- 토픽 명칭은 전송하고자 하는 데이터의 의미를 표현해야 한다. 토픽 명칭은 데이터 타입이나 데이터 구조를 나타내는 명칭이 아니라 전송하고자 하는 데이터의 의미를 나타내는 것이 좋다. 토픽을 데이터 타입마다 설정할 필요는 없다. 예를 들어, 아래와 같은 데이터 타입에 대해서 하나의 토픽을 생성할 수도 있으나 가능하면 공통적인 부분을 데이터 타입으로 정의하고 다양한 토픽에서 재사용할 수 있도록 설계하여야 한다.

|  |
| --- |
| struct VideoData  {     CodecKind videoCodec;     ResolutionType resolution;     sequence<octet, 2048> video;  }; |

위와 같은 데이터 타입은 "SecurityCamera", "UAVFeed", "RadarImage", "HighwayTrafficCam" 같은 토픽에서 재사용될 수 있다.

- 토픽 정의는 공유 가능한 하나의 장소에서 하라. 토픽은 소스 코드를 통하여 생성됨으로 개발자는 소스코드에 토픽 명칭을 직접 사용하기를 원할지도 모른다. 하지만 이런 경우에 토픽의 개수가 많아지거나 토픽을 공유하는 개발자가 늘어나는 경우에 오류를 일으킬 가능성이 매우 많으며 디버깅에 많은 시간을 보낼 수 있을 뿐만아니라 유지보수 또한 어렵다. 예를 들어 아래와 같이 소스에 직접 토픽 명칭을 사용하는 것은 좋은 방법은 아니다.

|  |
| --- |
| topic = participant->create\_topic("AirTrack", type\_name, …); |

이와 같은 소스는 아래와 같이 변경되는 것이 좋다.

|  |
| --- |
| topic = participant->create\_topic(AIR\_TRACK\_TOPIC, type\_name, …); |

위의 AIR\_TRACK\_TOPIC은 프로젝트에서 버전 관리되고 개발자끼리 공유 가능한 configuration 파일 등에 정의하여 사용하는 것이 좋다.

더 좋은 대안은 아래와 같이 토픽을 정의하는 IDL 파일을 사용하는 것이다.

|  |
| --- |
| IDL file  const string AIR\_TRACK\_TOPIC = "AirTrack"; |

- 가능한 적은 수의 Publisher와 Subscriber를 생성하라. DDS Publisher와 Subscriber는 데이터 전송과 수신을 책임지는 오브젝트이다. Publisher는 한 개 이상의 DataWriter를 가지며, Subscriber 또한 한 개 이상의 DataReader를 갖는다. 대부분의 어플리케이션은 하나의 DomainParticipant와 하나의 Publisher와 하나의 Subscriber를 가지면 충분하다.

하지만 간혹 복수개의 Publisher와 Subscriber를 가지는 경우가 있으며, 그런 경우는 다음과 같다.

* 서로 상이한 QoS를 가져야 할 경우. 당연한 이야기지만 서로 다른 QoS를 가져야 할 경우 반듯이 여러 개의 Publisher 또는 Subscriber를 생성해야 한다.
* 동시성을 높이기 위한 경우. 하나의 Subscriber에 속하는 DataReader는 샘플을 동시에 수신할 수 없기 때문에 동일한 QoS를 갖더라도 여러 개의 Subscriber를 생성하여 샘플 수신을 동시에 할 경우가 있다. 하나의 Publisher에 속한 DataWriter또한 동시에 사용할 수 없다. 비록 어플리케이션에서 서로 다른 쓰레드에서 서로 다른 DataWriter를 이용하여 데이터 전송을 시도하지만 이런 전송 오퍼레이션은 내부적으로 직렬화되어 동시성이 사라진다.

- DDS 엔티티의 생성과 삭제는 write() 또는 read() 루프 밖에서 이루어져야 한다. DDS 엔티티를 생성하거나 삭제할 경우에 DDS 미들웨어 내부에서 이루어지는 많은 일중의 하나는 네트워크에 존재하는 DDS 어플리케이션에 DDS 엔티티에 대한 다양한 정보를 주고 받는 discovery 과정이다. 이 과정 자체가 write() 또는 read() 오퍼레이션보다 시간을 더 소모하기 때문에 write() 또는 read() 루프에서 DDS 엔티티를 생성하는 경우 데이터는 거의 송수신되지 않게 된다.

- 동일한 토픽을 주고 받는 DataWriter와 DataReader 사이의 QoS는 서로 호환성이 있어야 한다. 예를 들어, DataWriter의 QoS는 RELIABLE 이나 DataReader의 QoS가 BEST\_EFFORT와 같이 서로 호환되지 않는 경우에는 통신 채널이 성립되지 않으며, 따라서 데이터의 송수신이 불가능하다.

- 일반적으로 데이터의 특징에 따른 전형적인 QoS의 분류는 아래와 같다.

|  |  |  |
| --- | --- | --- |
| 분류 | 특징 | QoS |
| 계속 발생하는 데이터 | 끊임없이 업데이트되는 데이터 | Best-effort |
| N 대 N 송수신 | Keys, multicast |
| 마지막 데이터가 의미 있는 센서 데이터 | Keep-last |
| 장애 발생 시에도 송수신되어야 하는 데이터 | Ownership, deadline |
| 상태 정보 | 간혹 변경되는 데이터 | Durability |
| 가장 최근의 데이터만 의미 있는 경우 | History |
| 알람 또는 이벤트 정보 | 비동기적인 메시지 | Liveliness |
| 송수신 확인이 필요한 경우 | Reliability |