네트워킹

강사 : 강병준

네트워킹의 개요

네트워크(Network)란 같은 데이터 전송 프로토콜(protocol)을 가지고 통신을 하는 연결된 장치들을 총칭하는 것을 말한다.
여러 장치들이 서로 교신하기 위해서는 같은 의사소통방식을 사용해야 하는데 이런 역할을 하는 것을 프로토콜이라 한다.

프로토콜은 컴퓨터와 컴퓨터가 통신 하기 위한 일종의 규약으로 서로 다른 언어를 사용하는 사람 사이에 의사소통이 안되듯이 네트워크 에서도 동일한 규격의 프로토콜을 사용해야만 네트워크가 가능 하다.

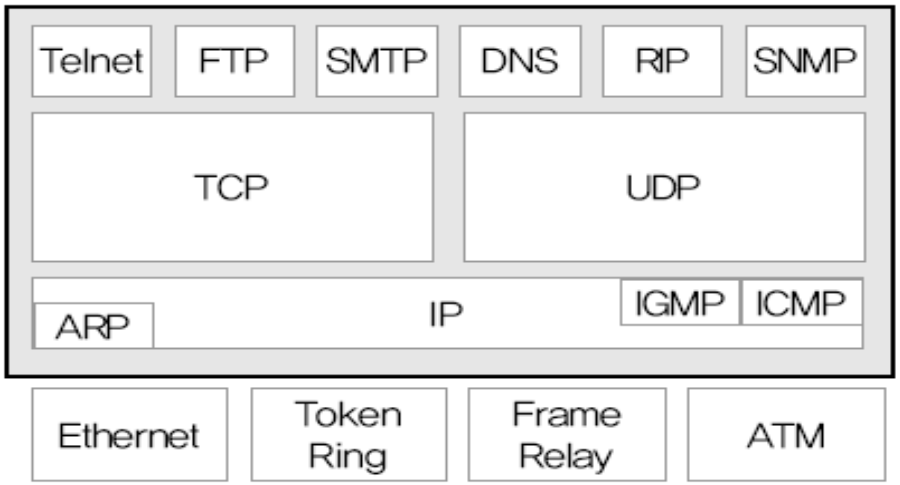
사전적 의미로 전선이나 혈관, 통로 등으로 이루어진 망형 조직을 말하는 것으로 IT에서 네트워크는 컴퓨터와 컴퓨터를 연결해주는 망을 의미한다.

일상적으로 사용하고 있는 인터넷은 네트워크 응용 서비스의 한 종류로 TCP/IP 라고 하는 통신 프로토콜에 기반하고 있다.

- TCP/IP(Transmission Control Prorocol/Internet Protocol)는 컴퓨터 통신을 위한 프로토콜 중 하나로 우리가 사용하는 인터넷의 기반이 된다.
- TCP/IP가 인터넷의 기반 프로토콜이 된 이유는 하드웨어, 운영체제, 접속 매체와 관계없이 동작할 수 있는 개방형 구조이기 때문이다.
- TCP/IP 는 보다 큰 네트워크 프로토콜 개념인 OSI 7 Layer 에서 유래한 것으로 복잡성을 단순화 한 4계층 구조를 가진다.

OSI (Open System Interconnection)

- 개방 구조 상호연결 이라는 뜻으로 다른 기종간의 통신시 네트워크 구조에 상관없이 통신을 할 수 있도록 국제표준으로 정해놓은 통신상황을 개방할 수 있도록 하는 규약



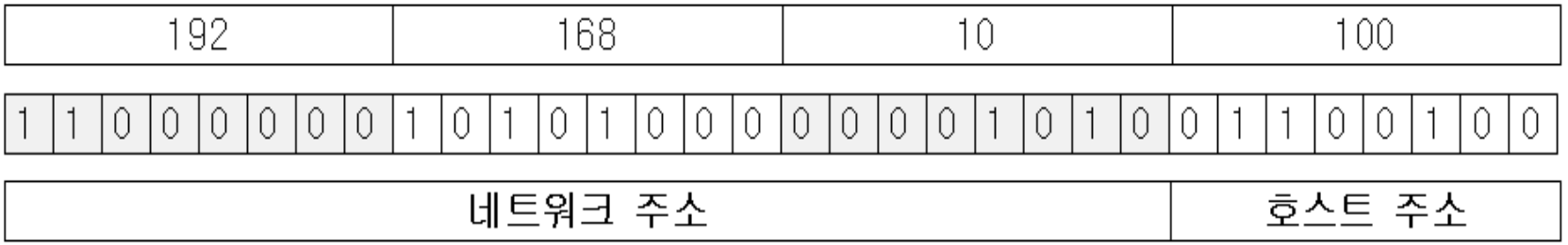
클라이언트/서버(client/server)

- 컴퓨터간의 관계를 역할(role)로 구분하는 개념
- 서비스를 제공하는 쪽이 서버, 제공받는 쪽이 클라이언트가 된다.
- 제공하는 서비스의 종류에 따라 메일서버(email server), 파일서버(file server), 웹서버(web server) 등이 있다.
- 전용서버를 두는 것을 '서버기반 모델', 전용서버없이 각 클라이언트가 서버역할까지 동시에 수행하는 것을 'P2P 모델'이라고 한다.

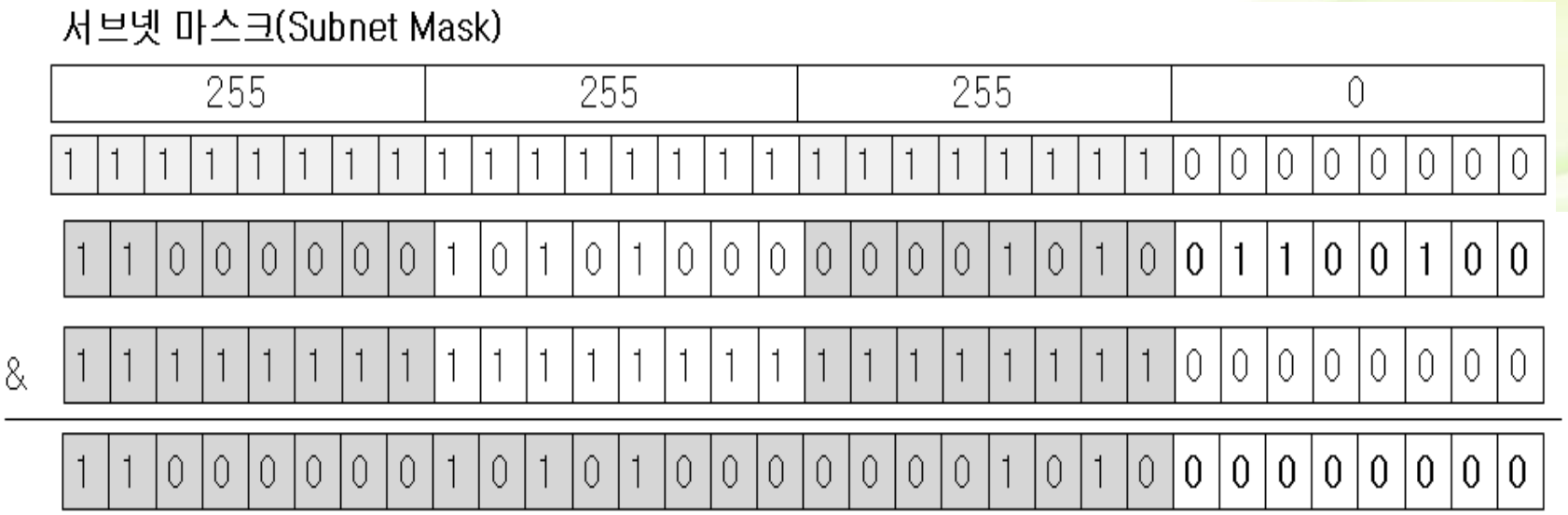
서버기반 모델(server-based model)	P2P 모델(peer-to-peer model)
- 안정적인 서비스의 제공이 가능하다. - 공유 데이터의 관리와 보안이 용이하다. - 서버구축비용과 관리비용이 든다.	- 서버구축 및 운용비용을 절감할 수 있다. - 자원의 활용을 극대화 할 수 있다. - 자원의 관리가 어렵다. - 보안이 취약하다.

IP주소(IP address)

- 컴퓨터(host, 호스트)를 구별하는데 사용되는 고유한 주소값
- 4 byte의 정수로 'a.b.c.d'와 같은 형식으로 표현.(a,b,c,d는 0~255의 정수)
- IP주소는 네트워크주소와 호스트주소로 구성되어 있다.



- 네트워크주소가 같은 두 호스트는 같은 네트워크에 존재한다.
- IP주소와 서브넷마스크를 '&'연산하면 네트워크주소를 얻는다.



IP 주소

- 네트워크에 연결된 컴퓨터를 구분하기 위해 사용
- 4개로 구분된 10진수를 사용함.
- 사설 IP는 NAT(Network Access Translator) 등을 이용해서 인터넷 접속 시
- 공인 IP로 매핑됨(일부 인터넷 서비스에 제약이 있을 수 있음)
- IP 주소 부족 문제를 해결하기 위해 IPV6가 등장함
- IP주소 구분

구분	범위	사용 목적
클래스 A	1.0.0.0~127.0.0.0	대형 통신망
클래스 B	128.0.0.0~191.255.0.0	중형 통신망, 주소 65536개 할당
클래스 C	192.0.0.0~223.255.255.0	소형 통신망, 주소 256개 할당
클래스 D	-	멀티 캐스트용으로 예약, 배포 중지
클래스 E	-	실험 목적, 배포 중지1

인터넷과 웹 프로그래밍

네트워크>>도메인 이름

- IP 주소를 알기 쉬운 이름으로 바꾼 것
- DNS(Domain Name System) 서버가 필요함.

- DNS 처리과정

인터넷과 웹 프로그래밍

- Internet : 일반적으로 우리가 알고 있는 인터넷으로 고유명사화 되었음.
- internet : 내부 네트워크를 의미하는 네트워크 용어
- 인터넷과 www 서비스
  - 인터넷은 TCP/IP 기반의 네트워크가 전세계적으로 확대되어 하나로 연결된 '네트워크의 네트워크'
  - 인터넷 = www가 아님. www는 인터넷 기반의 서비스 중 하나
  - 대표적인 인터넷 기반 서비스

이름	프로토콜	포트	기능
www	http	80	웹 서비스
Email	SMTP/POP3/IMAP	25/110/114	이메일 서비스
FTP	ftp	21	파일 전송 서비스
telnet	telnet	23	원격 로그인
DNS	DNS	83	도메인 이름 변환 서비스
News	NNTP	119	인터넷 뉴스 서비스

소켓

통신을 하기 위해 사용되는 TCP 포트를 나타내는 말이며 하부 네트워크에 신경 쓰지 않고 통신하기 위한 소프트웨어적인 메커니즘(mechanism)이다.

소켓은 전구 소켓과 같이 컴퓨터가 연결된 통신 끝 점에서 멀리 떨어져 있는 컴퓨터끼리 데이터를 송수신하도록 하기 위해 두 컴퓨터를 연결시켜 주는 도구라고 할 수 있다.

서버 : 정보나 서비스를 제공하는 측

클라이언트 : 정보나 서비스를 제공 받는 측

IP 주소 : 인터넷에 연결된 모든 컴퓨터을 구분하기 위해서 사용하는 것

포트 번호 : 전송된 데이터를 특정 프로그램으로 보내기 위해서는 각 응용 프로그램을 구분해야 하는데 이때 사용되는 것이 포트 번호이다.

InetAddress

InetAddress 클래스는 자바 프로그램 안에서 IP 번호와 URL 주소를 알아내고 싶을 때 사용한다.

메서드	설명
boolean equals(InetAddress other)	other 객체와 같은 주소를 갖고 있으면 true, 아니면 false 반환
byte[] getAddress()	주소를 갖고 있는 4개 요소의 바이트 배열 반환
String getHostAddress()	주소 정보를 나타내는 문자열 반환
String getHostName()	컴퓨터 이름 반환
InetAddress getLocalHost()	현재 컴퓨터의 InetAddress 객체 반환
InetAddress getByName(String host)	host 로 지정된 컴퓨터를 나타내는 InetAddress 객체 반환
InetAddress[] getAllByName(String host)	host로 지정된 모든 컴퓨터를 InetAddress 객체로 반환 (하나의 도메인 명으로 여러대의 컴퓨터가 작동할 경우)

InetAddress

```
import java.net.*;
class InetAddress1{
  public static void main(String[] args) throws
                                    UnknownHostException{
    InetAddress addr1 =
        InetAddress.getByName("www.choongang.co.kr");
    InetAddress addr2 =
        InetAddress.getByName("211.183.8.85");
    InetAddress addr3 = InetAddress.getLocalHost();
    System.out.println("============================");
    System.out.println( "addr1 = " + addr1.getHostAddress());
    System.out.println( "addr2 = " + addr2.getHostName());
    System.out.println( "로컬 주소 = " +
                addr3.getHostAddress());
    System.out.println( "로컬 이름 = " + addr3.getHostName());
  }
}
```

InetAddress

```
package ch17;
import java.net.*;
import java.util.Scanner;
public class InetAd1 {
	public static void main(String[] args) throws
UnknownHostException {
		Scanner sc = new Scanner(System.in);
		System.out.println("도메인을 입력하세요");
		String domain = sc.nextLine();
		InetAddress ia = InetAddress.getByName(domain);
		System.out.println("호스트 이름 : "+ia.getHostName());
		System.out.println("IP주소 : "+ia.getHostAddress());
		sc.close();
	}
}
```

URL 클래스

클래스명	메소드명	기능
URL	URL(String spc)	매개 변수 spec으로 URL 객체를 생성한다.
	URL(String protocol, String host, int port, String file)	protocol과 host, port와 file로 URL 객체를 생성한다.
	URL(String protocol, String host, String file)	protocol과 host, file로 URL 객체를 생성한다.

클래스명	메소드명	기능
URL	getAuthority()	URL의 호스트 명과 포트를 결합한 문자열을 얻음
	getPort()	URL의 포트 번호를 얻음
	getDefaultPort()	프로토콜의 default 포트 번호를 얻음
	getFile()	URL의 파일 명을 얻음
	getHost()	URL의 호스트 컴퓨터 명을 얻음
	getPath()	URL의 경로를 얻음
	getProtocol()	URL의 프로토콜을 얻음
	getQuery()	URL의 쿼리를 문자열로 얻음
	getRef()	URL의 참조를 문자열로 얻음
	openConnection()	URLConnection 객체를 생성해줌
	openStream()	InputStream의 객체를 생성해줌
	toExternalForm()	URL을 문자열로 얻음

URL 클래스

```
import java.net.*;
import java.util.Scanner;
public class Url1 {
        public static void main(String[] args) throws
MalformedURLException {
                Scanner sc = new Scanner(System.in);
                System.out.println("url을 입력하세요");
                String addr = sc.nextLine();
                URL url = new URL(addr);
                System.out.println("프로토콜 : "+url.getProtocol());
                System.out.println("호스트 : "+url.getHost());
                System.out.println("포트 : "+url.getPort());
                System.out.println("파일경로: " + url.getFile());
                sc.close();
        }
}
```

URConnection 클래스

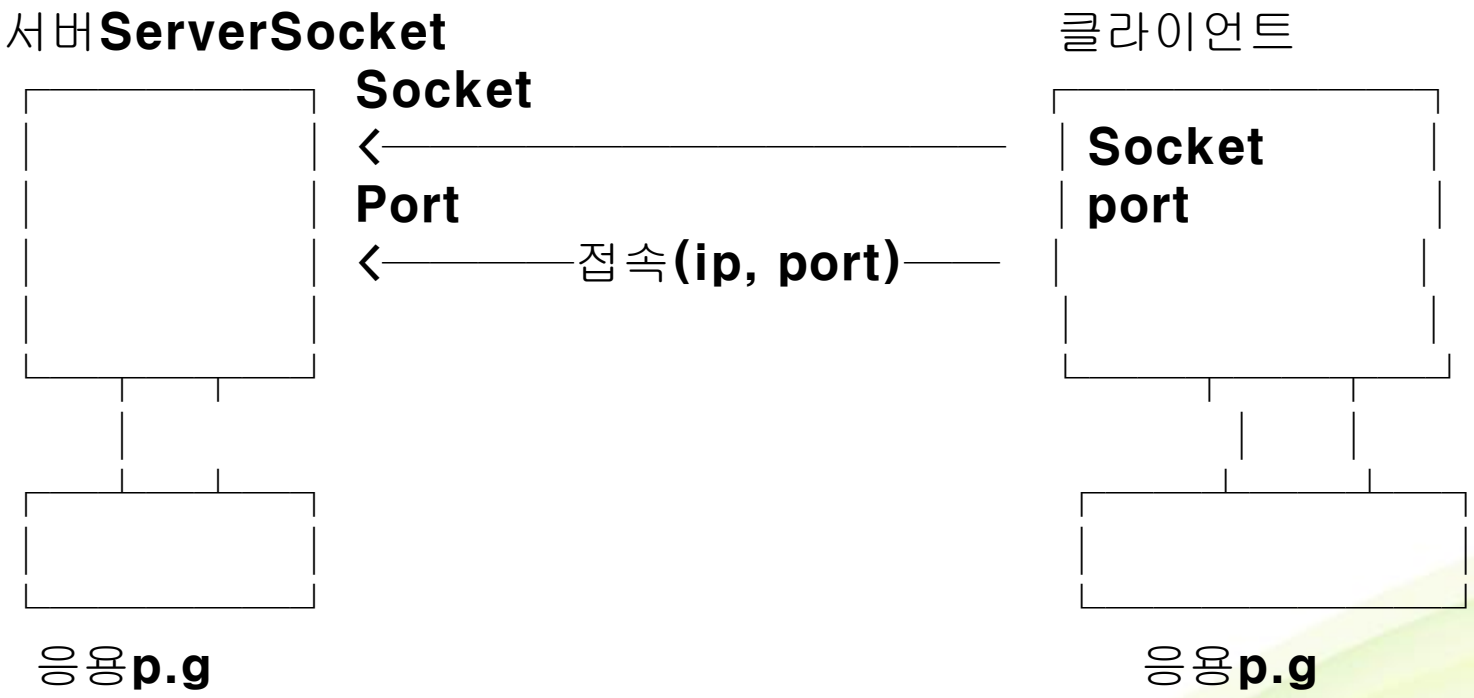
클래스명	메소드명	기능
URLConnection	URLConnection(URL url)	매개 변수 url으로 URL 객체를 생성한다.

클래스명	메소드명	기능
URLConnection	getContentEncoding()	헤더 필드의 content-encoding에 대한 값을 얻는다.
	getContentLength()	헤더 필드의 content-length에 대한 값을 얻는다.
	getHeaderField (String name)	헤더 필드의 이름에 대한 값을 얻는다.
	getHeaderFields()	헤더 필드의 구조를 map으로 변환한다.
	getInputStream()	입력하기 위하여 InputStream 객체를 얻는다.
	getOutputStream()	출력하기 위하여 OutputStream 객체를 얻는다.
	getURL()	URL 필드의 값을 얻는다.

URConnection 클래스

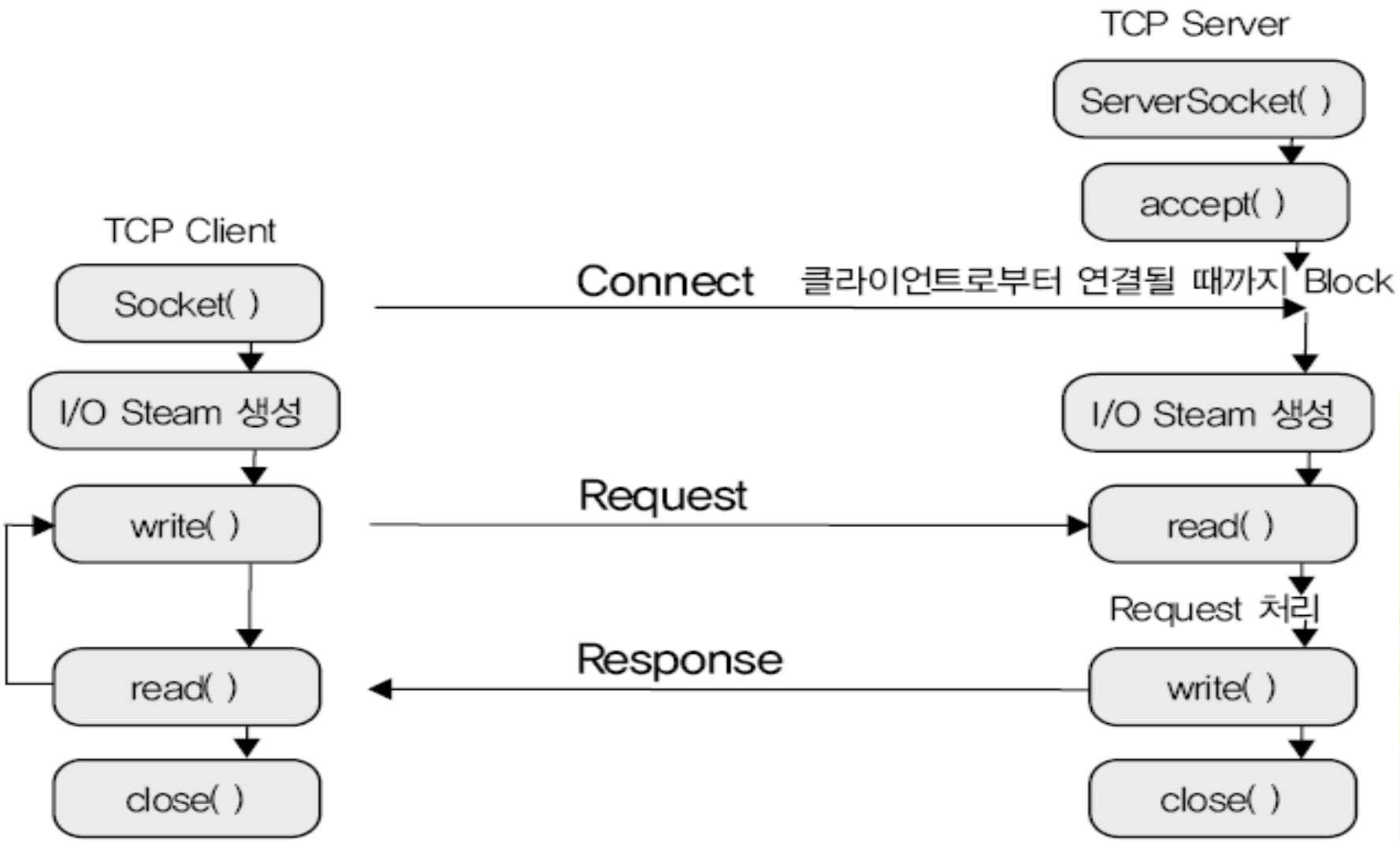
```
public class URLConn1 {
	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		BufferedReader br = null;
		System.out.println("url을 입력하세요");
		String addr = sc.nextLine();
		try {
			URL url = new URL(addr);
			URLConnection uc = url.openConnection();
			InputStream is = uc.getInputStream();
			br = new BufferedReader(
				new InputStreamReader(is, "utf-8"));
			String str = null;
			while((str=br.readLine()) != null) {
				System.out.println(str);
			}
			br.close();
		} catch (Exception e) {
			System.out.println(e.getMessage());
		}
		sc.close();
	}
}
```

TCP/IP



① **Socket (**전화기 비유**)**
② **port(**전화선 비유**)**
③ **ip address(**전화번호**)**

TCP/IP


TCP Server
ServerSocket( )
accept( )
TCP Client
Socket( )
Connect
클라이언트로부터 연결될 때까지 Block
I/O Steam 생성
I/O Steam 생성
Request
write( )
read( )
Request 처리
Response
read( )
write( )
close( )
close( )

TCP/IP

자바에서는 서버 클라이언트 프로그램을 작성하기 위해서 java.net 패키지에 ServerSocket 와 Socket 클래스를 제공한다.

ServerSocket 클래스는 서버 측에서 실행되는 애플리케이션을 작성하기 위해서 사용되는데 다음과 같은 형태의 생성자를 제공해 준다.

생성자	
ServerSocket(int port)	클라이언트 요청을 받아들일 포트(port)번호를 갖고 ServerSocket 객체 생성

ServerSocket 클래스의 주요 메서드

메서드	
Socket accept()	클라이언트 요청 받아 들인 다음 Socket 객체 생성해서 반환
void close()	서버 소켓 닫기

서버로 접속할 클라이언트 프로그램에서는 Socket 클래스를 생성해야 한다. Socket 클래스는 다음과 같은 생성자를 제공해준다.

생성자	설명
Socket(String host, int port)	host : 접속할 서버의 IP 번호 port : 접속할 포트 번호 addr : 접속할 서버의 InetAddress 객체
Socket(InetAddress addr, int port)	

데이터를 송수신기 위해서는 입력 출력 스트림을 생성해야 한다.

메서드	
InputStream getInputStream()	현재 소켓과 관련된 InputStream 객체 반환
OutputStream getOutputStream()	현재 소켓과 관련된 OutputStream 객체 반환
void close()	소켓 닫기
InetAddress getInetAddress()	소켓에 연결된 원격 컴퓨터의 InetAddress 객체 반환
InetAddress getLocalAddress()	소켓에 연결된 지역 컴퓨터의 InetAddress 객체 반환
int getPort()	소켓에 연결된 컴퓨터의 포트 번호 반환
int getLocalPort()	소켓에 연결된 지역 컴퓨터의 포트 번호 반환

Time서버

```
import java.io.*;
import java.net.*;
import java.util.GregorianCalendar;
public class TimeServer {
   public static void main(String[] args) throws IOException {
         ServerSocket ss = new ServerSocket(7000);
         try {
                  while(true) {
                     Socket client = ss.accept();
                     OutputStream os  = client.getOutputStream();
                     ObjectOutputStream oos=new ObjectOutputStream(os);
                     oos.writeObject(new GregorianCalendar());
                     oos.flush();
                     os.close();
                     oos.close();
                     client.close();
                  }
         }catch(Exception e) {
              System.out.println(e.getMessage());
          }
         ss.close();
    }
}
```

Time Client

```
import java.io.*;
import java.net.*;
import java.util.GregorianCalendar;
import java.util.Scanner;
public class TimeClient {
    public static void main(String[] args) throws UnknownHostException, I
                        OException, ClassNotFoundException {
            Scanner sc = new Scanner(System.in);
            System.out.println("연결할 ip를 입력하세요");
            String ip = sc.nextLine();
            Socket client = new Socket(ip, 7000);
            InputStream is = client.getInputStream();
            ObjectInputStream ois = new ObjectInputStream(is);
            GregorianCalendar gc=(GregorianCalendar)ois.readObject();
            System.out.printf("현재 %TF %TT",gc,gc);
            is.close();
            ois.close();
            client.close();
            sc.close();
    }
}
```

채팅 서버

```
import java.io.*;
import java.net.*;
public class ChatServer {
    public static void main(String[] args) throws IOException {
        ServerSocket ss = null;
        try{
                ss = new ServerSocket(7001);
                while(true) {
                        Socket client = ss.accept();
                        DataInputStream dis=
                            new DataInputStream(client.getInputStream());
                        BufferedReader br =
                            new BufferedReader(new InputStreamReader(dis));
                        System.out.println(client.getInetAddress()+" => "+
                                br.readLine());
                        dis.close();
                        br.close();
                        client.close();
                }
        }catch(Exception e) {
                System.out.println(e.getMessage());                }
        ss.close();
    }
}
```

채팅 Client

```
import java.io.*;
import java.net.*;
import java.util.Scanner;
public class ChatClient {
    public static void main(String[] args) throws
                    UnknownHostException, IOException {
        Scanner sc = new Scanner(System.in);
        System.out.println("연결할 ip ?");
        String ip  = sc.nextLine();
        System.out.println("전달할 메세지 ?");
        String msg = sc.nextLine();
        Socket client = new Socket(ip, 7001);
        OutputStream os = client.getOutputStream();
        BufferedWriter bw =
                    new BufferedWriter(new OutputStreamWriter(os));
        bw.write(msg); bw.flush();
        os.close(); bw.close(); client.close();
        sc.close();
        System.out.println("메세지 전달 완료");
    }
}
```

TCP 소켓

상호 연결된 상태에서 통신하는 방법이기에 높은 신뢰성이 요구되는 애플리케이션에 적합하다.

하지만, 높은 신뢰성이 요구되지 않거나 많은 양의 데이터를 전송할 경우에 TCP 소켓은 적합하지 않다. 왜냐하면 네트워크에 부담을 주기 때문이다.

UDP

User Datagram Protocol의 약자로서 TCP 소켓에 비해 신뢰성과 안정성 측면에서는 뒤떨어진다.

연결을 설정하지 않으므로 네트워크의 부담을 주지 않는다는 장점이 있다.

비연결형 소켓 클래스의 메소드

클래스명	메소드명	기능
DatagramSocket	DatagramSocket()	데이터 그램 소켓을 생성함
	close()	데이터 그램 소켓을 종료함
	receive()	소켓을 사용해 데이터그램을 읽어옴
	send()	소켓을 사용해 목적지로 데이터그램을 전송함
	getLocalPort()	이 소켓이 결합된 로컬 포트를 읽어옴
DatagramPacket	DatagramPacket()	송수신할 데이터를 위한 데이터 그램 패킷의 인스턴스를 생성함
	getAddress()	패킷의 목적지 주소를 얻음
	getData()	패킷의 데이터 내용을 얻음
	getLength()	패킷의 길이를 얻음
	getPort()	패킷 목적지의 포트를 얻음

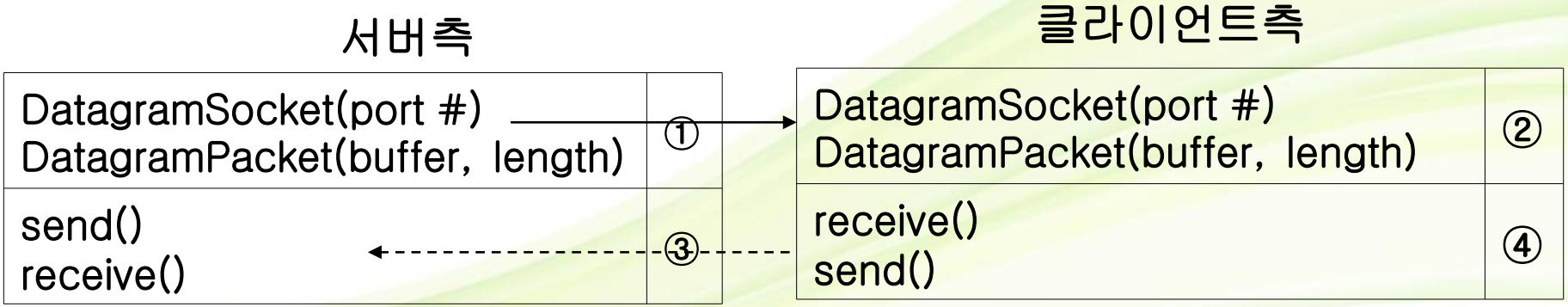
UDP 프로토콜의 동작 원리

① 서버 측에서는 먼저 java.net 패키지에 포함되어 있는 DatagramSocket, DatagramPacket 클래스를 생성한다. TCP처럼 클라이언트 측으로부터의 새로운 연결을 얻기 위한 accept() 메소드 사용 등은 필요가 없다.

② 마찬가지로, 클라이언트 측에서도 java.net 패키지에 포함되어 있는 DatagramSocket, DatagramPacket 클래스를 생성한다.

③ 서버 측에서는 클라이언트 측과 정보를 송수신하기 위하여 상대 측에 보낼 때는 send() 메소드를 사용하고, 상대 측으로부터 받을 때는 receive() 메소드를 사용한다.

④ 마찬가지로, 클라이언트 측에서는 서버 측과 정보를 송수신하기 위하여 상대 측에게 보낼 때는 send() 메소드를 사용하고, 상대 측으로부터 받을 때는 receive() 메소드를 사용한다.



UDP 프로토콜

다음은 UDP를 통하여 데이터를 전송하기 위한 DatagramPacket을 사용한 예이다.

예

```
try {
  InetAddress iaddr = new InetAddess("127.0.0.1");
  String s = "송지영"
  byte[] buffer = s.getBytes();
  DatagramPacket dp =
        new DatagramPacket(buffer, buffer.length, iaddr, 6000);
}
catch (UnknownHostException e) {
  System.err.println(e);
}
```

우선 전송할 데이터를 바이트 배열형태로 DatagramPacket으로 포장해야 한다.

DatagramPacket에는 전송할 곳의 주소와 포트 번호를 기술해야 한다.

UDP 프로토콜

패킷이 만들어지면 DatagramSocket 클래스 객체를 만든 후 send() 메서드를 호출하여 앞에서 만들어진 데이터 그램을 전송하면 된다.

예

```
try {
  DatagramSocket ds = new DatagramSocket();
  ds.send(dp);
}
catch (IOException e) {
  System.err.println(e);
}
```

DatagramSocket 객체를 통해 DatagramPacket 객체로 만들어진 데이터를 전송한다.

이때 DatagramSocket 객체(우체국)는 특별한 포트 번호를 가지지 않아도 된다.

이미 DatagramPacket 객체(소포)에 전송할 곳의 주소를 기재했기 때문이다.

UDP 프로토콜

UDP를 이용하여 데이터를 받기 위한 내용이다.

예

```
try {
  byte buffer = new byte[65536];
  //수신용 데이터그램 패킷
  DatagramPacket dp = new DatagramPacket(data, data.length);
  DatagramSocket ds = new DatagramSocket(6000);
  ds.receive(dp);
}
catch (IOException e) {
  System.err.println(e);
}
```

데이터를 받는 곳의 DatagramSocket 객체는 주소는 필요 없고 포트 번호만 열어 두기만 하면 된다.

데이터를 받는 곳에는 전송된 데이터를 보관하는 공간이 필요한데 이를 위해서 DatagramPacket 객체가 있어야 한다.

UDP 프로토콜

```
import java.net.*;
public class UdpServer {
    public static void main(String[] args) throws IOException {
      byte[] bt = new byte[50];
       DatagramPacket dp = new DatagramPacket(bt, bt.length);
       DatagramSocket ds = new DatagramSocket(5007);
       String msg = "";
        while(true) {
            ds.receive(dp);
            InetAddress addr = dp.getAddress();
            msg = new String(dp.getData(),0,dp.getLength());
            System.out.println(addr+"에서 메세지 : " + msg);
        }
   }
}
```

UDP 프로토콜

```
import java.net.*;    import java.io.*;
public class UdpClient {
    public static void main(String[] args) throws IOException {
        Scanner sc = new Scanner(System.in);
        System.out.println("연결할 ip를 입력 하세요");
        String ip = sc.nextLine();
        System.out.println("전달할 메세지를 입력 하세요");
        String msg = sc.nextLine();

        DatagramSocket ds = new DatagramSocket();
        InetAddress addr = InetAddress.getByName(ip);
        byte[] bt = msg.getBytes(); // 메세지를 byte단위로 변경
        DatagramPacket dp =
                new DatagramPacket(bt, bt.length,addr,5007);
        ds.send(dp);
        System.out.println("보내기 성공");
        ds.close();
        sc.close();
    }
}
```

여러 사용자가 함께 사용하는 채팅 : 서버 쪽

MultiServer.java

```
import java.io.*;
import java.net.*;
public class MultiServer {
    public static void main(String[] args) throws IOException {
        ServerSocket ss = null;
        try {
            ss = new ServerSocket(9001);
            while(true) {
                Socket client = ss.accept();
                Thread th = new PerClientThread(client);
                th.start();
            }
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
        ss.close();
    }
}
```

PerClinetThread.java

```
import java.io.*;
import java.net.*;
import java.util.*;
class PerClinetThread extends Thread {
   static List<PrintWriter> list =
            Collections.synchronizedList(new ArrayList<PrintWriter>());
   Socket socket;      PrintWriter writer;
   PerClinetThread(Socket socket) {
      this.socket= socket;
      try {
            writer = new PrintWriter(socket.getOutputStream());
            list.add(writer);
      }  catch (Exception e) {
           System.out.println(e.getMessage());
      }
   }
   public void run() {
      String name = null;
      try {
          BufferedReader reader = new BufferedReader(
                      new InputStreamReader(socket.getInputStream()));
          name = reader.readLine();
          sendAll("#" + name + "님이 들어오셨습니다");
```

```
            while (true) {
                String str = reader.readLine();
                if (str == null)                    break;
                sendAll(name + ">" + str);
            }
        }   catch (Exception e) {
            System.out.println(e.getMessage());
        }   finally {
                list.remove(writer);
                sendAll("#" + name + "님이 나가셨습니다");
                try {   socket.close();
            }   catch (Exception ignored) {         }
        }
    }
    private void sendAll(String str) {
        for (PrintWriter writer : list) {
            writer.println(str);
            writer.flush();
        }
    }
}
```

여러 사용자가 함께 사용하는 채팅 : 클라이언트 쪽

MutiClient.java

```
import java.io.*;
import java.net.*;
import java.util.*;
public class MutiClient {
   public static void main(String[] args) {
          Scanner sc = new Scanner(System.in);
          System.out.println("연결할 ip ?");
          String ip = sc.nextLine();
          System.out.println("사용할 별명");
          String name = sc.nextLine();
          try {
                    Socket client = new Socket(ip, 9001);
                    Thread send    = new SendThread(client, name);
                    Thread receive = new ReceiveThread(client);
                    send.start();
                    receive.start();
          }catch (Exception e){
                    System.out.println(e.getMessage());
          }
    }
}
```

SenderThread.java

```
import java.net.*;   import java.io.*;
class SenderThread extends Thread {
   Socket socket;        String name;
   SenderThread(Socket socket, String name) {
      this.socket = socket;        this.name = name;
   }
   public void run() {
      try {  BufferedReader reader = new BufferedReader(
                  new InputStreamReader(System.in));
         PrintWriter writer = new PrintWriter(socket.getOutputStream());
         writer.println(name);
         writer.flush();
         while (true) {
            String str = reader.readLine();
            if (str.equals("bye"))   break;
            writer.println(str);       writer.flush();
         }
      }   catch (Exception e) {  System.out.println(e.getMessage());
      }   finally {
         try {   socket.close();
         }        catch (Exception ignored) {          }
      }
   }
}
```

ReceiverThread.java

```
import java.io.*;
import java.net.*;
class ReceiverThread extends Thread {
    Socket socket;
    ReceiverThread(Socket socket) {
        this.socket = socket;
    }
    public void run() {
        try {
            BufferedReader reader =   new BufferedReader(
                        new InputStreamReader(socket.getInputStream()));
            while (true) {
                String str = reader.readLine();
                if (str == null)
                    break;
                System.out.println(str);
            }
        }
        catch (IOException e) {
            System.out.println(e.getMessage());
        }
    }
}
```