

## 22. 네트워크 프로그래밍

### 소켓 통신 (Socket, ServerSocket)

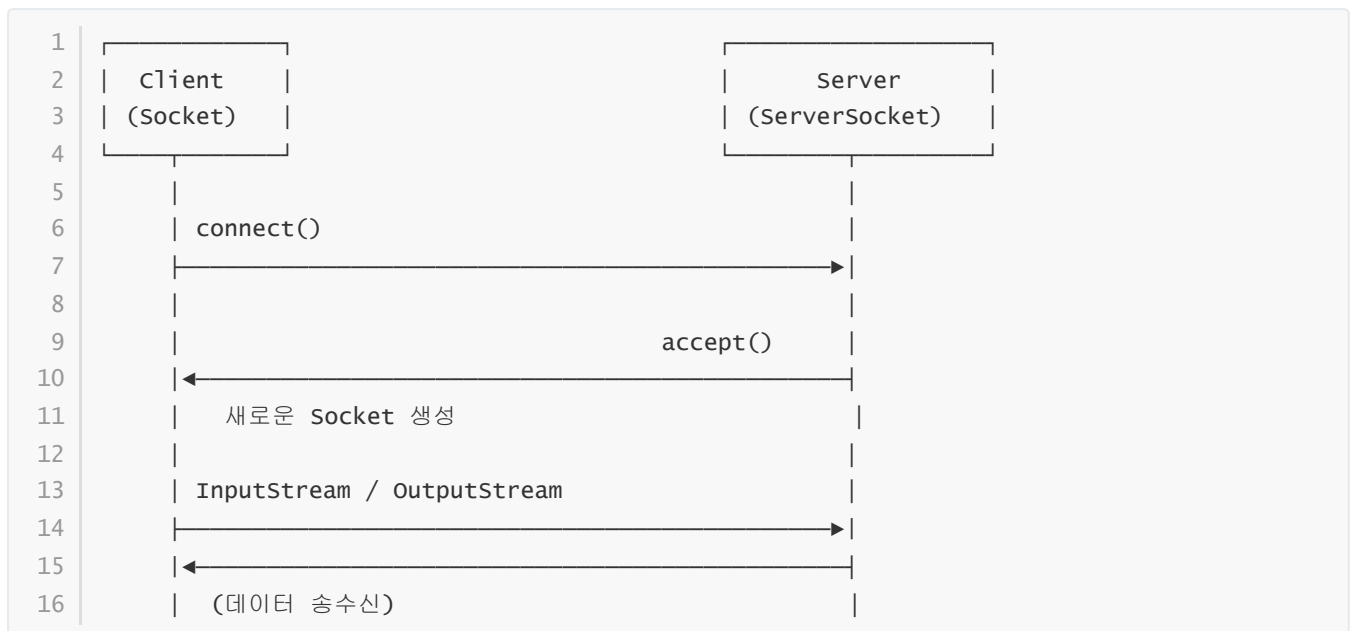
#### 1 소켓 통신이란?

소켓(Socket)은 두 장치 간 통신을 가능하게 하는 **종단점(endpoint)** 개념이다.

Java에선 클라이언트와 서버가 각각 `Socket` 과 `ServerSocket` 클래스를 사용해서 통신한다.

- **서버(Server):** `ServerSocket` 객체 생성 → 클라이언트 접속을 수신하고 `Socket` 반환
- **클라이언트(Client):** `Socket` 객체로 서버에 연결 요청
- 양쪽 모두 입출력 스트림을 통해 데이터 송수신

#### 2 구조 요약



#### 설명 요약

- **Client**는 `Socket` 을 생성하고 `connect()` 를 호출해 서버로 접속 시도
- **Server**는 `serverSocket.accept()` 를 통해 클라이언트 요청을 기다림
- 연결이 수락되면 **서버 측 Socket**이 생성됨
- 그 후, 양쪽에서 `InputStream / OutputStream` 을 통해 데이터를 주고받음

### 3 간단한 코드 예제

#### Server (EchoServer.java)

```
1  import java.io.*;
2  import java.net.*;
3
4  public class EchoServer {
5      public static void main(String[] args) throws IOException {
6          ServerSocket serverSocket = new ServerSocket(12345); // 12345 포트 열기
7          System.out.println("서버 대기 중...");
8
9          Socket clientSocket = serverSocket.accept(); // 클라이언트 접속 대기
10         System.out.println("클라이언트 연결됨: " + clientSocket.getInetAddress());
11
12         BufferedReader in = new BufferedReader(new
13         InputStreamReader(clientSocket.getInputStream()));
14         PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
15
16         String inputLine;
17         while ((inputLine = in.readLine()) != null) {
18             System.out.println("수신: " + inputLine);
19             out.println("Echo: " + inputLine);
20             if ("bye".equalsIgnoreCase(inputLine)) break;
21         }
22
23         in.close();
24         out.close();
25         clientSocket.close();
26         serverSocket.close();
27     }
28 }
```

#### Client (EchoClient.java)

```
1  import java.io.*;
2  import java.net.*;
3
4  public class EchoClient {
5      public static void main(String[] args) throws IOException {
6          Socket socket = new Socket("localhost", 12345); // 서버 접속
7          System.out.println("서버 연결 성공");
8
9          BufferedReader userInput = new BufferedReader(new
10         InputStreamReader(System.in));
11         BufferedReader in = new BufferedReader(new
12         InputStreamReader(socket.getInputStream()));
13         PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
14
15         String msg;
16         while ((msg = userInput.readLine()) != null) {
```

```

15         out.println(msg); // 서버로 전송
16         System.out.println(in.readLine()); // 서버 응답 출력
17         if ("bye".equalsIgnoreCase(msg)) break;
18     }
19
20     socket.close();
21 }
22 }

```

## 4 핵심 클래스 정리

클래스	설명
<code>ServerSocket</code>	서버 측 소켓. 포트 바인딩 후 <code>accept()</code> 로 클라이언트 연결 수락
<code>Socket</code>	클라이언트 또는 연결된 서버 측 소켓
<code>InputStream</code> , <code>OutputStream</code>	바이트 단위 전송 스트림
<code>BufferedReader</code> , <code>PrintWriter</code>	문자 기반 스트림 (라인 단위 처리에 유리)

## 5 포트와 IP

- 포트 번호는 1024 이상 (일반적으로 49152~65535 권장)
- IP는 로컬 테스트 시 `"localhost"` 또는 `"127.0.0.1"` 사용
- 실제 배포 시 서버 IP 지정 필요

## 6 주의사항

- 한 번에 하나의 `accept()` 만 처리하므로 **멀티스레드** 서버를 만들려면 `new Thread()` 로 각각 대응해야 해
- 입출력 스트림 닫지 않으면 연결 해제가 안 됨
- 클라이언트가 먼저 종료되면 `readLine()` 은 `null` 반환

## ✓ 실전 활용 시 확장

확장 기능	설명
 다중 클라이언트 처리	<code>while (true) { new Thread(...).start(); }</code> 구조로 처리
 SSL 소켓 통신	<code>SSLSocketFactory</code> , <code>SSLServerSocketFactory</code> 사용
 성능 최적화	<code>BufferedOutputStream</code> , <code>BufferedInputStream</code> 사용
 파일 전송	바이트 스트림 ( <code>DataInputStream</code> , <code>DataOutputStream</code> ) 활용

## 요약

항목	설명
<code>ServerSocket</code>	서버가 대기할 포트를 열고 클라이언트 연결 수락
<code>Socket</code>	클라이언트가 서버에 연결할 때 사용
통신 방법	스트림 기반 입출력 ( <code>InputStream</code> , <code>OutputStream</code> )
주요 기능	Echo 서버, 채팅, 파일 전송 등

## UDP 통신 (`DatagramSocket`, `DatagramPacket`)

### 1 UDP 개요

- **TCP**는 연결 기반(3-way handshake), **UDP**는 **비연결 기반**
- 데이터는 **Datagram(데이터그램)**이라는 단위로 전송됨
- 신뢰성, 순서 보장 없음 (손실 가능 있음)
- 빠르지만 재전송, 흐름 제어 없음

### 2 UDP 통신 클래스

클래스	설명
<code>DatagramSocket</code>	데이터를 송수신하는 소켓
<code>DatagramPacket</code>	전송할 데이터 및 수신할 데이터를 담는 객체

### 3 기본 구조

```
1 [Sender/Client]
2   └─ DatagramSocket.send(DatagramPacket)
3
4 [Receiver/Server]
5   └─ DatagramSocket.receive(DatagramPacket)
```

### 4 예제 코드

#### 서버 (UDPReceiver.java)

```
1 import java.net.DatagramPacket;
2 import java.net.DatagramSocket;
3
4 public class UDPReceiver {
5     public static void main(String[] args) throws Exception {
```

```

6      DatagramSocket socket = new DatagramSocket(9876); // 수신 포트 지정
7      byte[] buffer = new byte[1024]; // 수신 버퍼
8
9      System.out.println("서버 대기 중...");
10
11     while (true) {
12         DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
13         socket.receive(packet); // 수신 대기
14
15         String received = new String(packet.getData(), 0, packet.getLength());
16         System.out.println("수신된 메시지: " + received);
17
18         if (received.equalsIgnoreCase("exit")) break;
19     }
20
21     socket.close();
22 }
23 }

```

## 클라이언트 (UDPSender.java)

```




1  import java.net.DatagramPacket;
2  import java.net.DatagramSocket;
3  import java.net.InetAddress;
4  import java.util.Scanner;
5
6  public class UDPSender {
7      public static void main(String[] args) throws Exception {
8          DatagramSocket socket = new DatagramSocket();
9          InetAddress serverAddress = InetAddress.getByName("localhost");
10
11          Scanner scanner = new Scanner(System.in);
12          System.out.println("메시지를 입력하세요 (exit 입력 시 종료):");
13
14          while (true) {
15              String msg = scanner.nextLine();
16              byte[] data = msg.getBytes();
17
18              DatagramPacket packet = new DatagramPacket(data, data.length,
serverAddress, 9876);
19              socket.send(packet);
20
21              if (msg.equalsIgnoreCase("exit")) break;
22          }
23
24          socket.close();
25          scanner.close();
26      }
27  }

```

## 5 주요 특징 및 주의점

특징	설명
비연결성	<code>connect()</code> 과정 없이 송수신 가능
경량	헤더가 간단하고 빠름
손실 가능	보장되지 않음, 재전송 없음
브로드캐스트	같은 네트워크의 여러 장치에 전송 가능
순서 없음	순서를 보장하지 않음 (수신 순서 불일치 가능)

## 6 고급 확장 예시

기능	설명
 파일 조각 전송	UDP는 작은 패킷(일반적으로 512~1024바이트)으로 쪼개서 전송 필요
 브로드캐스트 통신	255.255.255.255 혹은 네트워크 대역 사용
 반복 재전송 로직	Ack + timeout 로직 직접 구현
 멀티스레드 수신	서버에서 여러 클라이언트의 메시지를 병렬로 처리

## 7 TCP vs UDP 비교표

항목	TCP	UDP
연결 여부	연결 지향	비연결 지향
신뢰성	높음 (패킷 보장)	낮음 (손실 가능)
속도	느림	빠름
용도	파일 전송, HTTP 등	게임, 스트리밍, 실시간 통신
프로토콜 사용 클래스	<code>Socket</code> , <code>ServerSocket</code>	<code>DatagramSocket</code> , <code>DatagramPacket</code>

## ✓ 요약

- `DatagramSocket` 은 UDP 송수신을 위한 소켓
- `DatagramPacket` 은 전송할 데이터나 수신할 데이터를 담는 컨테이너
- 빠르고 가볍지만 신뢰성 보장은 없음
- 주로 실시간 요구되는 서비스에 적합

# HTTP 클라이언트 (HttpURLConnection, HttpClient)

## 1 HttpURLConnection (레거시 API)

### 💡 개요

- JDK 1.1부터 제공된 표준 HTTP 통신 클래스
- GET, POST, PUT, DELETE 등 요청 가능
- 설정이 번거롭고 boilerplate 코드가 많음

### ✅ GET 요청 예제

```
1 import java.io.BufferedReader;
2 import java.io.InputStreamReader;
3 import java.net.HttpURLConnection;
4 import java.net.URL;
5
6 public class HttpGetExample {
7     public static void main(String[] args) throws Exception {
8         URL url = new URL("https://jsonplaceholder.typicode.com/posts/1");
9         HttpURLConnection conn = (HttpURLConnection) url.openConnection();
10
11         conn.setRequestMethod("GET");
12         conn.setRequestProperty("Accept", "application/json");
13
14         BufferedReader in = new BufferedReader(
15             new InputStreamReader(conn.getInputStream()));
16         String inputLine;
17         StringBuilder content = new StringBuilder();
18
19         while ((inputLine = in.readLine()) != null)
20             content.append(inputLine);
21
22         in.close();
23         conn.disconnect();
24
25         System.out.println(content.toString());
26     }
27 }
```

## 2 HttpClient (Java 11+)

### 💡 개요

- 비동기 지원 가능 (동기/비동기 모두 지원)
- Builder 패턴 도입, 사용성 향상
- `CompletableFuture` 및 `BodyHandler` 연동 가능

## ✓ GET 요청 예제 (동기)

```
1 import java.net.URI;
2 import java.net.http.HttpClient;
3 import java.net.http.HttpRequest;
4 import java.net.http.HttpResponse;
5
6 public class ModernHttpGet {
7     public static void main(String[] args) throws Exception {
8         HttpClient client = HttpClient.newHttpClient();
9         HttpRequest request = HttpRequest.newBuilder()
10             .uri(URI.create("https://jsonplaceholder.typicode.com/posts/1"))
11             .GET()
12             .build();
13
14         HttpResponse<String> response = client.send(request,
15             HttpResponse.BodyHandlers.ofString());
16
17         System.out.println("Status code: " + response.statusCode());
18         System.out.println("Body: " + response.body());
19     }
20 }
```

## ✓ POST 요청 예제 (JSON 전송)

```
1 import java.net.URI;
2 import java.net.http.HttpClient;
3 import java.net.http.HttpRequest;
4 import java.net.http.HttpResponse;
5 import java.net.http.HttpRequest.BodyPublishers;
6
7 public class ModernHttpPost {
8     public static void main(String[] args) throws Exception {
9         String json = "{\"title\":\"foo\", \"body\":\"bar\", \"userId\":1}";
10
11         HttpRequest request = HttpRequest.newBuilder()
12             .uri(URI.create("https://jsonplaceholder.typicode.com/posts"))
13             .header("Content-Type", "application/json")
14             .POST(BodyPublishers.ofString(json))
15             .build();
16
17         HttpClient client = HttpClient.newHttpClient();
18         HttpResponse<String> response = client.send(request,
19             HttpResponse.BodyHandlers.ofString());
20
21         System.out.println("Response code: " + response.statusCode());
22         System.out.println("Response body: " + response.body());
23     }
24 }
```



### 3 HttpURLConnection vs HttpClient 비교

항목	HttpURLConnection	HttpClient (Java 11+)
도입 시기	Java 1.1	Java 11
동기/비동기	동기만 지원	둘 다 지원 ( <code>sendAsync()</code> )
HTTP/2 지원	✗	✓
사용성	복잡하고 장황함	깔끔한 Builder 패턴
스트림 핸들링	직접 <code>InputStream</code> 처리	<code>BodyHandler</code> 로 자동화
사용 권장 여부	레거시 유지용	적극 권장 (신규 코드)

### 4 기타 응용

- ! 파일 다운로드: `HttpClient + BodyHandlers.ofFile(Path.of(...))`
- 🔑 인증: `Authorization` 헤더 ( `Bearer`, `Basic` )
- 🧪 테스트 서버: `https://httpbin.org/`, `https://jsonplaceholder.typicode.com`

### ✓ 요약

- `HttpURLConnection` 은 오래된 API이지만 여전히 사용 가능
- `HttpClient` 는 간결하고 현대적인 API로 대체 가능
- 실제 애플리케이션에서는 `HttpClient` 사용을 권장