ROS_4일차__보고서_20기_인턴_2025402040_컴퓨터정보공학부_노기문

-Lifecycle node + Qos 설정한 publisher, subscriber 패키지 구현-

목차

1. Lifecycle node

LifecycleNode는 ROS2에서 노드의 실행 단계를 명확히 구분하고 제어할 수 있도록 만든 특수한 노드 타입이다. 일반 Node와 달리, State와 Transition개념을 가진다.

주요 특징

State 기반 실행
unconfigured → inactive → active → finalized 등의 상태가 있음.
노드는 단순히 생성되자마자 바로 동작하지 않고, 명확한 단계별로 초기화와 활성화를 거친다.

Transition 관리
configure, activate, deactivate, cleanup, shutdown 같은 이벤트로 상태를 바꾼다.
각 전이마다 콜백 함수를 정의할 수 있어, 리소스 할당/해제·통신 시작/정지 같은 동작을 제어할 수 있다.

관리 용이성
복잡한 로봇 시스템에서 센서, 액추에이터 같은 노드의 안정적 초기화와 안전한 중지가 가능하다.
예: 센서 드라이버를 configure 단계에서 장치 열기 → activate에서 토픽 퍼블리시 시작 → deactivate에서 퍼블리시 중지 → cleanup에서 장치 해제.
또한 퍼블리셔/서스크라이버 관계에서도 적용이 가능하다.
예: service client를 통해 명령을 내림 → configure단계에서 준비완료 → activate에서 토픽 퍼블리시 시잣 → deactivate에서 퍼블리시 중지 → cleanup에서 한번 초기화 → shutdown 시 finalized로 이동

외부 제어 가능
ros2 lifecycle CLI 명령이나 서비스 호출을 통해 노드 상태를 원격으로 바꿀 수 있다.
따라서 런타임 중에도 안전하게 노드를 재시작하거나 비활성화할 수 있다.

일반 노드와 차이
일반 Node: 생성과 동시에 동작 시작 → 종료 시까지 동일한 상태.
LifecycleNode: 상태 머신을 갖고 있어 단계별 제어가 가능 → 더 정교한 관리.

state설명

unconfigured
노드가 생성된 직후 상태. 아직 리소스 초기화 안 됨.

inactive
초기화는 되었으나 동작은 하지 않는 상태. 토픽 퍼블리시·구독, 서비스, 액션 등 멈춰 있음.

active
정상 동작 상태. 퍼블리시/구독·서비스·액션 수행.

finalized
종료된 상태. 더 이상 어떤 전이도 불가.

transition 설명

configure: unconfigured → inactive

activate: inactive → active

deactivate: active → inactive

cleanup: inactive → unconfigured

shutdown: 어떤 상태에서든 finalized로 이동

LifecycleNode의 상태 전환은 토픽이 아니라 서비스(Service) 인터페이스를 통해 제어된다.

ROS2는 각 LifecycleNode마다 자동으로 lifecycle 관리용 서비스를 생성한다.
예:
/my_node/change_state (상태 전환 요청)
/my_node/get_state (현재 상태 조회)
/my_node/get_available_states
/my_node/get_available_transitions

따라서 외부에서 ros2 lifecycle CLI나 직접 서비스 호출을 통해 상태를 변경한다.

ros2 lifecycle get /my_node
ros2 lifecycle set /my_node configure
ros2 lifecycle set /my_node activate

정리
상태 전환은 토픽이 아니라 서비스 요청으로 이루어진다.
노드 내부 로직은 상태 전환 이벤트 콜백(on_configure, on_activate, on_deactivate, on_cleanup, on_shutdown)에서 실행된다.

2. QoS(Quality of Service)

QoS(Quality of Service)는 ROS2에서 노드 간 통신 품질을 제어하는 정책 집합이다. 퍼블리셔와 서브스크라이버가 어떤 방식으로 메시지를 주고받을지 결정한다.

주요 QoS 정책

Reliability (신뢰성)
RELIABLE : 메시지가 손실 없이 도착할 때까지 재전송 보장. (네트워크 지연 증가 가능)
BEST_EFFORT : 최대한 빨리 보내지만 손실될 수도 있음. (센서 스트리밍에 적합)

Durability (지속성)
VOLATILE : 새로 구독 시작한 노드는 과거 메시지를 받지 않음.
TRANSIENT_LOCAL : 퍼블리셔가 보낸 마지막 메시지를 캐시해 두었다가 새 구독자가 들어오면 즉시 전달.

History (저장 정책)
KEEP_LAST(n) : 최근 n개 메시지만 저장.
KEEP_ALL : 리소스가 허용하는 한 모든 메시지를 저장.

Depth (큐 크기)
KEEPLAST일 때 몇 개 메시지를 버퍼링할지 결정.
예: depth=10 → 구독자가 처리 속도가 늦으면 10개까지만 저장하고 이후 것은 drop.

Deadline / Lifespan (선택적 고급 옵션)
deadline : 메시지가 일정 주기 안에 반드시 와야 함.
lifespan : 메시지가 유효한 시간. 만료되면 drop.

3. 코드 및 구현 설명

1. service client package

이 패키지는 두 개의 라이프사이클,Qos 패키지안의 노드(talker, subscriber)에 대해 서비스 클라이언트로 상태 조회·전이(Configure/Activate/Deactivate/Cleanup/Shutdown)를 동시 수행하는 서비스 클라이언트입니다. 입력으로 메뉴를 골라 전이를 보냅니다.

구성요소

1. struct Lc : 한 대상 노드에 대한 핸들.
    name: 대상 노드 네임스페이스.
    change: /<name>/change_state 서비스 클라이언트.
    get: /<name>/get_state 서비스 클라이언트.

2. class Controller : public rclcpp::Node
    노드명: lifecycle_controller.
    파라미터 선언:
        talker 기본값 /lifecycle_talker
        listener 기본값 /lifecycle_listener
        make(n): 대상 이름 n으로 ChangeState/GetState 클라이언트 생성.
        wait(h): 두 서비스가 준비될 때까지 500ms 간격 대기. Ctrl+C 등으로 종료되면 예외.
        loop(): 메뉴 입력 루프. 숫자 선택 → 전이 ID 매핑 → apply_both() 호출.
        apply_both(id): 두 노드에 같은 전이 요청 비동기 전송. 3초 대기. 성공/실패 출력 후 show()로 현재 상태 출력.
        show(): 두 노드의 get_state 호출로 상태 문자열 출력.
        sname(id): 상태 ID →
                "unconfigured"|"inactive"|"active"|"finalized"|unknown 변환.
        help(): 메뉴와 대상 노드명 안내.

3. main
    rclcpp::init → Controller 생성·loop() 실행 → 예외 출력 → rclcpp::shutdown.

2. subscriber package

이 패키지의 노드는 라이프사이클 노드가 ACTIVE일 때만 /chatter를 구독한다. QoS는 파라미터로 결정한다. DEACTIVATE/CLEANUP/SHUTDOWN에서 구독을 해제한다.

주요 동작

노드 타입: rclcpp_lifecycle::LifecycleNode("lifecycle_listener")

파라미터

topic 기본 /chatter

qos.* 세트로 신뢰성·내구성·히스토리·깊이·라이블리니스·리스 기간을 받음

상태 콜백

on_configure 설정값 로깅

on_activate 구독 생성 → 수신 시 recv: ... 로그

on_deactivate / on_cleanup / on_shutdown 구독 포인터 reset()로 해제

실행 루프: SingleThreadedExecutor로 라이프사이클 인터페이스를 스핀

QoS 적용 흐름

파라미터 선언 (load_qos)

qos.reliability → "reliable" 또는 "best_effort"

qos.durability → "volatile" 또는 "transient_local"

qos.history → "keep_last" 또는 "keep_all"

qos.depth → keep_last일 때 몇 개까지 저장할지

qos.liveliness → "automatic" 또는 "manual_by_topic"

qos.liveliness_lease_ms → 라이블리니스 유지 시간(ms)

QoS 객체 생성 (make_qos)

```
rclcpp::QoS qos = (c.history == "keep_all")
                  ? rclcpp::QoS(rclcpp::KeepAll())
                  : rclcpp::QoS(rclcpp::KeepLast(c.depth));
```

메시지 히스토리 정책 결정 이후 reliability, durability, liveliness 설정 적용

subscribe 생성

```
sub_ = create_subscription<std_msgs::msg::String>(
                  topic_, qos, callback);
```

=> qos 설정이 실제 subscriber에 반영됨

3. publisher package

이 패키지의 노드는 라이프사이클 퍼블리셔 노드입니다. 활성화(activate) 상태일 때만 문자열 메시지를 주기적으로 퍼블리시합니다. 그리고 구독자와 마찬가지로 QoS 설정을 파라미터로 받아 적용합니다.

동작 흐름

생성자

```
LifecycleTalker() : LifecycleNode("lifecycle_talker") {
        topic_ = declare_parameter<std::string>("topic", "/chatter");
       qos_ = load_qos();
        period_ms_ = declare_parameter<int>("period_ms", 500);
       }
```

퍼블리시할 토픽 이름 /chatter 선언
QoS 관련 파라미터 로드 (qos.*)
메시지 발행 주기(기본 500ms) 파라미터화

상태 전이 콜백

on_configure
퍼블리셔 생성. 아직 활성화는 안 됨.
on_activate
퍼블리셔 활성화 + 타이머 생성 → 주기적으로 "hello 0", "hello 1" 같은 메시지 발행.
on_deactivate
타이머 중단 + 퍼블리셔 비활성화.
on_cleanup / on_shutdown
퍼블리셔와 타이머 객체 삭제.

퍼블리시 예시 실행 흐름

ros2 lifecycle set /lifecycle_talker configure
→ 퍼블리셔 준비됨.
ros2 lifecycle set /lifecycle_talker activate
→ "hello 0", "hello 1" … 메시지 발행 시작.
ros2 lifecycle set /lifecycle_talker deactivate
→ 메시지 발행 중단.

QoS 적용 방식

load_qos()에서 파라미터를 읽어 QosCfg에 저장하고, make_qos()에서 실제 rclcpp::QoS 객체로 변환합니다.

```
rclcpp::QoS qos = (c.history == "keep_all")
        ? rclcpp::QoS(rclcpp::KeepAll())
        : rclcpp::QoS(rclcpp::KeepLast(c.depth));
if (c.reliability == "best_effort") qos.best_effort(); else qos.reliable();
if (c.durability == "transient_local") qos.transient_local(); else
qos.durability_volatile();
if (c.liveliness == "manual_by_topic")
qos.liveliness(rclcpp::LivelinessPolicy::ManualByTopic);
else qos.liveliness(rclcpp::LivelinessPolicy::Automatic);
if (c.lease_ms > 0)
qos.liveliness_lease_duration(rclcpp::Duration(std::chrono::milliseconds(c.lease_ms)));
```

각 항목 동작

History

keep_last(N) → 최근 N개만 큐에 저장 (기본 10)

keep_all → 모든 메시지 저장 (메모리 증가 가능)

Reliability

reliable → 메시지 손실 없이 보장 (재전송)

best_effort → 가능한 한 빨리, 손실 가능

Durability

volatile → 구독자가 연결된 이후의 메시지만 받음

transient_local → 발행자가 캐시한 최신 메시지를 새 구독자에게 전달

Liveliness

automatic → RMW가 자동으로 생존 신호 전송

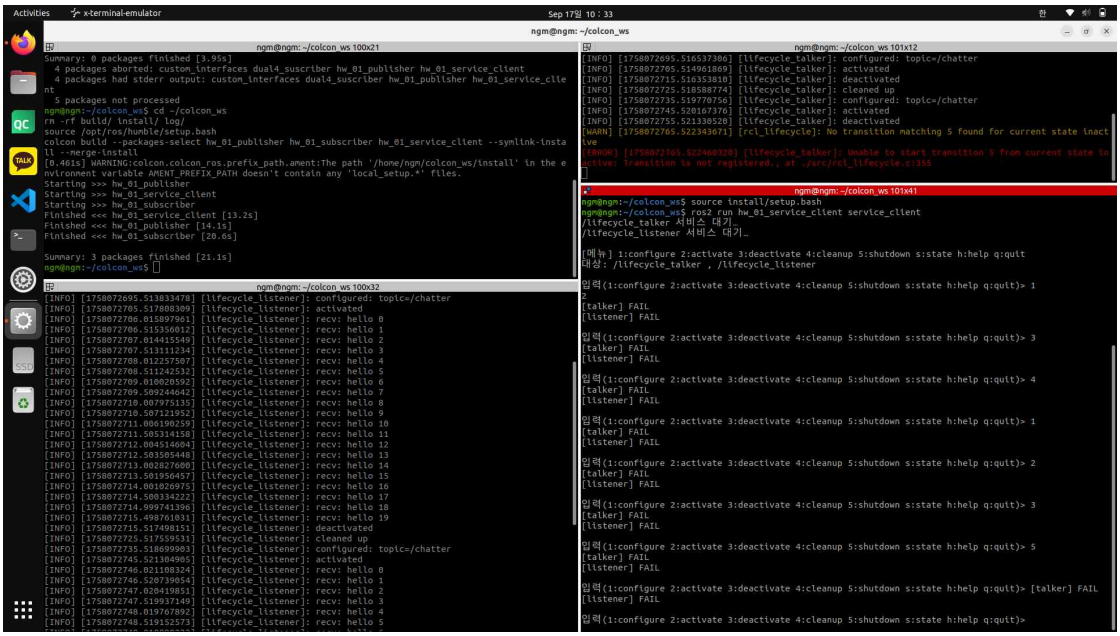
manual_by_topic → 애플리케이션이 직접 liveliness ping을 보내야 함

Lease Duration

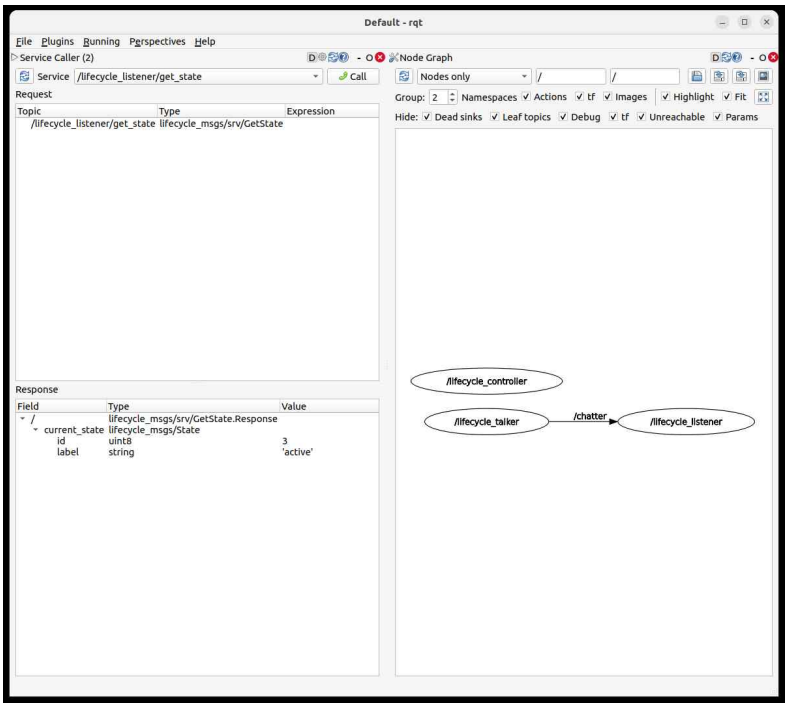
liveliness 보장 기간(ms). 지정 시간 내 신호가 없으면 발행자가 죽었다고 판단

4. Result screen

terminal



RQT



-끝-