Java의 정석

제 15 장

네트워킹(Networking)

1. 네트워킹(Networking)

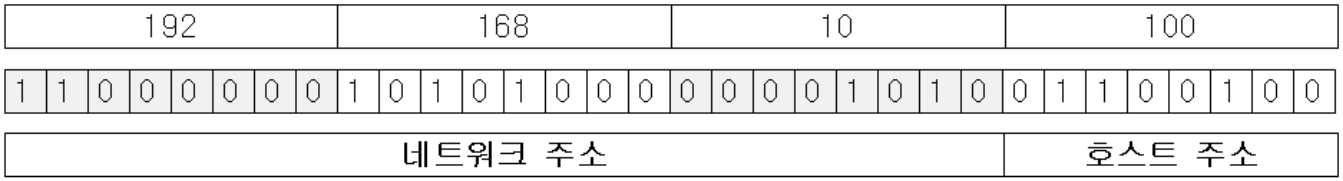
1.1 클라이언트/서버(client/server)

- 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.

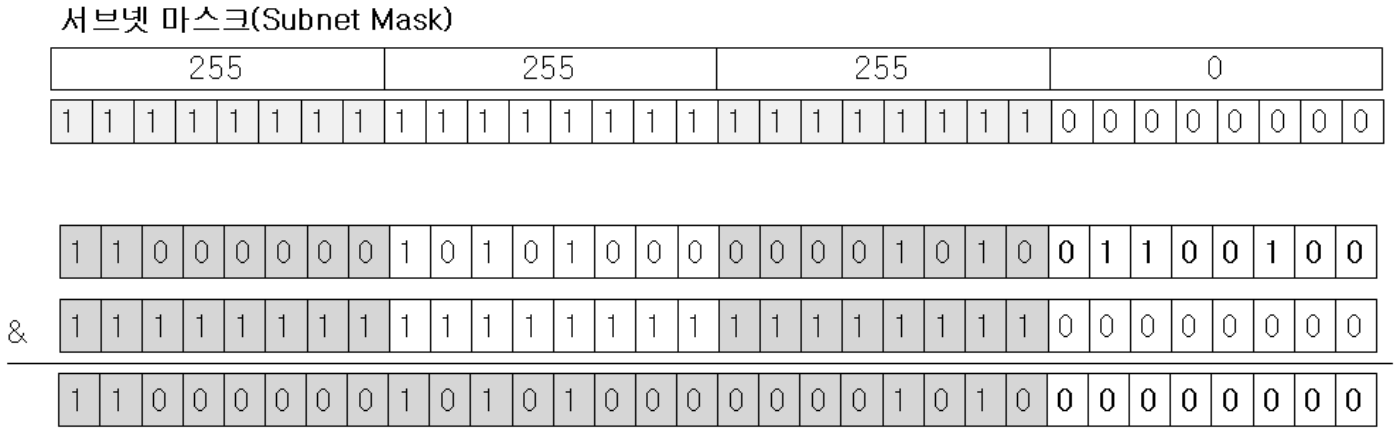
서버기반 모델(server-based model)	P2P 모델(peer-to-peer model)
- 안정적인 서비스의 제공이 가능하다. - 공유 데이터의 관리와 보안이 용이하다. - 서버구축비용과 관리비용이 든다.	- 서버구축 및 운용비용을 절감할 수 있다. - 자원의 활용을 극대화 할 수 있다. - 자원의 관리가 어렵다. - 보안이 취약하다.

1.2 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.



1.3 InetAddress

- IP주소를 다루기 위한 클래스

메서드	설명
byte[] getAddress()	IP주소를 byte배열로 반환한다.
static InetAddress[] getAllByName(String host)	도메인명(host)에 지정된 모든 호스트의 IP주소를 배열에 담아 반환한다.
static InetAddress getByAddress(byte[] addr)	byte배열을 통해 IP주소를 얻는다.
static InetAddress getByName(String host)	도메인명(host)을 통해 IP주소를 얻는다.
String getCanonicalHostName()	FQDN(fully qualified domain name)을 반환한다.
String getHostAddress()	호스트의 IP주소를 반환한다.
String getHostName()	호스트의 이름을 반환한다.
static InetAddress getLocalHost()	지역호스트의 IP주소를 반환한다.
boolean isMulticastAddress()	IP주소가 멀티캐스트 주소인지 알려준다.
boolean isLoopbackAddress()	IP주소가 loopback 주소(127.0.0.1)인지 알려준다.

```
InetAddress ip = InetAddress.getByName("www.naver.com");
```

```
getHostName() :www.naver.com
getHostAddress() :222.122.84.200
toString() :www.naver.com/222.122.84.200
getAddress() :[-34, 122, 84, -56]
getAddress()+256 :222.122.84.200.
```

```
InetAddress ip = InetAddress.getLocalHost();
```

```
getHostName() :mycom
getHostAddress() :192.168.10.100
```

```
InetAddress[] ipArr = InetAddress.getAllByName("www.naver.com");
```

```
ipArr[0] :www.naver.com/222.122.84.200
ipArr[1] :www.naver.com/222.122.84.250
ipArr[2] :www.naver.com/61.247.208.6
```

1.4 URL(Uniform Resource Location)

- 인터넷에 존재하는 서버들의 자원에 접근할 수 있는 주소.

http://www.javachobo.com:80/sample/hello.html?referer=javachobo#index1

프로토콜 : 자원에 접근하기 위해 서버와 통신하는데 사용되는 통신규약(http)
호스트명 : 자원을 제공하는 서버의 이름(www.javachobo.com)
포트번호 : 통신에 사용되는 서버의 포트번호(80)
경로명 : 접근하려는 자원이 저장된 서버상의 위치(/sample/)
파일명 : 접근하려는 자원의 이름(hello.html)
쿼리(query) : URL에서 '?'이후의 부분(referer=javachobo)
참조(anchor) : URL에서 '#'이후의 부분(index1)

```
URL url = new URL("http://www.javachobo.com/sample/hello.html");
URL url = new URL("www.javachobo.com", "/sample/hello.html");
URL url = new  URL("http","www.javachobo.com",80,"/sample/hello.html");
```

```
url.getAuthority():www.javachobo.com:80
url.getContent():sun.net.www.protocol.http.HttpURLConnection$HttpInputStream@c17164
url.getDefaultPort():80
url.getPort():80
url.getFile():/sample/hello.html?referer=javachobo
url.getHost():www.javachobo.com
url.getPath():/sample/hello.html
url.getProtocol():http
url.getQuery():referer=javachobo
url.getRef():index1
url.getUserInfo():null
url.toExternalForm():http://www.javachobo.com:80/sample/hello.html?referer=javachobo#index1
url.toURI():http://www.javachobo.com:80/sample/hello.html?referer=javachobo#index1
```

1.5 URLConnection(1/4)

- 어플리케이션과 URL간의 통신연결을 위한 추상클래스

메서드	설명
void addRequestProperty(String key, String value)	지정된 키와 값을 RequestProperty에 추가한다. 기존에 같은 키가 있어도 값을 덮어쓰지 않는다.
void connect()	URL에 지정된 자원에 대한 통신연결을 연다.
boolean getAllowUserInteraction()	UserInteraction의 허용여부를 반환한다.
int getConnectTimeout()	연결종료시간을 천분의 일초로 반환한다.
Object getContent()	content객체를 반환한다.
Object getContent(Class[] classes)	content객체를 반환한다.
String getContentEncoding()	content의 인코딩을 반환한다.
int getContentLength()	content의 크기를 반환한다.
String getContentType()	content의 type을 반환한다.
long getDate()	헤더(header)의 date필드의 값을 반환한다.
boolean getDefaultAllowUserInteraction()	defaultAllowUserInteraction의 값을 반환한다.
String getDefaultRequestProperty(String key)	RequestProperty에서 지정된 키의 디폴트 값을 얻는다.
boolean getDefaultUseCaches()	useCache의 디폴트 값을 얻는다.
boolean getDoInput()	doInput필드값을 얻는다.
boolean getDoOutput()	doOutput필드값을 얻는다.
long getExpiration()	자원(URL)의 만료일자를 얻는다.(천분의 일초단위)
FileNameMap getFileNameMap()	FileNameMap(mimetable)을 반환한다.
String getHeaderField(int n)	헤더의 n번째 필드를 읽어온다.
String getHeaderField(String name)	헤더에서 지정된 이름의 필드를 읽어온다.
long getHeaderFieldDate(String name,long Default)	지정된 필드의 값을 날짜값으로 변환하여 반환한다. 필드값이 유효하지 않을 경우 Default값을 반환한다.

1.5 URLConnection(2/4)

메서드	설명
int getHeaderFieldInt(String name,int Default)	지정된 필드의 값을 정수값으로 변환하여 반환한다. 필드값이 유효하지 않을 경우 Default값을 반환한다.
String getHeaderFieldKey(int n)	헤더의 n번째 필드를 읽어온다.
Map getHeaderFields()	헤더의 모든 필드와 값이 저장된 Map을 반환한다.
long getIfModifiedSince()	ifModifiedSince(변경여부)필드의 값을 반환한다.
InputStream getInputStream()	URLConnetion에서 InputStream을 반환한다.
long getLastModified()	LastModified(최종변경일)필드의 값을 반환한다.
OutputStream getOutputStream()	URLConnetion에서 OutputStream을 반환한다.
Permission getPermission()	Permission(허용권한)을 반환한다.
int getReadTimeout()	읽기제한시간의 값을 반환한다.(천분의 일초)
Map getRequestProperties()	RequestProperties에 저장된 (키, 값)을 Map으로 반환한다.
String getRequestProperty(String key)	RequestProperty에서 지정된 키의 값을 반환한다.
URL getURL()	URLConnection의 URL의 반환한다.
boolean getUseCaches()	캐쉬의 사용여부를 반환한다.
String guessContentTypeFromName(String fname)	지정된 파일(fname)의 content-type을 추측하여 반환한다.
String guessContentTypeFromStream(InputStream is)	지정된 입력스트림(is)의 content-type을 추측하여 반환한다.
void setAllowUserInteraction(boolean allowuserinteraction)	UserInteraction의 허용여부를 설정한다.
void setConnectTimeout(int timeout)	연결종료시간을 설정한다.
void setContentHandlerFactory(ContentHandlerFactory fac)	ContentHandlerFactory를 설정한다.
void setDefaultAllowUserInteraction(boolean defaultallowuserinteraction)	UserInteraction허용여부의 기본값을 설정한다.
void setDefaultRequestProperty(String key, String value)	RequestProperty의 기본 키쌍(key-pair)을 설정한다.
void setDefaultUseCaches(boolean defaultusecaches)	캐쉬 사용여부의 기본값을 설정한다.
void setDoInput(boolean doinput)	DoInput필드의 값을 설정한다.
void setDoOutput(boolean dooutput)	DoOutput필드의 값을 설정한다.
void setFileNameMap(FileNameMap map)	FileNameMap을 설정한다.

1.5 URLConnection(3/4)

메서드	설명
void setIfModifiedSince(long ifmodifiedsince)	ModifiedSince필드의 값을 설정한다.
void setReadTimeout(int timeout)	읽기제한시간을 설정한다.(천분의 일초)
void setRequestProperty(String key, String value)	ReqeustProperty에 (key, value)를 저장한다.
void setUseCaches(boolean usecaches)	캐쉬의 사용여부를 설정한다.

```
conn.toString():sun.net.www.protocol.http.HttpURLConnection:http://www.javachobo.com/sample/hello.html
getAllowUserInteraction():false
getConnectTimeout():0
getContent():sun.net.www.protocol.http.HttpURLConnection$HttpInputStream@61de33
getContentEncoding():null
getContentLength():174
getContentType():text/html
getDate():1189338850000
getDefaultAllowUserInteraction():false
getDefaultUseCaches():true
getDoInput():true
getDoOutput():false
getExpiration():0
getHeaderFields():{Content-Length=[174],      Connection=[Keep-Alive],      ETag=["12e391-ae-46dad401"],
Date=[Sun, 09 Sep 2007 11:54:10 GMT], Keep-Alive=[timeout=5, max=60], Accept-Ranges=[bytes], Server=[RC-Web
Server], Content-Type=[text/html], null=[HTTP/1.1 200 OK], Last-Modified=[Sun, 02 Sep 2007 15:17:21 GMT]}
getIfModifiedSince():0
getLastModified():1188746241000
getReadTimeout():0
getURL():http://www.javachobo.com/sample/hello.html
getUseCaches():true
```

1.5 URLConnection(4/4) - 예제

```
import java.net.*;
import java.io.*;

public class NetworkEx4 {
    public static void main(String args[]) {
        URL url = null;
        BufferedReader input = null;
        String address = "http://www.javachobo.com/sample/hello.html";
        String line = "";

        try {
            url = new URL(address);

            input = new BufferedReader(new InputStreamReader(url.openStream()));

            while((line=input.readLine()) !=null) {
                System.out.println(line);
            }
            input.close();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

InputStreamReader(InputStream in, String encoding)
지정된 인코딩을 사용하는 InputStreamReader를 생성한다.

```
URLConnection conn = url.openConnection();
InputStream in = conn.getInputStream();
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
 <HEAD>
  <TITLE>Sample Document</TITLE>
 </HEAD>
 <BODY>
Hello, everybody.
 </BODY>
</HTML>
```

2. 소켓 프로그래밍

2.1 TCP와 UDP

▶ 소켓 프로그래밍이란?

- 소켓을 이용한 통신 프로그래밍을 뜻한다.
- 소켓(socket)이란, 프로세스간의 통신에 사용되는 양쪽 끝단(end point)
- 전화할 때 양쪽에 전화기가 필요한 것처럼, 프로세스간의 통신에서도 양쪽에 소켓이 필요하다.

▶ TCP와 UDP

- TCP/IP프로토콜에 포함된 프로토콜. OSI 7계층의 전송계층에 해당

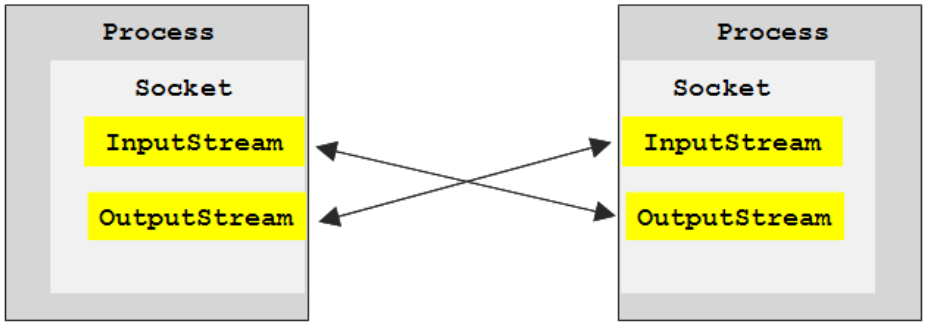
항목	TCP	UDP
연결방식	.연결기반(connection-oriented) - 연결 후 통신(전화기) - 1:1 통신방식	.비연결기반(connectionless-oriented) - 연결없이 통신(소포) - 1:1, 1:n, n:n 통신방식
특징	.데이터의 경계를 구분안함 (byte-stream) .신뢰성 있는 데이터 전송 - 데이터의 전송순서가 보장됨 - 데이터의 수신여부를 확인함 (데이터가 손실되면 재전송됨) - 패킷을 관리할 필요가 없음 .UDP보다 전송속도가 느림	.데이터의 경계를 구분함.(datagram) .신뢰성 없는 데이터 전송 - 데이터의 전송순서가 바뀔 수 있음 - 데이터의 수신여부를 확인안함 (데이터가 손실되어도 알 수 없음) - 패킷을 관리해주어야 함 .TCP보다 전송속도가 빠름
관련 클래스	.Socket .ServerSocket	.DatagramSocket .DatagramPacket .MulticastSocket

2.2 TCP소켓 프로그래밍

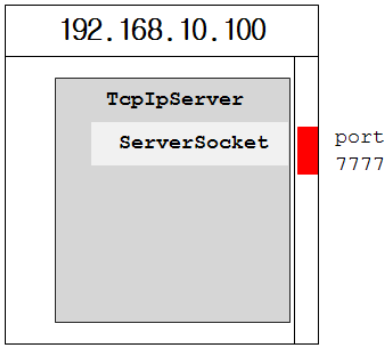
- 클라이언트와 서버간의 1:1 소켓 통신.
- 서버가 먼저 실행되어 클라이언트의 연결요청을 기다리고 있어야 한다.

1. 서버는 서버소켓을 사용해서 서버의 특정포트에서 클라이언트의 연결요청을 처리할 준비를 한다.
2. 클라이언트는 접속할 서버의 IP주소와 포트정보로 소켓을 생성해서 서버에 연결을 요청한다.
3. 서버소켓은 클라이언트의 연결요청을 받으면 서버에 새로운 소켓을 생성해서 클라이언트의 소켓과 연결되도록 한다.
4. 이제 클라이언트의 소켓과 새로 생성된 서버의 소켓은 서버소켓과 관계없이 1:1통신을 한다.

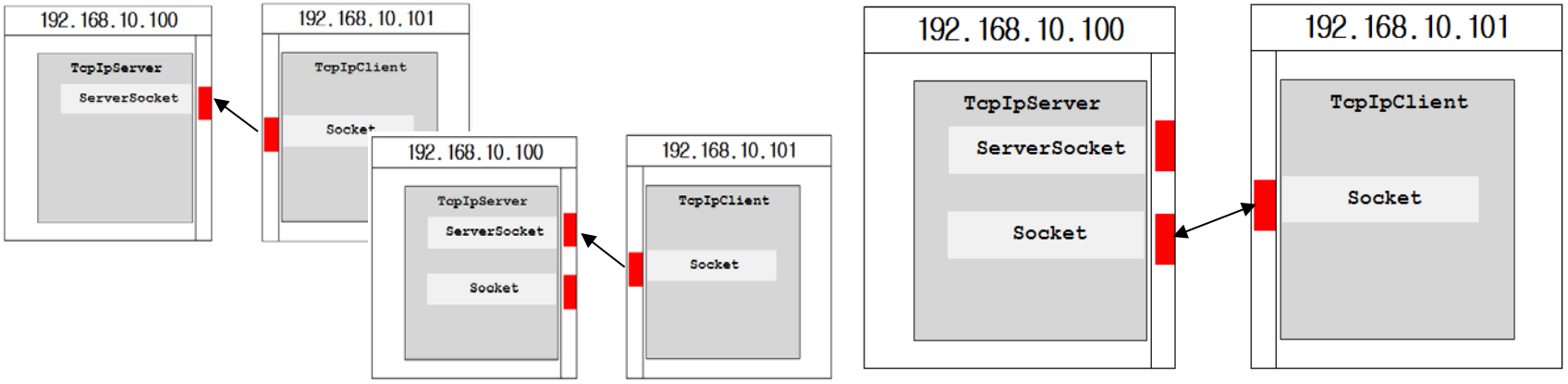
Socket - 프로세스간의 통신을 담당하며, InputStream과 OutputStream을 가지고 있다.
이 두 스트림을 통해 프로세스간의 통신(입출력)이 이루어진다.
ServerSocket - 포트와 연결(bind)되어 외부의 연결요청을 기다리다 연결요청이 들어오면,
Socket을 생성해서 소켓과 소켓간의 통신이 이루어지도록 한다.
한 포트에 하나의 ServerSocket만 연결할 수 있다.
(프로토콜이 다르면 같은 포트를 공유할 수 있다.)



2.2 TCP소켓 프로그래밍 - 예제



1. 서버프로그램을 실행한다.
   > java.exe TcpIpServer
2. 서버소켓을 생성한다.
   serverSocket = new ServerSocket(7777); // TcpIpServer.java
3. 서버소켓이 클라이언트 프로그램의 연결요청을 처리할 수 있도록 대기상태로 만든다. 클라이언트 프로그램의 연결요청이 오면 새로운 소켓을 생성해서 클라이언트 프로그램의 소켓과 연결한다.
   Socket socket = serverSocket.accept(); // TcpIpServer.java
4. 클라이언트 프로그램(TcpIpClient.java)에서 소켓을 생성하여 서버소켓에 연결을 요청한다.
   Socket socket = new Socket("192.168.10.100",7777); // TcpIpClient.java
5. 서버소켓은 클라이언트 프로그램의 연결요청을 받아 새로운 소켓을 생성하여 클라이언트의 소켓과 연결한다.
   Socket socket = serverSocket.accept(); // TcpIpServer.java
6. 새로 생성된 서버의 소켓(서버소켓 아님)은 클라이언트의 소켓과 통신한다.



2.3 UDP소켓 프로그래밍

- TCP소켓 프로그래밍에서는 Socket과 ServerSocket을 사용하지만, UDP소켓 프로그래밍에서는 DatagramSocket과 DatagramPacket을 사용.
- UDP는 연결지향적이지 않으므로 연결요청을 받아줄 서버소켓이 필요없다.
- DatagramSocket간에 데이터(DatagramPacket)를 주고 받는다.

```
DatagramSocket socket = new DatagramSocket(7777);
DatagramPacket inPacket, outPacket;

byte[] inMsg = new byte[10];
byte[] outMsg;

while(true) {
      // 데이터를 수신하기 위한 패킷을 생성한다.
      inPacket = new DatagramPacket(

      // 패킷을 통해 데이터를 수신(receive)한
      socket.receive(inPacket);

      // 수신한 패킷으로 부터 client의 IP주소와 Port를 얻는다.
      InetAddress address = inPacket.getAddress();
      int port = inPacket.getPort();
      ... 중간 생략...
      // 패킷을 생성해서 client에게 전송(send)한다.
      outPacket =  new DatagramPacket(outMsg, outMsg.length, address, port);
      socket.send(outPacket);
}
```

```
DatagramSocket datagramSocket = new DatagramSocket();
InetAddress serverAddress = InetAddress.getByName("127.0.0.1");

byte[] msg = new byte[100]; // 데이터가 저장될 공간으로 byte배열을 생성한다.

DatagramPacket outPacket = new DatagramPacket(msg, 1, serverAddress, 7777);
DatagramPacket inPacket = new DatagramPacket(msg, msg.length);

datagramSocket.send(outPacket);    // DatagramPacket을 전송한다.
datagramSocket.receive(inPacket); // DatagramPacket을 수신한다.

System.out.println("current server time :" + new String(inPacket.getData()));
```