목차

QoS(Quality of Service)

Hw1

Ros2에서 UDP와 DDS 통신을 사용하면서 정보의 신뢰도를 설정할 수 있게 됐다.

TCP만큼 안정적이거나 UDP만큼 빠르거나 그 사이의 신뢰성을 가진 통신을 설정할 수 있다.

각 노드마다 QoS를 설정할 수 있다.

서로 다른 QoS를 사용하면 메시지 전달이 불가능할 수 있다.

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
deadline	x	y (where y > x)	Yes
deadline	x	y (where y < x)	No
liveliness	Automatic	Automatic	Yes
liveliness	Automatic	Manual by topic	No
liveliness	Manual by topic	Automatic	Yes
liveliness	Manual by topic	Manual by topic	Yes
lease duration	Default	Default	Yes
lease duration	Default	x	No
lease duration	x	Default	Yes
lease duration	x	x	Yes
lease duration	x	y (where y > x)	Yes
lease duration	x	y (where y < x)	No

QoS 호환성 표

QoS policies

- **History**
- **Depth**
- **Reliability**
- **Durabillity**
- **Deadline**
- **Lifespan**
- **Liveliness**
- **Lease Duration**

Hw1

LifeCycleNode

노드의 상태를 세밀히 조정할 수 있는 관리형 노드로 만든다.

상태와 전환을 정의하여 초기화, 실행, 종료 중에 동작을 세밀하게 제어할 수 있다.

장점

시스템 자원 관리: 리소스의 할당 및 해제를 제어하여 시스템 성능을 최적화하고 리소스 누수의 위험을 줄일 수 있다.

디버깅: 노드의 상태를 볼수 있어 디버깅 및 유지보수에 용의하다.

LifeCycleNode 상태 종류

Hw1

- Unconfigured

노드가 생성됐지만 구성이 되어있지 않은 상태

- Inactive

노드가 구성은 되었지만 동작하지 않는 상태

- Active

노드가 동작하고 있는 상태

- Finalized

노드가 종료되고 리소스가 정리된 상태

- Configured

노드가 Unconfigured에서 Inactive로 전환

- Activate

노드가 Inactive 상태에서 Active 상태로 전환

- Deactivate

노드가 Active 상태에서 Inactive 상태로 전환

- Cleanup

노드가 Inactive에서 active로 전환

- Shutdown

노드가 finalized된 상태

구현 보고서 publisher


```
class LifecycleTalker : public rclcpp_lifecycle::LifecycleNode
{
public:
  explicit LifecycleTalker(const std::string & node_name, bool intra_process_comms = false)
  : rclcpp_lifecycle::LifecycleNode(node_name,
      rclcpp::NodeOptions().use_intra_process_comms(intra_process_comms))
  {}
```

기존 publisher에서 구현했던 클래스에서 상속을
rclcpp_lifecycle::LifecycleNode로 받아 lifecyclenode를 생성할 수 있는 클래스를 만든다.

구현 보고서 publisher

Hw1

```
void
publish()
{
  static size_t count = 0;
  auto msg = std::make_unique<std_msgs::msg::String>();
  msg->data = "Lifecycle HelloWorld #" + std::to_string(++count);

  if (!pub_->is_activated()) {
    RCLCPP_INFO(
      get_logger(), "Lifecycle publisher is currently inactive. Messages are not published.");
  } else {
    RCLCPP_INFO(
      get_logger(), "Lifecycle publisher is active. Publishing: [%s]", msg->data.c_str());
  }
  pub_->publish(std::move(msg));
}
```

Publish라는 함수를 만들어 송신할 메시지를 만들고

생선된 노드의 상태가 activated가 아니면 메시지가 송신되지 않았다고 로그를 띄운다.

노드의 상태가 activated면 로그에 송신되는 메시지를 띄운다

메시지를 송신한다.

구현 보고서 publisher

Hw1

```
rclcpp_lifecycle::node_interfaces::LifecycleNodeInterface::CallbackReturn
on_configure(const rclcpp_lifecycle::State &)
{
  rclcpp::QoS qos_prrofile(10);
  pub_ = this->create_publisher<std_msgs::msg::String>("lifecycle_chatter", qos_prrofile);
  timer_ = this->create_wall_timer(
    1s, std::bind(&LifecycleTalker::publish, this));

  RCLCPP_INFO(get_logger(), "on_configure() is called.");

  return rclcpp_lifecycle::node_interfaces::LifecycleNodeInterface::CallbackReturn::SUCCESS;
}
```

On_configure 상태일때

QoS를 이용해서 메시지 버퍼를 10으로 설정한다.
Pub_라는 노드를 생성하고 토픽 명과 QoS버퍼를 입력한다.
Timer_를 통해 1초마다 publish 함수가 실행되도록 만든다.

해당 메시지를 로그에 남긴다.

Callback이 성공하면 inactivate로의 전환을 호출
아니면 노드는 unconfigure 상태가 되고
이 외는 오류처리 상태가 된다.

구현 보고서 publisher

Hw1

```
rclcpp_lifecycle::node_interfaces::LifecycleNodeInterface::CallbackReturn
on_activate(const rclcpp_lifecycle::State & state)
{
  LifecycleNode::on_activate(state);

  RCLCPP_INFO(get_logger(), "on_activate() is called.");
  std::this_thread::sleep_for(2s);
  return rclcpp_lifecycle::node_interfaces::LifecycleNodeInterface::CallbackReturn::SUCCESS;
}
```

On activate 상태에서

메서드 재정의(선택사항)

로그에 해당 글 올린다
활성화를 위해 2초간 쉰다

Callback이 성공으로 반환되면 노드가 활성화가 된다.
만약 실패하면 TRANSITION_CALLBACK_FAILURE를
반환하여 노드가 비활성화 되고 이 외는 오류 처리상태다.

구현 보고서 publisher

Hw1

```
rclcpp_lifecycle::node_interfaces::LifecycleNodeInterface::CallbackReturn
on_deactivate(const rclcpp_lifecycle::State & state)
{
  LifecycleNode::on_deactivate(state);

  RCLCPP_INFO(get_logger(), "on_deactivate() is called.");

  return rclcpp_lifecycle::node_interfaces::LifecycleNodeInterface::CallbackReturn::SUCCESS;
}
```

Publish라는 함수를 만들어 송신할 메시지를 만들고

메서드 재정의(선택사항)

해당 메시지를 로그에 남긴다.

위에 처럼 성공시 inactivate 전환 호출,
실패시 active상태 유지,
아니면 오류처리로 상태 전환한다.

구현 보고서 publisher

Hw1

```
on_cleanup(const rclcpp_lifecycle::State &)
{
  timer_.reset();
  pub_.reset();

  RCUTILS_LOG_INFO_NAMED(get_name(), "on cleanup is called.");
  return rclcpp_lifecycle::node_interfaces::LifecycleNodeInterface::CallbackReturn::SUCCESS;
}
```

On_cleanup 상태에서는

Timer_ , pub_ 모두 할당 해제, 더 이상 사용 못한다.

메시지 로그에 올린다.
성공시 unconfigured 호출, 아니면 inactive,
이외는 errorprocessing으로 처리된다.

구현 보고서 publisher

Hw1

```
on_shutdown(const rclcpp_lifecycle::State & state)
{

  timer_.reset();
  pub_.reset();

  RCUTILS_LOG_INFO_NAMED(
    get_name(),
    "on shutdown is called from state %s.",
    state.label().c_str());
  return rclcpp_lifecycle::node_interfaces::LifecycleNodeInterface::CallbackReturn::SUCCESS;
}
```

On_sshutdown에서는

On_clear와 똑같다.

하지만 return에서 On_clear와 달리
성공시finalized가 된다.
실패시 현 상태를 유지하고
이외는 errorprocessing으로 처리된다.

구현 보고서 publisher

Hw1

```
private:
  std::shared_ptr<rclcpp_lifecycle::LifecyclePublisher<std_msgs::msg::String>> pub_;

  std::shared_ptr<rclcpp::TimerBase> timer_;
};

int main(int argc, char * argv[])
{
  setvbuf(stdout, NULL, _IONBF, BUFSIZ);
  rclcpp::init(argc, argv);
  rclcpp::executors::SingleThreadedExecutor exe;

  std::shared_ptr<LifecycleTalker> lc_node =
    std::make_shared<LifecycleTalker>("lc_talker");

  exe.add_node(lc_node->get_node_base_interface());
  exe.spin();
  rclcpp::shutdown();
  return 0;
}
```

클래스의 Private에서는

변수 선언이 이루어진다.

Stdout 버퍼를 플래쉬 하고(모든 출력에 대한 동기화 보장)
기존 publisher에 있던 선언을 해준다.
Lifecycle용으로 exe라는 객체를 생성한다.

기존 publisher와 유사하게 구성한다.

구현 보고서 publisher

Hw1

```
private:
  std::shared_ptr<rclcpp_lifecycle::LifecyclePublisher<std_msgs::msg::String>> pub_;

  std::shared_ptr<rclcpp::TimerBase> timer_;
};

int main(int argc, char * argv[])
{
  setvbuf(stdout, NULL, _IONBF, BUFSIZ);
  rclcpp::init(argc, argv);
  rclcpp::executors::SingleThreadedExecutor exe;

  std::shared_ptr<LifecycleTalker> lc_node =
    std::make_shared<LifecycleTalker>("lc_talker");

  exe.add_node(lc_node->get_node_base_interface());
  exe.spin();
  rclcpp::shutdown();
  return 0;
}
```

클래스의 Private에서는

변수 선언이 이루어진다.

Stdout 버퍼를 플래쉬 하고(모든 출력에 대한 동기화 보장)
기존 publisher에 있던 선언을 해준다.
Lifecycle용으로 exe라는 객체를 생성한다.

기존 publisher와 유사하게 구성한다.

구현 보고서 subscriber


```cpp
class LifecycleListener : public rclcpp::Node
{
public:
  explicit LifecycleListener(const std::string & node_name)
  : Node(node_name)
  {
    rclcpp::QoS qos_profile(10);
    sub_data_ = this->create_subscription<std_msgs::msg::String>(
      "lifecycle_chatter", qos_profile,
      std::bind(&LifecycleListener::data_callback, this, std::placeholders::_1));


    sub_notification_ = this->create_subscription<lifecycle_msgs::msg::TransitionEvent>(
      "/lc_talker/transition_event",qos_profile,
      std::bind(&LifecycleListener::notification_callback, this, std::placeholders::_1));
  }

  void data_callback(std_msgs::msg::String::ConstSharedPtr msg)
  {
    RCLCPP_INFO(get_logger(), "data_callback: %s", msg->data.c_str());
  }

  void notification_callback(lifecycle_msgs::msg::TransitionEvent::ConstSharedPtr msg)
  {
    RCLCPP_INFO(
      get_logger(), "notify callback: Transition from state %s to %s",
      msg->start_state.label.c_str(), msg->goal_state.label.c_str());
  }
```

Subscriber는 일반 노드로 설정했다.(예제가 그러했다)

데이터 수신부 노드 설정이다.(sub_data_)
버퍼를 QoS를 이용해 10만큼 설정을 해준다.
이외는 기존 subscriber와 똑같이 선언을 해준다.

Publisher의 상태 변환을 받는 TransitionEvent를 받는 노드를 만든다.

데이터를 받아 로그에 띄우는 함수(sub_data_용)

Publisher의 상태 변화를 입력 받아 로그에 띄어준다.(sub_notification용)

구현 보고서 subscriber

Hw1

```
private:
  std::shared_ptr<rclcpp::Subscription<std_msgs::msg::String>> sub_data_;
  std::shared_ptr<rclcpp::Subscription<lifecycle_msgs::msg::TransitionEvent>>
  sub_notification_;
};

int main(int argc, char ** argv)
{
  // force flush of the stdout buffer.
  // this ensures a correct sync of all prints
  // even when executed simultaneously within the launch file.
  setvbuf(stdout, NULL, _IONBF, BUFSIZ);

  rclcpp::init(argc, argv);

  auto lc_listener = std::make_shared<LifecycleListener>("lc_listener");
  rclcpp::spin(lc_listener);

  rclcpp::shutdown();

  return 0;
}
```

Private에서
노드 선언을 한다.

이 외는 기존 subscriber의 main과 유사하다.

Stdout 버퍼를 플래쉬 하고(모든 출력에 대한 동기화 보장)

참고 자료

Hw1

https://yhoons.tistory.com/109#google_vignette

https://github.com/ros2/demos/tree/rolling/lifecycle

https://velog.io/@i_robo_u/%EB%A7%88-ROS2-Node-%EA%BB%90%EB%8B%A4-%EC%BC%B0%EB%8B%A4-%ED%95%A0%EC%88%98-%EC%9E%88%EB%82%98-Lifecycle-Node%EA%B5%AC%ED%98%84%ED%95%98%EA%B8%B0C

https://docs.ros.org/en/rolling/Concepts/Intermediate/About-Quality-of-Service-Settings.html

https://soohwan-justin.tistory.com/96