ROS2 DAY4
과제 1번 보고서

2023741024
로봇학부 박건후

목차

(1) lifecycle node와 QOS 개념

lifecycle node은 개발자가 노드의 상태와 동작을 제어하여 원할 때만 정확한 값으로 통신할 수 있도록 하며 예측 가능한 동작을 하도록 만들어지 노드를 의미합니다. 라이프사이클 노드는 미리 정의된 상태를 갖춘 노드입니다. 그래서 일반 노드와 다르게 라이프사이클 노드에게는 상태라는 개념이 있고 그 상태를 전환하며 생성, 종료도 하기에 동작을 원할 때 어떤 동작을 하며 행동 방식 혹은 규칙을 정의한 것과 같습니다. lifecycle node의 상태는 Unconfigured, inactive, active, finalized로 4개로 나뉘어집니다. 각각 노드가 구성되지 않은 상태, 구성은 되었지만 publish 같은 주요 기능은 못하는 상태, active는 주요 기능이 가능한 상태, 노드가 종료되고 메모리 정리가 들어간 상태를 의미합니다. 그리고 이를 configure, activate, deactivate, cleanup, shutdown 같은 메서드로 이를 전환할 수 있게 됩니다. 이와 같은 노드 시스템으로는 예측 가능한 동작을 구현할 수 있게 되고 여러 가능한 활용 혹은 접목이 가능한데, 그 중 QOS에 접목하면 통신을 훨씬 고차원스럽게 구현할 수 있게 됩니다.

QOS는 DDS 통신을 할 때 통신 품질을 어떻게 보장할지를 관리하는 것입니다. 즉 메시지를 언제까지 보관할지, 어떻게 보낼지 네트워크 상태가 나쁠 때 어떻게 처리할지 등을 설정하게 해줍니다. QOS에는 Reliability, Durability, History, Depth, Deadline 등이 있습니다. Reliability에서는 RELIABLE, BEST_EFFORT 등으로 메시지를 다 받게 하거나, 혹은 메시지가 빠짐없이 다 전달되도록 하는 것, 혹은 데이터 손실이 있더라도 빠른 속도로 보내는 등의 설정을 할 수 있습니다.

Durability은 받은 데이터를 얼마만큼 저장하고 다시 볼 수 있는가에 대한 설정입니다. VOLATILE, TRANSIENT_LOCAL 2개로 설정하게 되는데 디폴트가 VOLATILE이고, TRANSIENT_LOCAL을 하면 나중에도 해당 메시지를 볼 수 있게 됩니다.

QoS policies	Publisher	Subscription	Compatible
reliability	Best effort	Best effort	Yes
reliability	Best effort	Reliable	No
reliability	Reliable	Best effort	Yes
reliability	Reliable	Reliable	Yes
durability	Volatile	Volatile	Yes
durability	Volatile	Transient local	No
durability	Transient local	Volatile	Yes
durability	Transient local	Transient local	Yes
deadline	Default	Default	Yes
deadline	Default	x	No
deadline	x	Default	Yes
deadline	x	x	Yes

위의 사진은 publish하는 쪽과 subscribe하는 쪽 간의 QOS설정 조합으로 인한 결과를 나타낸 것입니다. compatible이 No하는 것은 해당 경우에서는 호환이 안 되어 에러가 나게 되니 이를 유의하여 사용해야 합니다.

(2) publish 쪽 설명

클래스 이름을 pub_cycle이라고 짓고 LifecycleNode를 상속받았습니다. LifecycleNode의 인터페이스를 사용하기 위함입니다. publish 쪽 노드에서는 rclcpp의 TimerBase를 사용하여 해당 노드가 configure 상태인 경우에 1초 간격으로 publish를 하고자 했습니다. 이를 timer_callback이라는 private메서드를 통해 구현한 것입니다. 그 외에는 on_configure, on_activate등의 LifecycleNode의 메서드를 오버라이딩하며, LifecycleNode의 메서드들을 오버라이딩한 메서드 내에서 내부적으로 호출하고 노드의 현 상황을 터미널에 출력하는 명령어를 작성했습니다.

on_configure:

on_configure는 configure가 되면 이 메서드가 호출되며 멤버변수인 LifecyclePublisher 객체의 shared_ptr을 생성하여 메모리, 객체를 생성하게 됩니다. 이 때 QOS객체를 생성하고 reliablility와 durability를 각각 RELIABLE, TRANSIENT_LOCAL등으로 설정하는 것을 진행하고 create_publisher메서드를 통해 shared_ptr을 생성할 때 이 설정한 QOS 객체를 넣어서 publish쪽의 QOS설정을 해주었습니다. 이렇게 할 수 있는 이유가 create_publisher의 두 번째 매개변수가 const rclcpp::QoS& 밴로 선언되어 있는 헬퍼함수이기 때문입니다. 전에 말씀드렸듯, 타이머를 사용하기 위한 rclcpp::TimerBase::SharedPtr의 timer_멤버변수도 이 때 초기화를 합니다. 이 때 1초가 지

날 때마다 호출될 메서드로 timer_callback이라는 메서드를 바인딩하여 1초마다 메서드가 호출되도록 했습니다. lifecycle 객체가 생성되었으므로 이를 통해 publish가 가능해졌기 때문입니다. 그리고 해당 터미널 정보를 출력합니다. 오버라이딩을 하였기에 lifecycle 노드의 메서드를 모두 똑같이 작성해줘야 합니다. 그렇기에 메서드의 매개변수를 const rclcpp_lifecycle::State& 로 작성해줘야 했고 타입 또한 rclcpp_lifecycle::node_interfaces::LifecycleNodeInterface::CallbackReturn으로 반환해줘야 하기에 이에 대한 SUCCESS라는 열거형 멤버를 작성해줬습니다.

on _activate:

이 부분에서는 LifecyclePublisher 객체의 메서드인 on_activate, on_deactivate 메서드 중 on_activate라는 메서드를 내부적으로 호출하고 터미널 출력을 하게 했습니다.

on_deactivate:

이 부분에서는 on_deactivate메서드를 내부적으로 호출하며 관련 메시지를 터미널에 출력하게 했습니다.

on_cleanup:

이 경우는 configure 상태를 해제하는 것입니다. 그래서 LifecyclePublisher의 메모리를 해제시켜야 합니다. 그리고 configure가 해제되기에 publish를 할 수 없게 되고 그에 따라 timer_또한 관련 객체를 해제해줘야 합니다.

on_shutdown:

이 경우도 on_cleanup와 같은 기능을 가집니다. 하지만 이것을 오버라이딩한 이유는 configure-> activate 되었다면 deactivate를 하고 on_cleanup을 해야 하지만 이와 같은 과정을 없애고 바로 메모리를 즉시 해제하는 시스템 또한 lifecycle에서 정의해놓았기에 이를 구현하기 위해 만들어야 했습니다.

timer_callback():

이 함수는 강의자료 예제에 있던 1초마다 호출되도록 하는 부분을 참고하여 제작하였습니다. 1초마다 publish가 되어야 하기에 메시지를 구성하고 터미널 출력을 activate일 때와 configure일 때를 구분하여 출력하게 하고, publish메서드를 호출하도록 했습니다. 이 때 configure라면 LifecyclePublisher 객체 메서드의 publish 내에서 publish하는 것을 막기 때문에 메시지가 발행되지 않습니다.

(3) subscribe 쪽 설명

publish쪽과 같이 구현하는 방법이 유사합니다. 하지만 subscribe쪽에서는 LifecycleSubscriber 같은 특별한 클래스가 없기 때문에 일반 Subscription으로 작성하였고 라이프사이클 노드의 인터페이스를 사용해야 하기에 이를 상속하고 오버라이딩 하였습니다.

on_configure:

publish쪽 때와 유사하게 Qos 객체를 생성하여 둘 간의 통신이 호환이 되도록 RELIABLE, TRANSIENT_LOCAL로 설정하였고, Subscription 객체를 생성해줬습니다. 이 때 topic_callback이라는 메서드를 바인딩하였습니다.

topic_callback:

이 메서드에서는 m_isActiavted 플래그 멤버변수를 활용하여 on_actiavte메서드에서 이 플래그 멤버변수를 true로 하고 on_deactivate메서드를 호출하게 되면 false로 바꿈으로써 activate인 경우에만 터미널로 받은 메시지 내용을 출력하게 하여 사용자가 보기에는 active 상황이 아닐 때는 메시지를 안 받는 것처럼 보이게 구성하였습니다.

on_activate:

m_isActiavted 변수를 true로 하고 터미널에 active관련된 문자열을 출력했습니다.

on_deactivate:

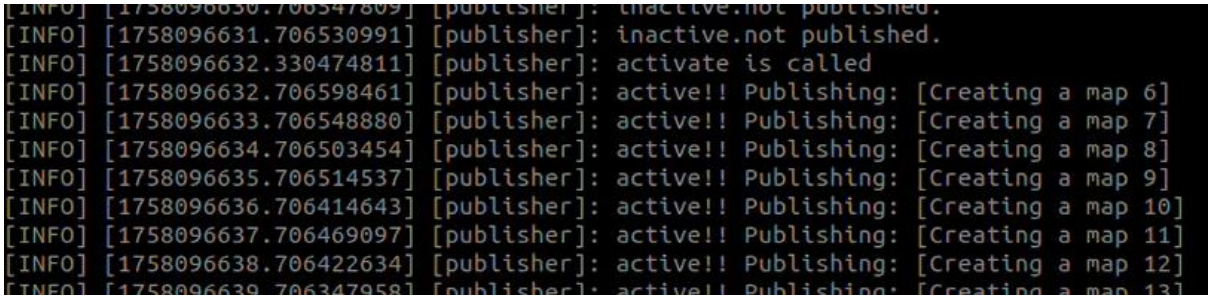
m_isActivated를 false로 하고 관련 문자열을 터미널에 출력합니다.

on_cleanup:

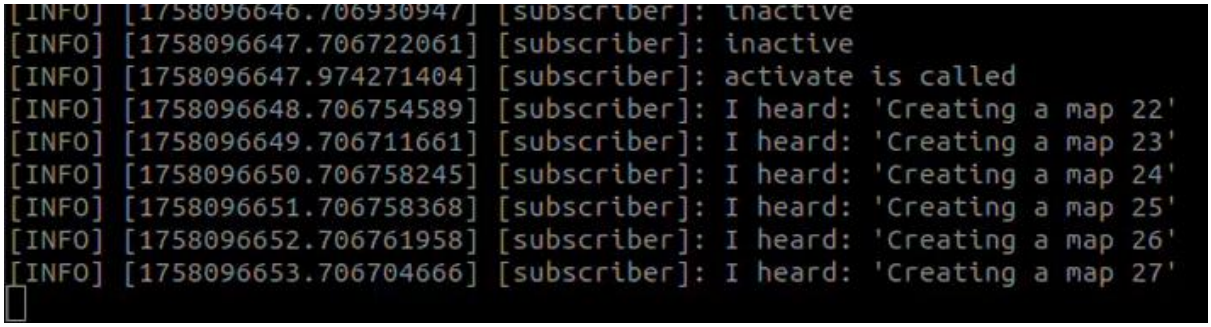
configure 상황이 아니므로 subscription 객체를 해제합니다.

on_shutdown:

on_cleanup과 같은 코드입니다.



실행 결과입니다. 현재 위의 사진은 publish되는 노드의 실행화면이며 inactive이었다가 active설정이 되어 메시지를 보내는 것과 함께 터미널에 보내는 메시지를 출력하고 있는 것입니다.



subscriber도 inactive였다가 active되자마자 받은 메시지 값을 출력하고 있는 것을 볼 수 있습니다. 나머지 부분은 github의 실행 영상을 보시면 더 자세히 실행과정을 보실 수 있습니다. 감사합니다.