

제29장

네트워크 기초



1. 네트워크 기초

❖ 네트워크

- 여러 대의 컴퓨터를 통신 회선으로 연결한 것
 - 홈 네트워크: 컴퓨터가 방마다 있고, 이들 컴퓨터를 유·무선 등의 통신 회선으로 연결
 - 지역 네트워크: 회사, 건물, 특정 영역에 존재하는 컴퓨터를 통신 회선으로 연결한 것
 - 인터넷: 지역 네트워크를 통신 회선으로 연결한 것

1. 네트워크 기초

❖ 서버와 클라이언트

- 서버 : 서비스를 제공하는 프로그램
 - 웹 서버, FTP서버, DBMS, 메신저 서버
 - 클라이언트의 연결을 수락하고, 요청 내용 처리한 후 응답 보내는 역할
- 클라이언트 : 서비스를 받는 프로그램
 - 웹 브라우저, FTP 클라이언트, 메신저
 - 네트워크 데이터를 필요로 하는 모든 애플리케이션이 해당(모바일 앱 포함)



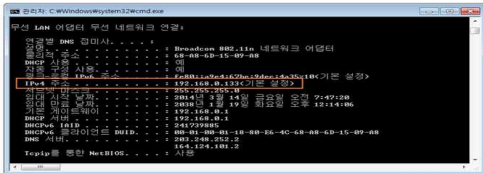
1. 네트워크 기초

❖ IP 주소와 포트(port)

■ IP(Internet Protocol) 주소

- 네트워크상에서 컴퓨터를 식별하는 번호
- 네트워크 어댑터(랜 (Lan) 카드) 마다 할당
- IP 주소 확인 법 - 명령 프롬프트 (cmd.exe) 사용
- xxx.xxx.xxx.xxx 형식으로 표현 (xxx는 0~255 사이의 정수)

C:\W>ipconfig /all



```
관리자: C:\Windows\system32\cmd.exe

무선 LAN 어댑터 무선 네트워크 연결:

   . . . . . : Broadcom 802.11n 네트워크 어댑터
   . . . . . : 68-08-6D-15-09-08
DHCP . . . . . : 예
   . . . . . : 사용 . . . . . : 예
   . . . . . : IP . . . . . : E-80-1a-9c-4-62bc-2dec-14a35-10<기본 설정>
IPv4 주소 . . . . . : 192.168.0.133<기본 설정>
   . . . . . : 255.255.255.0
   . . . . . : 201.4.9월 14일 7:47:28
   . . . . . : 2838년 1월 19일 12:14:06
   . . . . . : 192.168.0.1
   . . . . . : 192.168.0.1
DHCP 서버 . . . . . : 241.729885
DHCPv6 IAD . . . . . : 08-01-00-01-10-00-E6-4C-68-08-6D-15-09-08
DHCPv6 클라이언트 GUID . . . . . : 203.248.252.2
DNS 서버 . . . . . : 164.124.101.2

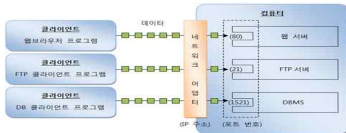
Tcpip를 통한 NetBIOS . . . . . : 사용
```

1. 네트워크 기초

■ 포트(Port)

- **같은 컴퓨터 내에서 프로그램을 식별하는 번호**
- 클라이언트는 서버 연결 요청 시 IP 주소와 Port 같이 제공
- 0~65535 범위의 값을 가짐
- 포트 범위는 세 가지로 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



1. 네트워크 기초

❖ InetAddress로 IP 주소 얻기

■ java.net.InetAddress

- IP 주소 표현한 클래스
- 로컬 컴퓨터의 IP 주소 뿐만 아니라,
- 도메인 이름을 DNS에서 검색한 후 IP 주소를 가져오는 기능 제공

www.naver.com

■ 로컬 컴퓨터에서 얻기

1. 도메인 이름
2. 보통 큰 도메인의 경우 여러 개의 IP로 구성되어 있다.

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

■ 도메인 이름으로 얻기

```
InetAddress ia = InetAddress.getByName(String host);  
InetAddress[] iaArr = InetAddress.getAllByName(String host);
```

■ InetAddress로 IP 주소 얻기

```
String ip = InetAddress.getHostAddress();
```

2. TCP 네트워킹

❖ TCP(Transmission Control Protocol)

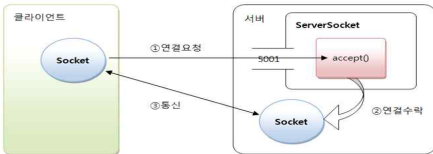
■ 특징

- 연결 지향적 프로토콜 -> 시간 소요(연결부터 먼저 해야 된다.)
- 통신 선로 고정 -> 전송 속도 느려질 수 있음(네트워크 환경에 영향 받는다)
- 데이터를 정확하고 안정적으로 전달하기 때문에 데이터의 누수가 없다.

■ java.net API

- ServerSocket, Socket

■ ServerSocket과 Socket 용도



-- 중요 --

클라이언트의 소켓을 통하여 요청이 들어오면, ServerSocket에서 특정 포트번호로 프로그램을 식별하여 특정프로그램과 바인딩을 하게 된다. 즉, 클라이언트 프로그램과 연동이 된다. 그리고 accept()에서 클라이언트의 연결을 수락하기 위해서 대기한다. 그리고 연결이 수락되어지면 accept()는 Socket객체를 만들어 클라이언트와 통신을 하게 된다.

2. TCP 네트워킹

❖ ServerSocket 생성과 연결 수락(서버 쪽)

- ServerSocket 생성과 포트 바인딩(2가지의 방법)

```
ServerSocket serverSocket = new ServerSocket(5001);
```

 5001은 port넘버이다.

```
ServerSocket serverSocket = new ServerSocket();  
serverSocket.bind( new InetSocketAddress( "127.0.0.1" ,5001) );
```

- 연결 수락

localhost로 적어도 된다.

```
try{  
    Socket socket = serverSocket.accept(); //클라이언트의 요청이 오면 서버는 통신용 소켓을 만듦.  
}catch(Exception e){}
```

- accept() 메소드는 클라이언트가 연결 요청 전까지 블로킹 → 대기(중요)
- 연결된 클라이언트 IP 주소 얻기

```
InetSocketAddress socketAddress = (InetSocketAddress) socket.getRemoteSocketAddress();
```

리터타입	메소드명(매개변수)	설명	getReomoteSocketAddress() 를 통하여 클라이언트의 InetS ocketAddress를 읽어내어 클라 이언트의 IP와 포트번호를 얻어 낼수가 있게 된다.
String	getHostName()	클라이언트 IP 리턴	
int	getPort()	클라이언트 포트 번호 리턴	
String	toString()	"IP:포트번호" 형태의 문자열 리턴	

- ServerSocket 포트 언바인딩

- 더 이상 클라이언트 연결 수락 필요 없는 경우

```
serverSocket.close();
```


2. TCP 네트워킹

❖ Socket 생성과 연결 요청(클라이언트 쪽)

■ Socket 생성 및 연결 요청

- java.net.Socket 이용
- 서버의 IP 주소와 바인딩 포트 번호를 제공하면 생성과 동시에 사용가능

```
try{
    ①번째 방법
    Socket socket = new Socket( "localhost" ,5001); //서버의 IP와 포트번호
    ②번째 방법
    Socket socket = new Socket(new InetSocketAddress( "localhost" , 5001) );
}catch(UnknownHostException e) {
    // IP 표기 방법이 잘못된 경우
}catch(IOException e) {
    //해당 포트의 서버에 연결할 수 없는 경우
}
```

★ 아래는 바로 Socket을 생성 후, connect를 하는 방법이다.

```
socket = new Socket();
Socket.connect( new InetSocketAddress( "localhost" ,5001) );
```

■ 연결 끊기

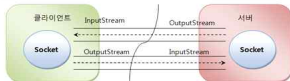
```
try{
    socket.close();
}catch(IOException e) {}
```

2. TCP 네트워킹

❖ Socket 데이터 통신

■ Socket 객체로 부터 입출력 스트림 얻기

```
//입력 스트림 얻기  
InputStream is = socket.getInputStream();  
  
//출력 스트림 얻기  
OutputStream os = socket.getOutputStream();
```



▶ 데이터 보내기

```
String data = "보낼 데이터" ;  
byte[] arr = data.getBytes("UTF-8") ;  
OutputStream os = socket.getOutputStream();  
os.write(arr);  
os.flush();
```

UTF-8 문자셋으로 인코딩함.

▶ 데이터 받기

```
byte[] arr = new byte[100];  
InputStream is = socket.getInputStream();  
int readByte = is.read(arr);  
String data = new String(arr, 0, readByte, "UTF-8") ;
```

UTF-8 문자셋으로 디코딩함.

▶ read()의 블로킹 해제

블로킹이 해제되는 경우	리턴값
상대방이 데이터를 보냄	읽은 바이트 수
상대방이 정상적으로 Socket 의 close()를 호출	-1
상대방이 비정상적으로 종료	IOException 발생

2. TCP 네트워킹

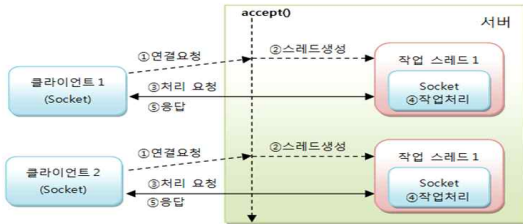
❖ 스레드 병렬 처리

- 블로킹(대기 상태)가 되는 메소드
 - ServerSocket의 accept()
 - Socket 생성자 또는 connect()
 - Socket의 read(), write()
- 병렬 처리의 필요성
 - 스레드가 블로킹되면 다른 작업을 수행하지 못한다.
 - 입출력 할 동안 다른 클라이언트의 연결 요청 수락하지 못한다.
 - 입출력 할 동안 다른 클라이언트의 입출력을 하지 못한다.
 - UI 생성/변경 스레드에서 블로킹 메소드를 호출하지 않도록 한다.
 - UI 생성 및 변경이 안되고 이벤트 처리가 안된다.

2. TCP 네트워킹

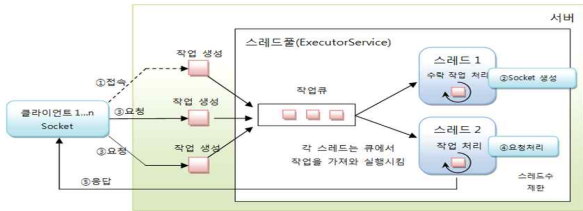
- 스레드 병렬 처리

- `accept()`, `connect()`, `read()`, `write()`는 별도 작업 스레드 생성



2. TCP 네트워킹

■ 스레드풀 사용해 스레드 수 관리



- 스레드풀은 스레드 수를 제한해서 사용하기 때문에 갑작스런 클라이언트 폭증은 작업큐의 작업량만 증가시킬 뿐이지 스레드 수는 변함이 없으므로, 서버 성능은 완만히 저하된다.
- 다만, 대기하는 작업량이 많기 때문에 개별 클라이언트에서 응답을 늦게 받을 수 는 있다.

2. TCP 네트워킹

❖ 채팅 서버 및 클라이언트 구현

■ 서버

- **startServer ()**
 - Executor Service , 서버소켓 생성, 포트 바인딩, 연결수락 코드
- **stopServer()**
 - 연결된 모든 소켓, 서버소켓 닫기, Executor Service 종료
- **클라이언트 클래스**
 - 다수 클라이언트 관리 → 각자 클라이언트 인스턴스 생성해 관리
- **UI 생성 코드 (UI 생성 코드)**

■ 클라이언트

- **startclient()** - 소켓 생성 및 연결요청 코드
- **stopclient()** - 소켓 통신 닫는 기능도 포함
- **receive ()** - 서버에서 보낸 데이터 받을
- **send(String data)** - 사용자가 보낸 메시지 서버로 보냄
- **UI 생성 코드**

3. UDP 네트워킹

❖ UDP(User Datagram Protocol)

■ 특징

• 비연결 지향적 프로토콜

- 연결 절차 거치지 않고 발신자가 일방적으로 데이터 발신하는 방식
- 연결 절차가 없기 때문에 TCP 보다는 빠르게 전송할 수 있다.

• 통신 선로가 고정적이지 않다

- 데이터 패킷들이 서로 다른 통신 선로 통해 전달될 수 있다.
- 먼저 보낸 패킷이 느린 선로 통해 전송될 경우, 나중에 보낸 패킷보다 늦게 도착할 수 있다.

• 데이터 손실 발생 가능성

- 일부 패킷은 잘못된 선로로 전송되어 유실될 수 있다.
- 데이터 전달 신뢰성 떨어짐

■ java.net API

• DatagramSocket, DatagramPacket



3. UDP 네트워킹

■ 발신자 구현 코드

```
DatagramSocket datagramSocket = new DatagramSocket();
```

```
String data = "전달 데이터" ;  
Byte[] byteArr = data.getBytes( "UTF-8" );  
DatagramPacket packet = new DatagramPacket(byteArr, byteArr.length,  
                                             new InetAddress( "localhost" ,5001)  
);
```

```
datagramSocket.send(packet);
```

■ 수신자 구현 코드

```
DatagramSocket datagramSocket = new DatagramSocket(5001);
```

```
DatagramPacket datagramPacket = new DatagramPacket(new byte[100], 100);
```

```
datagramSocket.receive(datagramPacket);
```

■ DatagramSocket닫기

```
datagramSocket.close();
```


감사합니다.

