# IO 기반 입출력 및 네트워킹

주효진

# CONTENTS

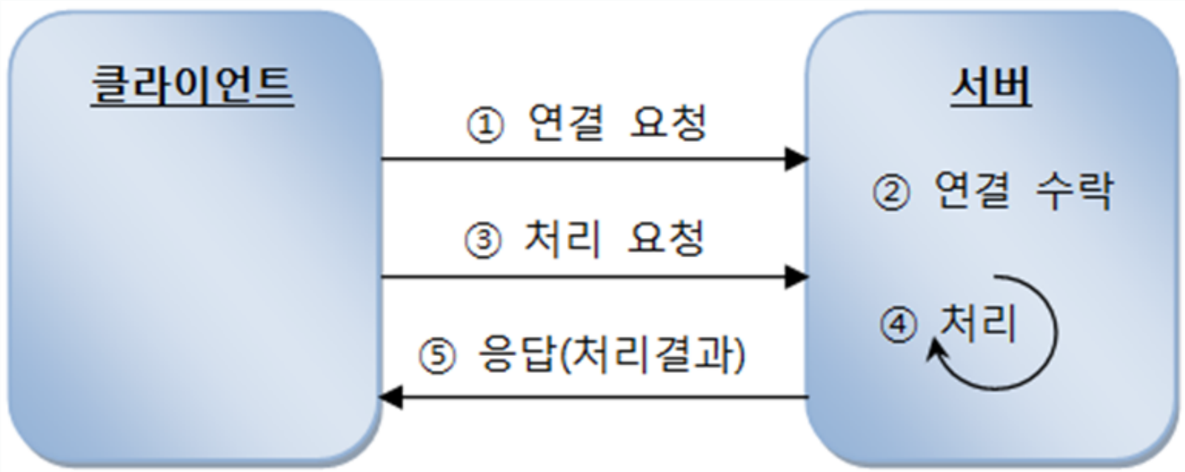
Chapter 06

# 네트워크 기초

# 서버와 클라이언트

- 네트워크는 여러 대의 컴퓨터를 통신 회선으로 연결한 것
  ex) 방마다 컴퓨터가 있고 이 컴퓨터를 통신회선으로 연결한 경우 > 홈네트워크
  회사, 건물, 특정 영역에 존재하는 컴퓨터를 통신회선으로 연결한 경우 > 지역 네트워크
  지역 네트워크를 통신회선으로 연결한 것 > 인터넷

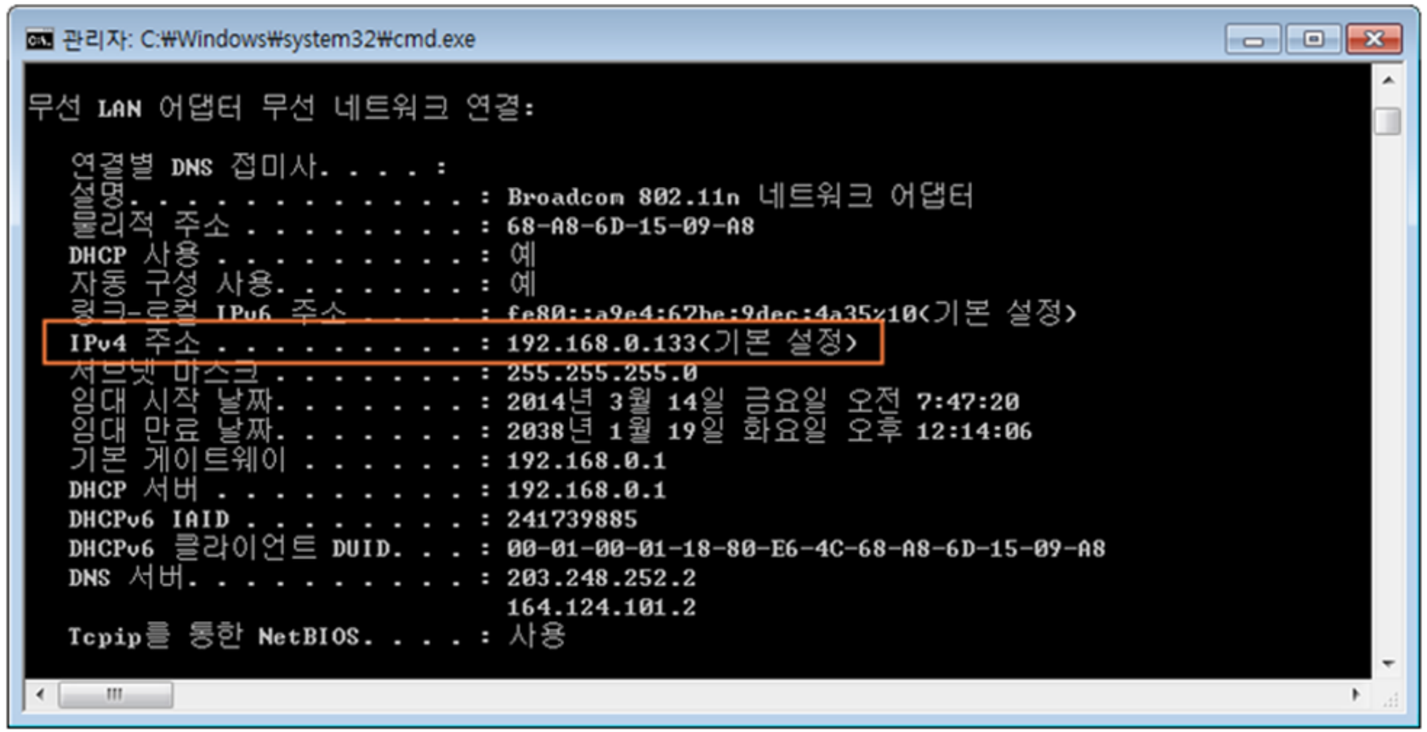
- 서비스를 제공하는 프로그램을 서버
  서비스를 받는 프로그램을 클라이언트
  두 프로그램이 통신하기 위해서는 연결을 요청하는 역할과 연결을 수락하는 역할이 필요
  서버는 클라이언트 가 요청(request)하는 내용을 처리해주고 응답(response)을 클라이언트에게 보냄



# IP 주소와 포트

- IP(Internet Protocol) : 컴퓨터의 고유한 주소
  IP는 네트워크 어댑터 마다 할당
  cmd에서 ipconfig /all을 치면 IP 주소를 확인 할 수 있음

C:₩>ipconfig /all



# IP 주소와 포트

- IP 주소는 xxx.xxx.xxx.xxx와 같은 형식으로 표현 (xxx는 0~255)
  상대의 IP 주소를 모르면 통신할 수 없기 때문에 DNS(Domain Name System)을 이용하여 연결할 컴퓨터의 IP 주소를 찾음

  [DNS]

  도메인 이름 [www.naver.com] : IP 주소 [222.122.195.5]

- Port : 같은 컴퓨터 내에 프로그램을 식별하는 번호
  클라이언트는 연결 요청 시 IP 주소와 Port를 같이 제공
  포트 번호의 전체 범위는 0~655350이고 다음 표로 구분

| 구분명 | 범위 | 설명 |
|---|---|---|
| Well Know Port Numbers | 0~1023 | 국제인터넷주소관리기구(ICANN)가 특정 애플리케이션용으로 미리 예약한 포트 |
| Registered Port Numbers | 1024~49151 | 회사에서 등록해서 사용할 수 있는 포트 |
| Dynamic Or Private Port Numbers | 49152~65535 | 운영체제가 부여하는 동적 포트 또는 개인적인 목적으로 사용할 수 있는 포트 |

# InetAddress로 IP 주소 얻기

- 자바는 java.net.InetAddress 객체로 IP 주소를 표현
  InetAddress는 로컬 컴퓨터의 IP 주소 및 DNS에 도메인 이름을 검색한 후 IP 주소를 가져오는 기능을 제공

  로컬 컴퓨터의 InetAddress를 얻고 싶을 경우 아래를 실행
  InetAddress ia = InetAddress.getLocalhost();

  외부 컴퓨터의 도메인 이름을 통해 IP 주소를 얻기 위한 메소드
  InetAddress ia = InetAddress.getByName("www.naver.com");
  InetAddress[] iaArr = InetAddress.getAllByName("www.naver.com");

  리턴받은 InetAddress 객체의 IP 주소를 얻기 위한 메소드
  String ip = ia.getHostAddress

Chapter 07

# TCP 네트워킹

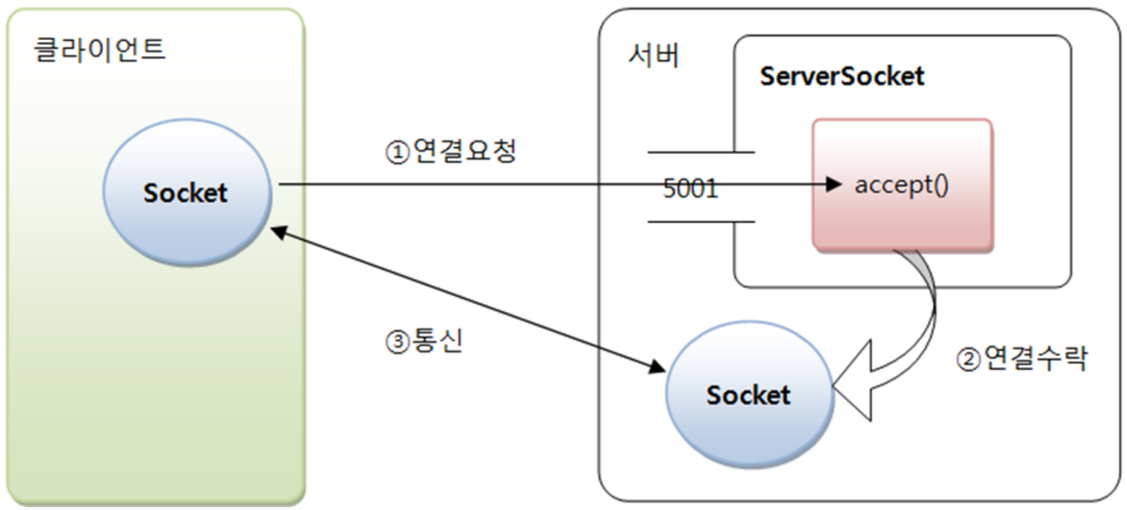
# TCP 네트워킹

- TCP(Transmission Control Protocol) : 연결 지향적 프로토콜.
  클라이언트와 서버가 연결된 상태에서 데이터를 주고받는 프로토콜
  클라이언트가 연결을 요청 - 서버가 연결을 수락 > 통신 선로 고정

  장점 : 통신 선로를 통해 순차적으로 데이터를 정확하고 안정적으로 전달
  단점 : 데이터를 보내기 전에 반드시 연결이 되어야 하고 고정된 통신 선로가 최단선이 아닐 경우 UDP 보다 데이터 전송 속도가 느릴 수 있음

# ServerSocket과 Socket의 용도

- TCP 네트워킹을 위해 java.net.ServerSocket과 java.net.Socket 클래스 제공
  TCP 서버의 역할
  1. 클라이언트가 연결 요청을 해오면 연결을 수락 -> ServerSocket
  2. 연결된 클라이언트와 통신 -> Socket




- 바인딩 포트 : 클라이언트가 서버에 접속할 수 있는 포트
  서버는 고정된 포트 번호에 바인딩 후 실행
  ServerSocket 생성 시 포트 번호를 지정해야 함
  클라이언트가 연결 요청을 하면 accept() 메소드로 연결 수락

# ServerSocket 생성과 연결 수락

```java
public class ServerExample {
    public static void main(String[] args) {
        ServerSocket serverSocket = null;
        try {
            serverSocket = new ServerSocket();
            serverSocket.bind(new InetSocketAddress( hostname: "localhost",  port: 5001));
            while(true) {
                System.out.println("[연결 기다림]");
                Socket socket = serverSocket.accept();
                InetSocketAddress isa = (InetSocketAddress) socket.getRemoteSocketAddress();
                System.out.println("연결 수락함 " + isa.getHostName());
            }
        } catch (Exception e) {}

        if (!serverSocket.isClosed()){
            try {
                serverSocket.close();
            } catch (IOException e) {}
        }
    }
}
```

- ServerSocket을 만들어 바인딩 후
  accept() 메소드를 통해 연결
  SocketAddress를 받아 저장 (아래 표 참고)
  ServerSocket의 상태를 확인 후 닫음

| 리터타입 | 메소드명(매개변수) | 설명 |
|---|---|---|
| String | getHostName() | 클라이언트 IP 리턴 |
| int | getPort() | 클라이언트 포트 번호 리턴 |
| String | toString() | "IP:포트번호" 형태의 문자열 리턴 |

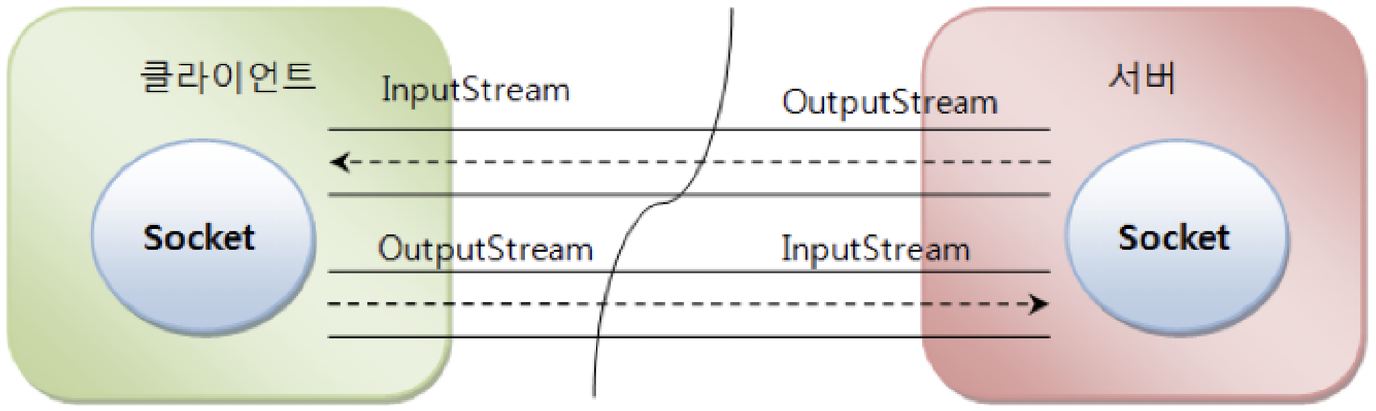
# Socket 생성과 연결 요청

```java
public class ClientExample {
    public static void main(String[] args) {
        Socket socket = null;
        try {
            socket = new Socket();
            System.out.println("[연결 요청]");
            socket.connect(new InetSocketAddress( hostname: "localhost",  port: 5001));
            System.out.println("[연결 성공]");
        }catch (Exception e) {}

        try {
            if (!socket.isClosed()) {
                socket.close();
            }
        } catch (IOException e) {}
    }
}
```

- 클라이언트가 서버에 연결 요청을 위해 Socket 사용
  connect() 메소드를 통해 연결 요청
  연결 요청을 위해 IP 주소와 바인딩 포트 번호를 매개값으로

# Socket 데이터 통신



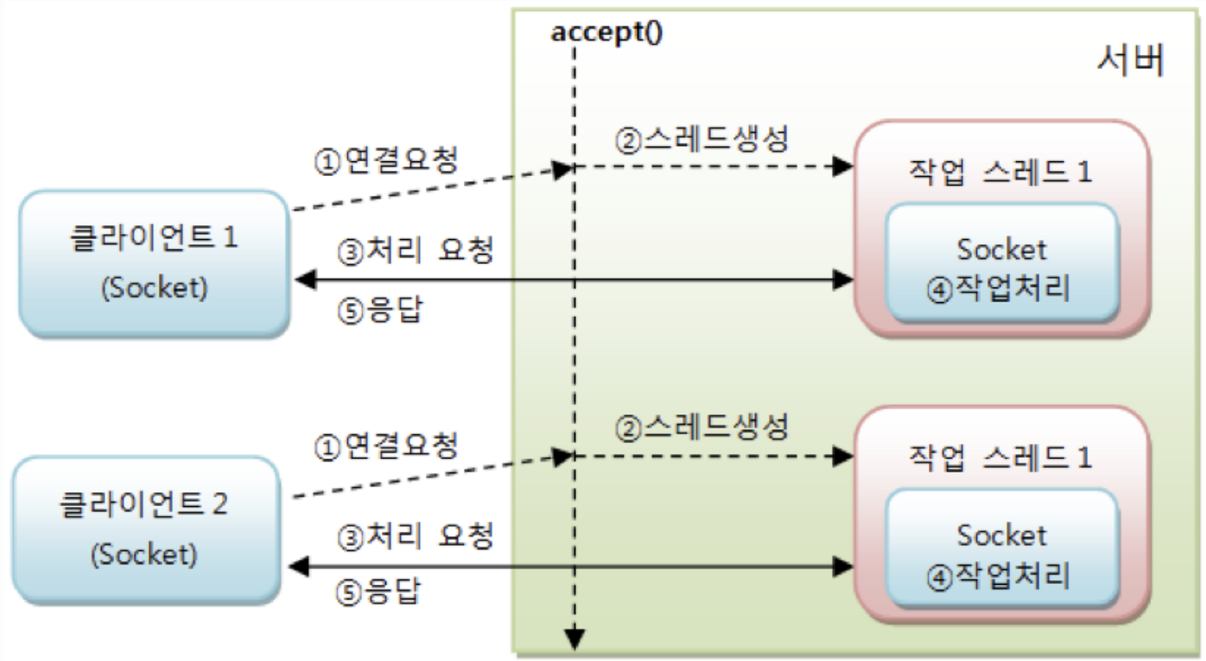

| OutputStream | InputStream |
| --- | --- |
| OutputStream os = socket.getOutputStream();<br>message = "Hello Server";<br>bytes = message.getBytes("UTF-8");<br>os.write(bytes);<br>os.flush(); | InputStream is = socket.getInputStream();<br>bytes = new byte[100];<br>int readByteCount = is.read(bytes);<br>message = new String(bytes, 0, readByteCount,"UTF-8"); |

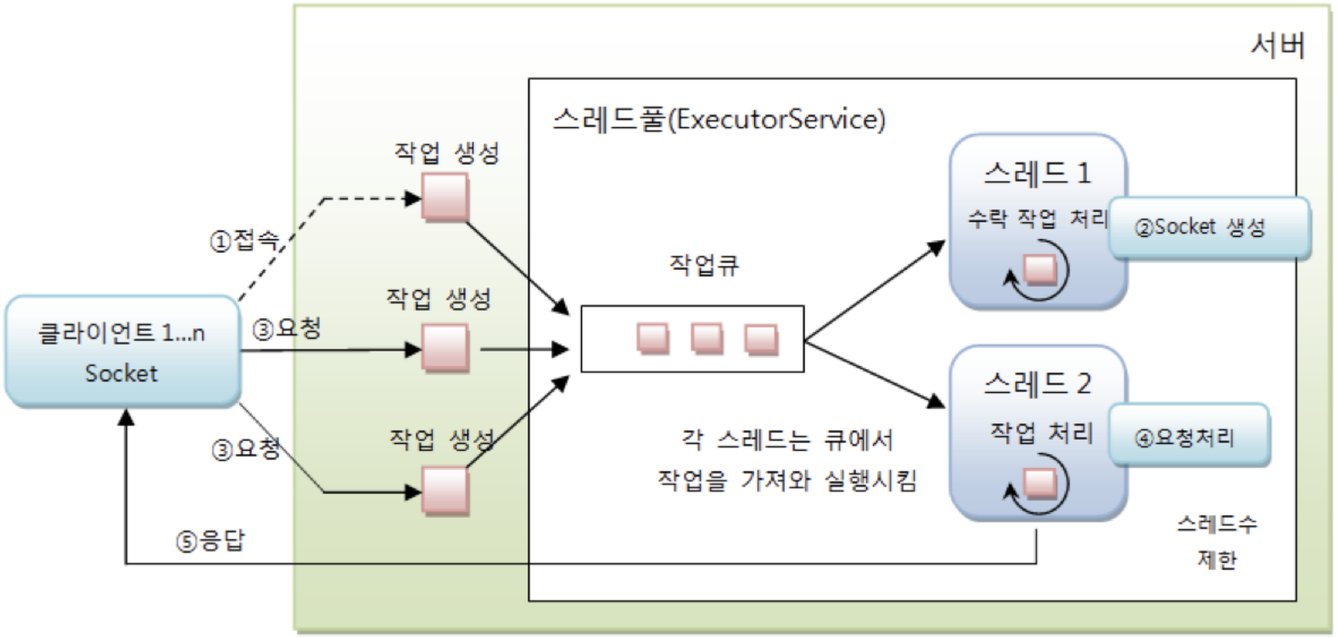
# 스레드 병렬 처리

- accept(), connect(), read(), write()는 해당 작업이 완료되기 전까지 블로킹이 되기 때문에 별도의 작업 스래드를 생성해 병렬적으로 처리하는 것이 좋음

  그림 1. 메인 스레드에서 직접 작업

  그림 2. 병렬 처리



Chapter 08

# UDP 네트워킹

# UDP 네트워킹

- UDP(User Datagram Protocol) : 비연결 지향적 프로토콜
  지연결 지향적이란? 데이터를 주고 받을 때 연결을 하지 않고 발신자가 일방적으로 데이터를 발신하는 방식
  연결과정이 생략되어 TCP보다 빠른 전송이 가능하지만 데이터 전달의 신뢰성은 하락
  > 데이터 패킷을 순차적으로 보내더라도 패킷은 서로 다른 통신 선로를 통해 전달될 수 있어 데이터 순서가 섞이거나 잃어버릴 수 있음


- java.net.DatagramSocket : 발신점과 수신점
  java.net.DatagramPacket : 주고받는 패킷

# 발신자 구현

```java
public class UdpSendExample {
    public static void main(String[] args) throws IOException {
        DatagramSocket datagramSocket = new DatagramSocket();
        System.out.println("[발신 시작]");
        for(int i = 1; i < 3; i++) {
            String data = "메세지" + i;
            byte[] byteArr = data.getBytes( charsetName: "UTF-8");
            DatagramPacket packet = new DatagramPacket(
                    byteArr, byteArr.length,
                    new InetSocketAddress( hostname: "localhost",  port: 5001)
            );
            datagramSocket.send(packet);
        }
        System.out.println("[발신 종료]");
        datagramSocket.close();
    }
}
```

- SatagramSocket을 생성
  전달할 데이터를 send() 메소드를 통해 전달
  close() 메소드를 호출하여 DatagramSocket을 닫음

# 수신자 구현

```java
public class UdpReceiveExample extends Thread {
    public static void main(String[] args) throws Exception {
        DatagramSocket datagramSocket = new DatagramSocket( port: 5001);
        Thread thread = new Thread() {
            @Override
            public void run() {
                System.out.println("[수신 시작]");
                try {
                    while (true) {
                        DatagramPacket packet = new DatagramPacket(new byte[100], length: 100);
                        datagramSocket.receive(packet);
                        String data = new String(packet.getData(), offset: 0, packet.getLength(), charsetName: "UTF-8");
                        System.out.println("[받은 내용: ]" + packet.getSocketAddress() + "] " + data);
                    }
                } catch (Exception e) {
                    System.out.println("[수신 종료]");
                }
            }
        };
        thread.start();

        Thread.sleep( millis: 10000);
        datagramSocket.close();
    }
}
```

- 5001번 포트를 수신하는 DatagramSocket을 생성
  receive() 메소드를 호출 해 패킷을 읽을 준비
  패킷을 받을 때 까지 블로킹 상태
  패킷이 들어오면 DatagramSocket에 저장

  DatagramPacket은 저장할 바이트와 읽을 수 있는 최대 바이트 수를 매개값으로 생성
  이 때 최대 바이트 수는 배열의 크기보다 크거나 같음

  패킷을 받고 getSocketAddress를 통해 발신자의 IP와 포트를 얻어 응답을 보낼 때 사용

감사합니다.