Netty

2023.01.30 이지유

Netty란?

유지 관리가 용이한 **고성능** 프로토콜 서버와 클라이언트를 신속하게 개발할 수 있도록 하는
비동기 이벤트 기반 네트워크 프레임워크이다.

Netty를 왜 쓰는가?

네티는 적절한 **추상화** 모델을 제공하기 때문에 간단한 코드 작성만으로 안정적이고 빠른 네트워트 애플리케이션을 개발할 수 있게 도와준다.

Netty의 주요 키워드

- 비동기
- 이벤트 기반
- 고성능
- 추상화

동기 vs 비동기

동기식 호출

특정 서비스를 호출하면 완료될 때까지 그 응답을 기다리는 방식
-> 함수 결과를 호출한 쪽에서 처리하므로 쉬운 디버깅과, 직관적인 흐름 추적이 가능하다.
-> 서비스가 종료될 때까지 호출자가 마냥 기다려야 하므로 컴퓨팅 자원을 비효율적으로 사용

비동기식 호출

서비스를 호출한 후 즉시 응답을 받고, 다른 작업을 하다가 처리가 완료되었는지 확인하여 결과를 받는 방식
-> 함수의 결과를 호출한 쪽에서 처리하지 않는다.
-> 요청 결과를 기다리는 시간에 다른 작업을 수행할 수 있으므로 효율적으로 자원을 사용

=> 동기, 비동기는 함수 또는 서비스의 호출 방식을 뜻함.

블로킹 vs 논블로킹

블로킹

요청한 작업이 성공하거나 에러가 발생하기 전까지는 응답을 돌려주지 않는다.
=> 호출된 메소드의 작업이 끝날때 까지 호출한 메소드에게 제어권을 넘겨주지 않고 대기하게 만든다면 블로킹이다.

논블로킹

요청한 작업의 성공 여부와 상관없이 바로 결과를 돌려준다.
=> 어떤 메소드 실행 과정에서 다른 메소드를 호출한다고 해도, 호출된 메소드의 완료 여부와 상관 없이 바로 제어권을 다시 넘겨받아 호출한 메소드가 계속해서 다른 작업을 처리할 수 있도록 한다.

그래서
동기 / 비동기와 블로킹 / 논블로킹이
다른 점은 뭘까?

동기와 비동기는 호출되는 함수의 작업 완료 여부를 누가 신경쓰냐
=> 함수의 작업 완료를 호출한 함수가 신경을 쓰면 **동기**, 함수의 작업 완료를 호출된 함수가 신경쓰면 **비동기**

블로킹과 논블로킹은 호출되는 함수가 바로 리턴하느냐 마느냐를 관심사로 가진다.
=> 호출된 함수가 바로 리턴하지 않고, 제어권을 가지고 있으면서 해당 함수가 종료될 때 제어권을 넘긴다면 **블로킹**, 호출된 함수가 완료되지 않아도 제어권을 바로 호출한 함수에게 넘긴다면 **논블로킹**

이벤트 기반 프로그래밍

프로그래밍 패러다임 중 하나로 프로그램의 흐름이 특정 이벤트에 따라 결정되는 것을 말한다.
이벤트 기반 프로그래밍은 전통적으로 사용자 인터페이스가 포함된 프로그램에 많이 사용된다.
예를 들어 마우스 클릭에 반응하는 코드가 이에 해당된다. 이와 같이 이벤트 기반 프로그래밍은 각 이벤트를 먼저 정의해두고 발생한 이벤트에 따라서 코드가 실행되도록 프로그램을 작성한다.

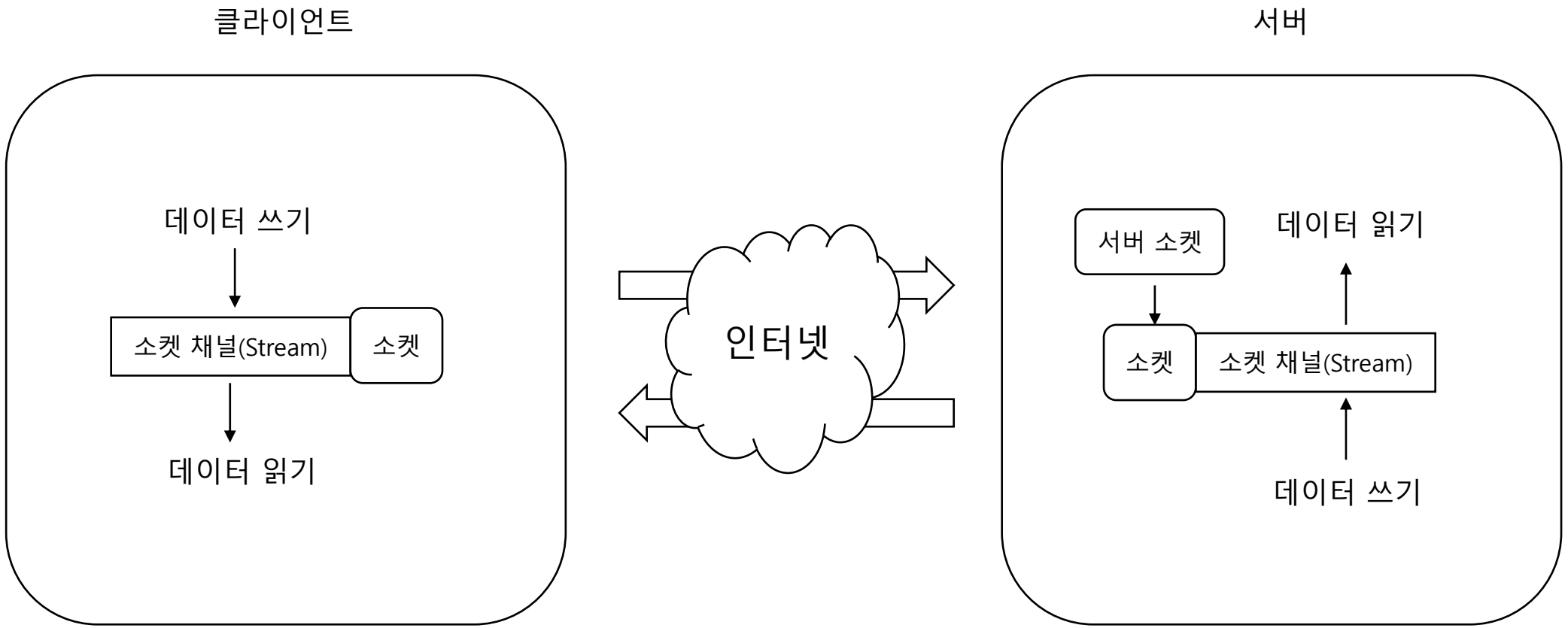
추상화 수준

이벤트를 나눌 때 작은 단위로 나누었는지 큰 단위로 나누었는지의 정도를 말한다.
이벤트의 추상화가 너무 고수준이면 세부적인 제어가 힘들어지고, 반대로 매우 저수준으로 추상화하면 한 동작에 대해서 너무 많은 이벤트가 발생하여 애플리케이션 성능에 악영향을 미치게 된다.
=> 서버에 연결될 클라이언트의 수는 매우 가변적이며 예측 불가능하므로 이벤트 기반 프레임워크의 적절한 추상화 단위는 매우 중요하다.

이벤트 기반 네트워크 프로그래밍

네티를 사용하지 않은 네트워크 프로그램에서 이벤트 발생 주체는 '소켓'이며, 이벤트 종류는 소켓연결, 데이터 송수신이다.
소켓에 데이터를 기록하고 읽으려면 소켓 채널(NIO) 또는 스트림을 사용한다.
클라이언트 애플리케이션이 소켓에 연결된 스트림에 데이터를 기록하면 소켓이 해당 데이터를 인터넷으로 연결된 서버로 전송한다. 이때 소켓에 문제가 발생하거나 연결된 스트림에 문제가 발생하면 새로운 소켓을 생성하고 데이터를 전송하는 예외 처리 코드가 작동되어야 할 것이다. 언제 어떠한 문제가 발생할 지 모르기 때문에 수신 대기 상황에서의 오류, 연결 시도 중 오류 등 다양한 상황에 대한 예외처리를 만들어야 한다. 이는 코드의 중복을 만들게 된다.

<네트워크 프로그램 구조>


클라이언트
데이터 쓰기
소켓 채널(Stream)
소켓
데이터 읽기
인터넷
서버
서버 소켓
데이터 읽기
소켓
소켓 채널(Stream)
데이터 쓰기

이벤트 기반 네트워크 프로그래밍

네티를 사용한다면 데이터를 소켓으로 전송하기 위해서 채널에 직접 기록하는 것이 아니라 데이터 핸들러(=이벤트 핸들러)를 통해서 기록하기 때문에 이벤트에 따라서 로직을 분리하는 장점과 코드를 재사용할 수 있다는 장점이 생긴다. 또한 네티의 이벤트 핸들러로 에러 이벤트도 정의할 수 있다.

<네티를 사용한 네트워크 프로그램 구조>