디지털 컨버전스 기반 UI/UX 프론트 개발자 양성과정

2021.01.19

Fifteenth

이 범 진
E-mail : bumttang01@naver.com
GitHub : https://github.com/Leebumjin01

복습

소켓(Socket) 이란?

자바 프로그램에서 소켓(Socket)이라는 개념을 통해서 네트워크 통신을 함.
나만의 통신구조를 설계하고 직접 내용을 일일이 전부 다 설계함.
서버 소켓 생성시 서비스 번호를 부여
서비스 번호로 port를 설정
네트워크 포트가 16비트를 사용하므로 제한은 0 ~ 65535개
단, 고정된 포트를 사용하면 안됨

소켓에서의 예외처리 (Exception)

UnknownHostException : 호스트를 찾을 수 없거나, 서버의 포트가 열려 있지 않은 경우
IOException : 네트워크 연결 실패, 서버에 접근 할 수 없는 경우

소켓을 이용한 입출력 스트림

Stream 은 한 방향으로만 통신할 수 있음, 입력과 출력을 동시에 처리할 수 없다.

InputStream 변수명 = socket.getInputStream();
outputStream 변수명 = socket.GetOutputStream();

SocketServerTest 클래스 입니다.

```java
public class SocketServerTest {
    public static void main(String[] args) throws IOException {
        // 문자열을 숫자로 바꾸는 기능
        int port = Integer.parseInt( s: "33333"); // 굳이 왜 바꾸는지?

        ServerSocket serverSocket = new ServerSocket( port: 33333);

        try {
            // 소켓이란 ? 네트워킹을 할 수 있는 클래스 객체
            // 서버 소켓 생성시 서비스 번호를 부여해야 하는데
            // 이 서비스 번호로 port(33333)을 설정했다. 법
            // 네트워크 포트가 16비트를 사용하므로 제한은 0 ~ 65535
            // (단, 고정된 포트를 사용하면 안됨)
            ServerSocket servSock = new ServerSocket(port);

            System.out.println("Server: Listening - " + port);

            while(true) {
                // accept()의 경우 클라이언트가 접속을 요청했는지 체크해서
                // 만약 요청 있었다면 요청을 승인한다.
                // (accept()는 블로킹 연산이다)

                // 결국 sock은 클라이언트 소켓을 의미하게 된다.
                // 그래서 좀 더 가독성이 좋은 코드는 Socket clntSock으로 작성하면 좋을
                // (전화왔을때 통화하기 슬라이드가 accept()라 보면됨)
                Socket sock = servSock.accept();

                // 접속한 클라이언트의 IP를 확인하는 코드
                System.out.println(
                        "[" + sock.getInetAddress() +
                                "] client connected"
                );
```

SocketServerTest 클래스 입니다.

```
OutputStream out = sock.getOutputStream();
// println의 결과를 out으로 전송한다라는 뜻
// 내가 서버에 요청하는 것: 출력이 아닌 입력이다.
// 서버가 처리해서 돌려주는 것: 이것이 출력이다.
// 그렇다면 현재 클래스의 OutputStream out = sock.getOutputStream(); 은
// Server(출력) -> Client(입력) 이 되는 건가요 ??
// 서버는 출력이 되고 클라이언트는 입력이 된다 ??

PrintWriter writer = new PrintWriter(out, autoFlush: true);
// autoFlush 는 무엇인가요 ??
// 즉 여기서 println의 출력은 클라이언트에게 간다.
// Date()는 시간을 가져온다.
// toString()은 문자열로 만듬

// 접속 상대에게 보내고 싶은 데이터를 이곳에 기록한다.
writer.println(new Date().toString());

// 입력: 클라이언트
// 클라이언트가 서버에게 보낸 것
InputStream in = sock.getInputStream();
// 여기서의 InputStream in = sock.getInputStream(); 은
// Client(출력) -> Server(입력) 이 맞나요 ??
```

SocketClientTest 클래스 입니다.

```java
public class SocketClientTest {
    public static void main(String[] args) {
        // 내가 접속할 서버의 IP 주소를 적습니다.
        String hostname = "192.168.0.9";
        // 서버에 여러 서비스가 있을 수 있는데
        // 그 중에서 내가 사용하고자 하는 서비스의 포트 번호를 적습니다.
        int port = 33333;

        for(int i = 0; i < 10; i++) {   // for 문을 돌리는 이유를 모르겠습니다.
            try {                       // 무엇을 반복 하는 건가요 ??
                // Socket 객체를 할당해서
                // 서버의 IP, 포트 번호를 가지고 접속을 요청합니다.
                // 서버에 대한 소켓을 획득하게 됩니다.
                // 이 요청이 들어갈때 서버의 accept()가 동작하게 됩니다.
                // 예를 들자면 이 행위는 전화를 거는것과 같다.
                // (서버쪽 주석을 살펴보면 감이 더 잘 올 것이다)
                Socket sock = new Socket(hostname, port);

                // 서버의 출력을 획득
                // 즉 서버가 수신하게 만들도록 설정을 해주는 것
                OutputStream out = sock.getOutputStream();
                // Client(출력) -> Server(입력) ??

                String str = "Hello Network Programming";
                // 위의 문자열을 바이트 단위로 쪼개서 서버로 전송한다. getBytes
                out.write(str.getBytes());

                // 서버의 입력을 생성(수신 준비)
                InputStream in = sock.getInputStream();
                // Server(출력) -> Client(입력) ??
                BufferedReader reader =
                        new BufferedReader(new InputStreamReader(in));
```

SocketClientTest 클래스 입니다.

```
} catch (UnknownHostException e) {
    System.out.println("Server Not Found: " + e.getMessage());

} catch (IOException e) {
    System.out.println("I/O Error: " + e.getMessage());
    // try ~ catch 문 ...?
    // 구글링 해보니 Exception 은 개발자가 구현한 로직에서 발생하고,
    // *발생할 상황을 미리 예측*하여 처리할 수 있다 라는것 같은데
    // Socket 에서의 Exception 은 UnknownHostException 과 IOException
    // 이 두개 밖에 없기 때문에 try ~ catch 문을 쓴건가요?
    // 다른 로직에서는 다른 예외처리가 있을거 같은데
    // 개발자가 예외처리 케이스를 다 파악하고 일일이 다 해주어야 하나요 ??
```