"?>
<package format="3">
  <name>hangman_interfaces</name>
  <version>0.0.0</version>
  <description>TODO: Package description</description>
  <maintainer email="">user</maintainer>
  <license>TODO: License declaration</license>

  <buildtool_depend>ament_cmake</buildtool_depend>
  <buildtool_depend>rosidl_default_generators</buildtool_depend>

  <exec_depend>builtin_interfaces</exec_depend>
  <exec_depend>rosidl_default_runtime</exec_depend>
  <member_of_group>rosidl_interface_packages</member_of_group>

  <test_depend>ament_lint_auto</test_depend>
  <test_depend>ament_lint_common</test_depend>

  <export>
    <build_type>ament_cmake</build_type>
  </export>
</package>
```

Executor
ROS2계층구조

▶ ROS2 계층 구조

1. **User code :** 사용자가 작성한 코드
2. **rcl :** 클라이언트 라이브러리로, 사용자 코드와 미들웨어를 연결
3. **rmw :** 미들웨어와 클라이언트 간의 인터페이스 역할. 미들웨어와의 상호작용을 추상화
4. **rmw adapter :** RMW와 미들웨어를 연결하는 중간 계층 역할. 이를 통해 ROS2는 특정 DDS 구현체에 종속되지 않고, 다양한 미들웨어를 지원할 수 있는 유연성을 가짐
5. **Middleware :** 실제 메시지 전달과 QoS 설정을 처리하는 계층



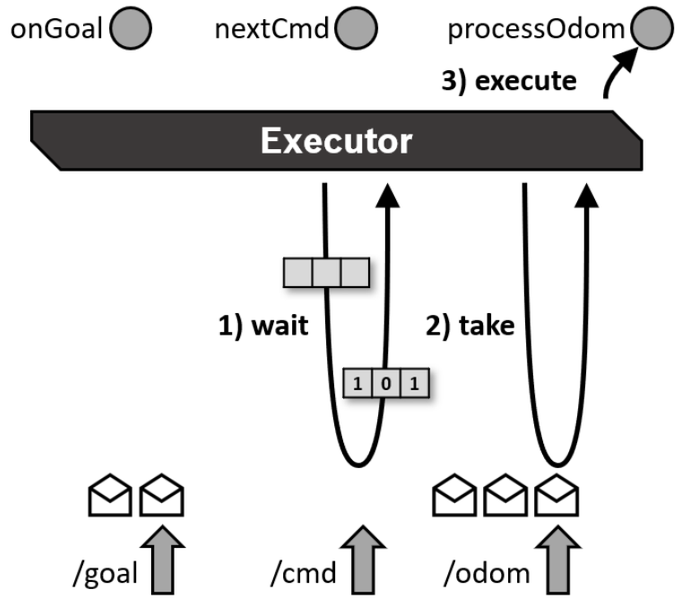
- **미들웨어 :** 분산 네트워크에서 애플리케이션 또는 구성 요소 간에 하나 이상의 종류의 통신 또는 연결을 가능하게 하는 소프트웨어. ROS2에서는 노드 간의 메시지 송수신 및 이벤트 처리를 담당

Executor

Executor 동작과정

▶ 동작 과정

1. **wait :** 미들웨어에 메시지가 도착할 때까지 대기
2. **take :** 새로운 메시지 도착 시, 해당 메시지를 가져옴. 이 과정에서 미들웨어에 저장된 메시지가 클라이언트로 전달됨
3. **execute :** 메시지 처리를 위한 콜백함수(onGoal, nextCmd, processOdom) 실행

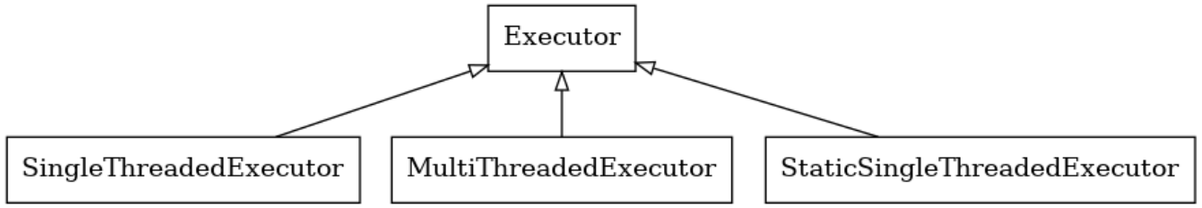


Executor

Executor의 종류

▶ SingleThreadedExecutor : 단일 스레드에서 콜백을 실행

- 하나의 스레드만 사용하여 이벤트를 처리하므로, 콜백이 완료될 때까지 다른 작업을 처리할 수 없음
- 주로 처리 속도가 중요한 것이 아니거나, 콜백이 충돌하지 않도록 하기 위해 단일 스레드 환경에서 사용




```
from rclpy.executors

import SingleThreadedExecutor

executor = SingleThreadedExecutor()
executor.add_node(my_node)
executor.spin() # 이벤트 루프 실행
```

▶ StaticSingleThreadedExecutor : 단일 스레드에서 정적으로 콜백을 실행

- 정적 단일 스레드 실행자는 구독, 타이머, 서비스 서버, 액션 서버 등의 노드 구조를 스캔하는 런타임 비용을 최적화
- 노드가 추가될 때 콜백 스캔을 한 번만 수행되며, 다른 두 Executer는 이러한 변화를 정기적으로 스캔
- 정적 단일 스레드 실행자는 초기화 중에 모든 구독, 타이머 등을 생성하는 노드와 함께 사용해야 함

Executor

Multi Thread Executor

▶ MultiThreadedExecutor : 여러 스레드에서 콜백을 병렬로 실행

- 여러 스레드가 동시에 실행되기 때문에 복잡한 작업이나 멀티태스킹 환경에서 유리
- 그러나 다중 스레드 간의 자원 경쟁이나 동기화 문제가 발생할 수 있으므로, 적절한 동기화 처리(locking) 필요

```
from rclpy.executors import MultiThreadedExecutor

executor = MultiThreadedExecutor(num_threads=4)  # 4개의 스레드를 사용
executor.add_node(my_node)
executor.spin()
```

Executor

Executor 기본동작

▶ Executor 동작 순서

1. **이벤트 감지 :** 노드에서 수신된 토픽, 서비스 요청, 타이머 이벤트 등을 감지
2. **콜백 큐 생성 :** 이벤트가 발생할 때마다 대응하는 콜백을 큐(queue)에 수집
3. **콜백 실행 :** 큐에 있는 콜백을 적절한 순서대로 실행. SingleThreadedExecutor는 하나씩 처리하고, MultiThreadedExecutor는 병렬로 처리

Executor

Executor 사용 예

▶ Executor 사용 예

```python
import rclpy
from rclpy.node import Node
from std_msgs.msg import String

class MyNode(Node):
    def __init__(self):
        super().__init__('my_node')
        self.publisher = self.create_publisher(String, 'my_topic', 10)
        self.timer = self.create_timer(1.0, self.timer_callback)

    def timer_callback(self):
        msg = String()
        msg.data = 'Hello ROS2'
        self.publisher.publish(msg)
        self.get_logger().info(f'Published: {msg.data}')
```

```python
def main(args=None):
    rclpy.init(args=args)
    node = MyNode()

    # Single-threaded executor 예시
    executor = rclpy.executors.SingleThreadedExecutor()
    executor.add_node(node)

    try:
        executor.spin()  # 이벤트 루프 실행
    except KeyboardInterrupt:
        pass
    finally:
        node.destroy_node()
        rclpy.shutdown()

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

- SingleThreadedExecutor를 사용하는 간단한 퍼블리셔 노드
- 이벤트는 1초마다 발생하여 메시지를 퍼블리싱

Executor
Executor 와 spin 차이점

▶ Executor 와 spin 차이점

	Executor	spin()
역할	콜백 관리 및 실행	이벤트 루프 실행
설명	▪ 노드에 포함된 콜백(토픽, 서비스, 타이머 등)을 관리하고, 이벤트가 발생하면 적절한 콜백을 실행하는 역할 수행 ▪ SingleThreadedExecutor와 MultiThreadedExecutor 같은 다양한 종류가 있으며, 콜백을 실행하는 방식에 따라 동작 방식이 달라짐	▪ Executor가 이벤트를 처리하는 루프를 실행 ▪ spin()이 호출되면 노드는 콜백이 발생하기를 기다리며, 이벤트가 감지될 때 콜백을 실행 ▪ Executor가 동작할 수 있는 환경을 유지하는 루프이며, 명시적으로 종료하거나 프로그램이 종료될 때까지 계속 실행

Executor

Executor 와 spin 의 관계

▶ 관계

- Executor는 콜백을 처리하는 "관리자"이고, spin()은 해당 Executor가 콜백을 계속해서 처리하도록 하는 "루프"
- Executor는 콜백을 실행하는 규칙과 방식을 정의하고, spin()은 그 규칙에 따라 Executor가 동작하도록 해주는 실행 메커니즘

▶ 멀티스레드와의 연관성

- spin()을 사용하면 Executor가 콜백을 처리하는 동안 계속해서 대기하지만, MultiThreadedExecutor를 사용하면 여러 콜백을 동시에 병렬로 처리할 수 있음
- 이때도 spin()을 호출하여 이벤트 루프가 유지되지만, 여러 스레드가 동시에 동작하면서 여러 콜백을 병렬 처리 가능

SingleThreadedExecutor + spin(): 한 번에 하나의 콜백만 처리
MultiThreadedExecutor + spin(): 여러 콜백을 동시에 처리

ROS2 bag 이해 및 사용법

명령어

▶ BAG 파일 레코딩

- 다음 명령어는 “topic_name”에서 수신되는 메시지를 “my_bag.bag”라는 이름의 BAG 파일에 레코딩 함

```
sparkx@sparkx-750XDA:~$ ros2 bag record -o my_bag topic_name
```

▶ BAG 파일 재생

- 다음 명령어는“my_bag.bag”라는 이름의 BAG 파일을 재생함

```
sparkx@sparkx-750XDA:~$ ros2 bag play my_bag
```

▶ BAG 파일 정보 표시

- 다음 명령어는“my_bag.bag”라는 이름의 BAG 파일의 정보를 표시함

```
sparkx@sparkx-750XDA:~$ ros2 bag info my_bag
```

BAG 파일 만들기

실습

세번째 터미널에서 다음 명령어를 이용하여 저장되어 있는 BAG 파일 재생

- 간혹 인터럽트 등의 이유로 turtle의 궤적이 기록 당시와 정확히 일치하지 않을 수 있음
- 이 문제를 방지하려면 turtle의 움직임을 기록할 때, 각 명령이 완전히 완료된 후 다음 명령을 실행해야 함

```
sparkx@sparkx-750XDA:~$ ros2 bag play turtle_bag
[INFO] [1728012245.437376268] [rosbag2_storage]: Opened database 'turtle_bag/turtle_bag_0.db3' for READ_ONLY.
[INFO] [1728012245.437414199] [rosbag2_player]: Set rate to 1
[INFO] [1728012245.439882832] [rosbag2_player]: Adding keyboard callbacks.
[INFO] [1728012245.439901765] [rosbag2_player]: Press SPACE for Pause/Resume
[INFO] [1728012245.439908007] [rosbag2_player]: Press CURSOR_RIGHT for Play Next Message
[INFO] [1728012245.439915295] [rosbag2_player]: Press CURSOR_UP for Increase Rate 10%
[INFO] [1728012245.439920372] [rosbag2_player]: Press CURSOR_DOWN for Decrease Rate 10%
[INFO] [1728012245.440136526] [rosbag2_storage]: Opened database 'turtle_bag/turtle_bag_0.db3' for READ_ONLY.
```

BAG 파일 실습

실습

BAG 파일에 기록된 2d 카메라 정보 불러오기

- 아래 제공된 링크에서 rosbag2_video.tar.gz 파일을 다운로드 받은 후 압축 풀기
  https://drive.google.com/drive/folders/1zjGVRD5YQkiM_yLwjGOGRF5cruz8-Wsn
- 다운로드 된 압축 파일의 압축 풀기

```
tar -zxvf rosbag2_video.tar.gz
```

BAG 파일 실습

실습

BAG 파일에 기록된 2d 카메라 정보 불러오기

- 압축 풀린 bag 파일의 정보 확인

```
ros2 bag info rosbag2_video
```

```
user@user:~/Downloads$ ros2 bag info rosbag2_video

Files:             rosbag2_video_0.db3
Bag size:          44.0 MiB
Storage id:        sqlite3
Duration:          10.077088296s
Start:             Dec  5 2024 11:27:45.740351043 (1733365665.740351043)
End:               Dec  5 2024 11:27:55.817439339 (1733365675.817439339)
Messages:          304
Topic information: Topic: /video_frames | Type: sensor_msgs/msg/Image | Count: 304 | Serialization Format: cdr
```

- Bag 파일 반복 재생

```
ros2 bag play rosbag2_video --loop
```

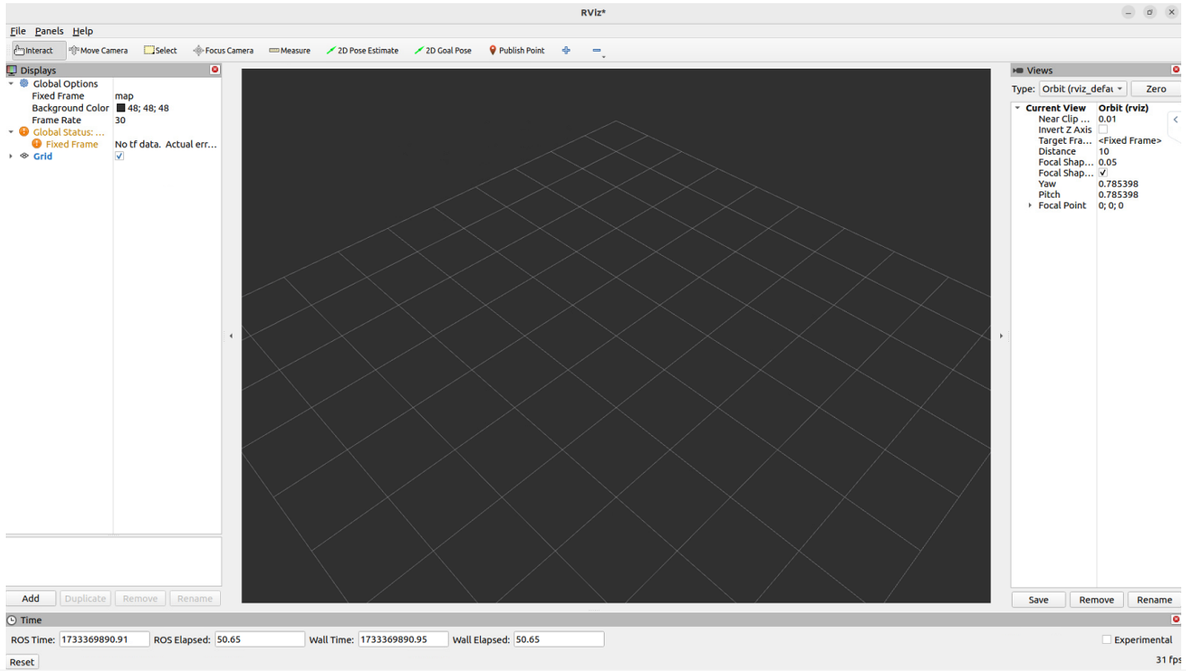
BAG 파일 실습

실습

BAG 파일에 기록된 2d 카메라 정보 불러오기

- 새로운 터미널에서 rviz 실행

```
rviz2
```

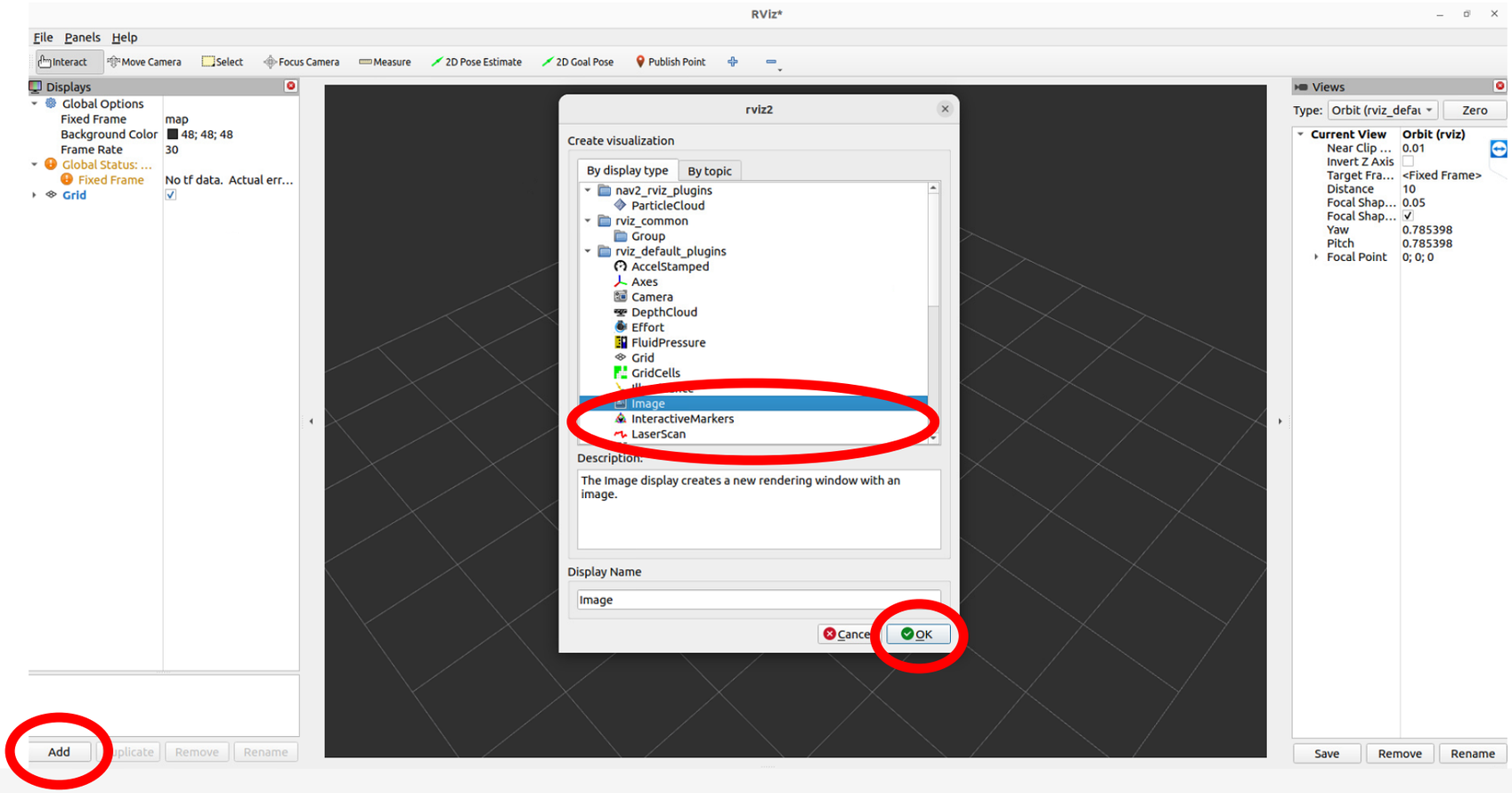


BAG 파일 실습

실습

BAG 파일에 기록된 2d 카메라 정보 불러오기

- Rviz에서 add버튼을 누른 후 Image 선택 후 OK 버튼 누르기

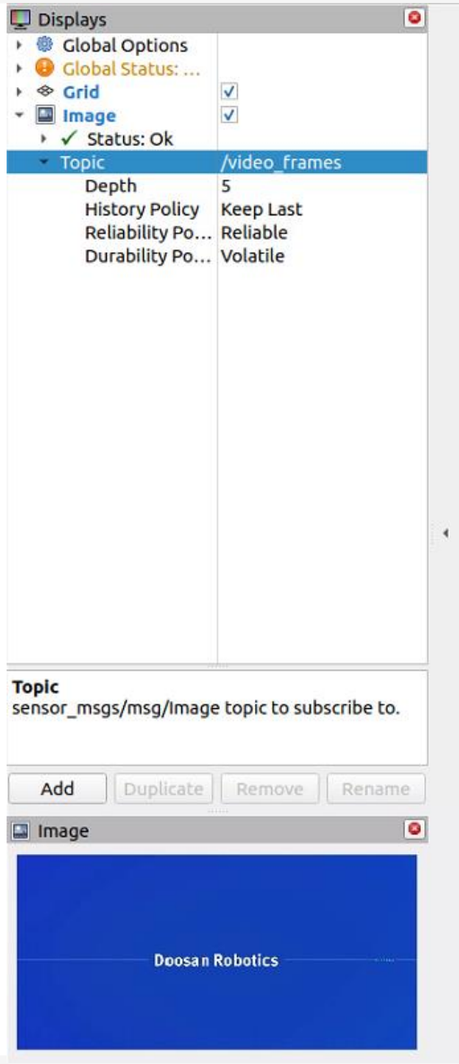


BAG 파일 실습

실습

BAG 파일에 기록된 2d 카메라 정보 불러오기

- 사이드바에서 이미지의 topic name을 “ \ video_frames” 로 바꾸기
- 하단의 Image 창에 영상이 재생되는 것을 확인



영상 출처: https://www.youtube.com/watch?v=29iFysOZg3Q

Jupyter를 이용한 프로그래밍
VSCode에서 Jupyter 사용

▶ Visual Studio Code - extension

- **Extension의 주요 역할 및 기능**
  - 프로그래밍 언어 지원 추가: 추가 언어를 지원하거나 기존 언어의 기능을 확장 (예: Python, YAML등)
  - 생산성 향상 도구: 코딩, 탐색, 리팩토링, 반복 작업을 자동화해 생산성 향상 (예: XML Tools, Markdown All in One)
  - 특정 기술 또는 프레임워크 지원: 특정 프레임워크나 기술을 위한 추가 기능 제공 (예: ROS, URDF)
  - 자동화 및 DevTools: 반복 작업을 자동화하거나 개발 환경을 확장 (예: Colcon Tasks)

프로그래밍 응용

Service 와 Action 에서의 Client 차이

Visual Studio Code - extension

항목	**Service Client**	**Action Client**
목적	요청-응답 1회성 호출	장시간 작업의 관리
단계 수	1단계 (call_async)	3단계 (send_goal_async → GoalHandle → get_result_async)
중간 상태	없음	진행 상태(feedback), 취소 등 처리 가능
.result() 위치	call_async 후 future에서	get_result_async 후 future에서
피드백 처리	불가	가능 (feedback_callback)
쓰임새	빠른 계산 요청 (예: IK)	장기 제어 작업 (예: 궤적 추종)