5957 객체지향 프로그래밍

60222117 이서현

목차

주제 : Socket Programming

1. Socket Programming

소켓 프로그래밍이란, 표준화된 네트워크의 부재로 생성되었다. 통신을 통해 Server와 Client가 각각 독립된 context에 있더라도 규칙에 맞게 데이터를 전송하고 송신할 수 있도록 하기 위해 소켓 프로그래밍이 등장하였다.

소켓은 OS에서 최소한의 통신을 지원하기 위해 생성했다. 통신에는 동기적 통신과 비동기적 통신으로 나뉘는데, 동기적 통신은 동 시간대에 이루어지는 통신이고, 비동기적 통신은 서로 다른 시간대에 이루어지는 통신이다. 소켓은 동기적 통신을 구현하기 위해 등장하였다. 동기적 통신의 경우, client와 Server가 모두 준비가 되어야 가능하다. 따라서 적어도, Server에서는 Client의 요청을 대기할 인력이 필요하다. 이를 Socket이 대체하는 것이다.

2. Client Socket과 Server Socket

Socket은 Server Socket과 Client Socket으로 나뉜다. 이는 기능적 차이가 아닌 용도의 차이로 나눈다. 같은 Socket이지만 용도에 따라 나누는 것뿐이지 구조적, 기능적 차이가 존재하지는 않다.

"Client Socket은 socket을 생성한 뒤, 서버 측에 연결을 요청한다. 서버 소켓에서 연결이 받아들여지면 데이터를 송수신하고, 모든 처

리가 완료되면 소켓을 닫는다. 즉, create, connect, send/recv, close 4 단계를 거친다."[참고문헌 1번- 인용]

"Server Socket은 Socket을 생성한 뒤, 서버가 사용할 IP주소와 Port 번호를 결합시키고, Client의 요청을 기다린다. Client의 요청이 수신되면, 이를 받아들여 데이터 통신을 위한 소켓을 생성한다. 새로운 소켓을 통해 연결이 수립되면, 데이터를 송수신하고 소켓을 닫는다."[참고문헌 1번- 인용] 다시 말해 처음에 생성한 Server Socket은 각 Client의 요청이 들어오면 이에 데이터를 주고받을 socket을 새로 생성하고 Client와 연결해주는 역할을 하는 것이지, 처음에 생성한 Server Socket이 직접 데이터를 다루지 않는다. 따라서 Server Socket은 create, bind, listen, accept, send/recv, close의 순서를 거친다.

3. Stub과 Skeleton, 직렬화와 역직렬화

위의 Socket만을 이용해 Server와 Client 사이의 통신을 이루기엔 한계점이 있다. Client에서 요청을 보낼 때에, Server에 어떤 data가 있고, 어떤 것을 요구할 수 있을지 모르기 때문에 Client는 올바른 요청을 보내기 어렵다. 이를 위해 나타난 개념이 stub이다. Stub은 Server에 있는 객체의 메소드의 존재를 추상적으로 알려주고자 인터페이스 형태로 추상화되어 Server를 보여주는 역할을 한다. 또한, 메소드의 인자를 일정한 포맷으로 바꿔주는 parameter marshalling 역할을 한다. 이는 모든 컴퓨터 시스템에서 데이터 표현방법의 차이 때문이다.

반대로 Client또한, Stub과 같이 대리인의 역할을 하는 Skeleton이 있다. Skeleton은 stub의 역할과 반대로, 조율된 인자들을 서버에 맞는 형태로 역 직렬화를 해준 뒤 알맞은 메소드를 실행시키고, 실행이 끝나면 다시 parameter marshalling을 통해 stub에게 전달한다. Stub은 이 결과를 받아 다시 시스템에 맞게 변환한 뒤 원래 메소드를 불렀던 객체에게 결과를 넘긴다.

참고문헌

1)"소켓 프로그래밍", tistory, https://recipes4dev.tistory.com/153 ,2.2 소켓흐름 2,3 단락 인용

2)"[Java RMI] Stub/Skeleton, RMI 통신 메커니즘" ,minkysoft, https://blog.naver.com/garaboss/100120711144