CHAPTER

15

네트워크

Objectives

- TCP/IP 프로토콜의 개념을 이해한다.
- 자바의 소켓과 포트의 개념을 이해한다.
- 서버 클라이언트 통신 프로그램의 구조를 이해한다.
- 서버 소켓과 클라이언트 소켓을 구분하여 이해한다.
- 간단한 채팅 프로그램 예제를 통해 소켓 통신을 이해한다.
- 수식 계산 서버–클라이언트 통신 예제로 서버–클라이언트 통신을 이해한다.
- 소켓 프로그래밍을 할 수 있다.

주요용어

TCP/IP, IP 주소, 소켓, Socket 클래스, 서버 소켓, ServerSocket 클래스, accept(), close()

CHAPTER 15

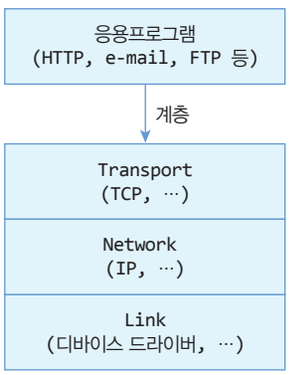
네트워크

15.1 TCP/IP

TCP/IP 프로토콜 소개

TCP 프로토콜
IP 프로토콜

TCP 프로토콜은 Transmission Control Protocol의 약자로 다른 두 시스템 간에 신뢰성 있는 데이터의 전송을 관장하는 통신 프로토콜로서 IP(Internet Protocol) 프로토콜 위에서 동작한다. TCP 프로토콜을 사용하는 응용프로그램으로는 e-mail, FTP, 웹(HTTP) 등이 있다. IP는 패킷 교환 네트워크에서 송신 호스트와 수신 호스트가 데이터를 주고받는 것을 관장하는 프로토콜로서 TCP의 하위 레벨 프로토콜이다. TCP는 IP 기능을 활용하여 두 시스템 사이에 데이터가 손상 없이 안전하게 전송되도록 하며, TCP와 IP를 묶어 TCP/IP로 표기한다. TCP/IP 프로토콜 및 e-mail, 웹 응용프로그램의 관계는 [그림 15-1]과 같다.



[그림 15-1] 네크워크 계층

IP 주소

IP 주소

네트워크 장치의 주소

IP 주소는 네트워크상에서 유일하게 식별될 수 있는 네트워크 장치의 주소로서, 예를 들면 192.156.11.15와 같이 4개의 숫자가 '.'으로 연결된다. 하나의 숫자 범위는 0~255로서 한 바이트로 표현이 가능하다. IP 주소는 마치 전화번호나 집주소와 같아 이 주소를 통해 네트워크에 연결된 장치를 식별할 수 있으며, 동일한 주소를 여러 네트워크 장치에 중복해서 사용할 수 없다. 숫자로 된 주소는 기억하기 어려우므로 www.naver.com과 같은 문자열로 구성된 도메인 이름으로 바꿔 사용한다. 사용자가 문자열로 구성된 도메인 이름을 사용하면 DNS(Domain Name System) 서버에 의해 숫자로 구성된 IP 주소로 자동 변환되게 된다.

현재는 4개의 숫자로 구성된 IP 주소를 표현하기 위해 32비트의 IP 버전 4(IPv4)가 사용되고 있다. 그러나 세계적으로 네트워크 장치의 개수가 폭발적으로 증가하여 각 장치에 고유하게 부여할 수 있는 IP 주소가 고갈됨에 따라 128비트의 IP 버전 6(IPv6)이 점점 사용되는 추세이다.

localhost

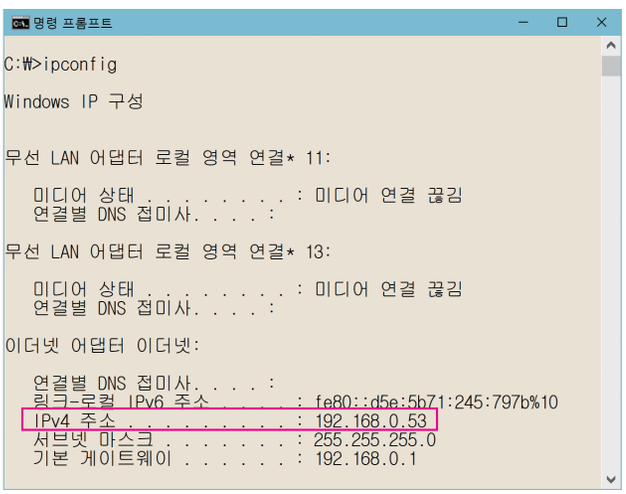
127.0.0.1

자신의 컴퓨터에서 자신의 IP 주소를 간단히 localhost라는 이름으로 사용해도 된다. localhost의 IP 주소는 127.0.0.1로 정해져 있다.

● 내 컴퓨터의 IP 주소 확인하기

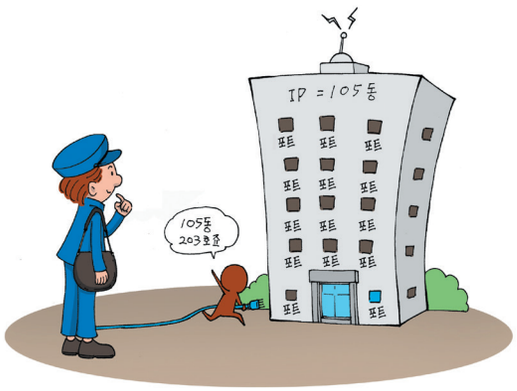
ipconfig

윈도우 PC에서는 명령창을 열어 ipconfig 명령을 수행하면 [그림 15-2]와 같이 컴퓨터의 IP 주소를 확인할 수 있다. 대학이나 연구소 등에서는 한 컴퓨터에 항상 동일한 IP(고정 IP)를 설정하는 경우가 많지만, 가정에서는 대개 무선 공유기가 자동으로 할당해 주는 IP 주소를 부여받는다. 이를 유동 IP라고 부른다.



[그림 15-2] 내 컴퓨터 IP 주소 확인

포트

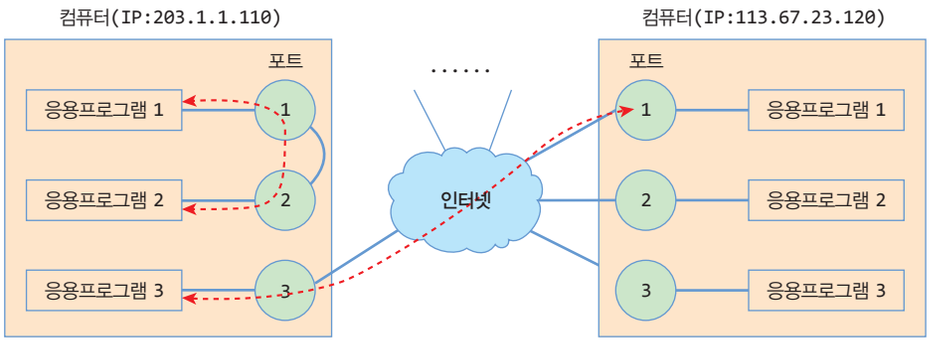


IP 주소는 네트워크상에 있는 한 컴퓨터를 유일하게 식별한다. 하지만, 한 컴퓨터에는 여러 응용프로그램이 네트워크를 사용하고 있기 때문에, IP 주소만 가지고는 통신하고자 하는 응용프로그램을 식별할 수 없다. 이를 위해 한 컴퓨터 내의 각 응용프로그램은 통신을 위해 가상의 연결단인 포트(port)를 생성하고, 이 포트 번호로 상대방이 자신을 식별하게 한다.

IP 주소는 아파트의 동 번호와 같고, 포트 번호는 그 동에 있는 호 번호에 비유할 수 있다. 다른 예로 은행의 사례를 들면, IP 주소는 은행 지점의 주소이고, 포트 번호는 은행 내의 고객 창구 번호와 같다.

포트

따라서 통신을 수행하는 모든 응용프로그램은 IP 주소와 포트를 이용하여 상대편 통신 프로그램을 인지하며 데이터를 교환한다. 물론 이때 상대편 응용프로그램은 자신의 IP 주소와 포트 번호를 알고 통신 접속이나 데이터가 오기를 기다리고 있어야 한다. 응용프로그램과 포트 사용 사례는 [그림 15-3]과 같다.



[그림 15-3] 포트를 이용한 두 응용프로그램의 통신

잘 알려진 포트

포트 번호는 응용프로그램 개발자가 임의로 선택하여 사용할 수 있으나, 기존 응용프로그램에서 사용하고 있는 포트 번호나 시스템의 포트 번호는 피하는 것이 좋다. 시스템이나 기존에 알려진 응용프로그램에서 사용하는 포트 번호를 잘 알려진 포트(well-known ports)라고 한다. 예를 들어, SSH는 22번 포트, HTTP는 80번 포트, FTP는 21번 포트 등이며, 이들은 주로 0~1023 사이의 번호를 가지므로 사용자가 작성하는 응용프로그램에서는 이 범위의 포트 번호는 피해서 선택하도록 한다.

1. IP 주소와 포트에 대해 설명하라.
2. 현재 자신의 컴퓨터의 IP 주소가 얼마인지 확인하라.
3. 한 컴퓨터에서 2개의 통신 응용프로그램이 동일한 포트 번호를 사용한다면 어떤 문제가 발생할 수 있는가?
4. 하나의 통신 응용프로그램은 반드시 하나의 포트만 사용하여야 하는가?

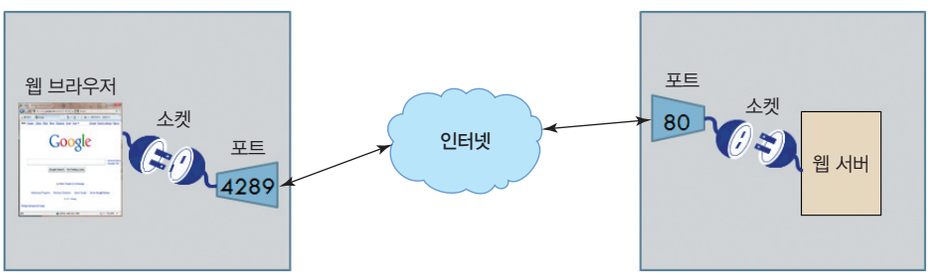
15.2 소켓 프로그래밍

소켓(socket)

소켓 통신

소켓

소켓 통신은 개발자가 TCP/IP 네트워크를 이용하여 쉽게 통신 프로그램을 작성하도록 지원하는 기반 기술이다. 여기서 소켓은 통신하는 두 응용프로그램 간의 통신 링크의 각 끝단(endpoint)으로서, TCP/IP의 네트워크 기능을 활용하여 다른 컴퓨터의 소켓과 데이터를 주고받는다. 소켓을 활용하는 통신의 모양은 [그림 15-4]와 같으며, 소켓은 특정 포트에 연결되어 데이터를 보내거나 받을 때 해당 응용프로그램을 식별한다.



[그림 15-4] 소켓을 이용하는 통신 사례

응용프로그램은 소켓과 연결한 후 소켓에 데이터를 주기만 하면, 소켓이 상대방 응용프로그램에 연결된 소켓에 데이터를 보낸다. 또는 응용프로그램은 연결된 소켓으로부터 도착한 데이터를 단순히 받기만 하면 된다. 인터넷을 경유하여 데이터를 주고받는 기능은 순전히 소켓의 몫이다. 데이터를 주고받는 동안 전송받은 데이터에 오류가 없는 지 검사하고, 만일 손상된 데이터가 오면 다시 받기를 요청하는 등 온전한 데이터를 받는 과정은 모두 소켓의 몫이다.

소켓과 서버 클라이언트 통신

서버
클라이언트

소켓을 이용하는 통신에서는 반드시 서버 응용프로그램과 클라이언트 응용프로그램으로 구분된다. 정보를 제공하는 쪽을 서버(server)라고 부르며, 정보를 이용하는 쪽을 클라이언트(client)라고 부른다. 통신은 서버가 먼저 클라이언트의 접속을 기다리고, 클라이언트에서 서버에 접속하면, 그 때부터 서버나 클라이언트가 데이터를 서로 주고받을 수 있다. 서버나 클라이언트가 보내는 순서를 정하거나 순서에 상관없이 데이터를 전송하는 것은 개발자가 프로그램을 작성하기에 달려 있다.

● 서버 소켓과 클라이언트 소켓

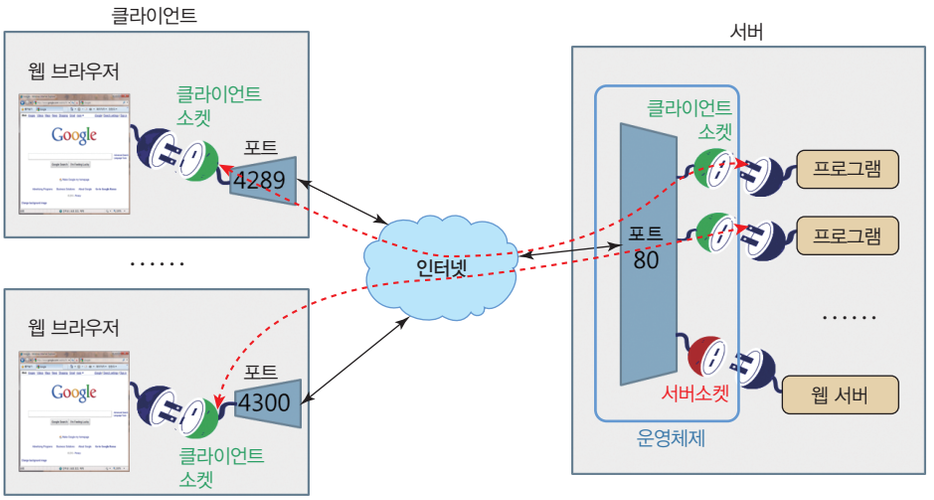
서버 소켓
클라이언트 소켓

소켓에는 서버 소켓과 클라이언트 소켓의 2가지 종류가 있다. 이 둘은 다음과 같이 용도가 서로 다르다.

- 서버 소켓은 서버 응용프로그램이 사용자의 접속을 기다리는(listen) 목적으로만 사용된다.
- 클라이언트 응용프로그램에서는 클라이언트 소켓을 이용하여 서버에 접속한다.
- 서버 소켓은 클라이언트가 접속해오면, 클라이언트 소켓을 추가로 만들어 상대 클라이언트와 통신하게 한다.

이 내용을 정리하면 서버 소켓은 클라이언트의 접속을 기다리는 소켓이며, 클라이언트 소켓은 데이터 통신을 실시하는 소켓이다.

[그림 15-5]는 소켓을 이용한 전형적인 웹 서버 클라이언트 통신 프로그램의 구조와 함께, 클라이언트 응용프로그램과 클라이언트 소켓, 그리고 서버 응용프로그램에서 서버 소켓, 클라이언트 소켓의 관계를 보여준다.



[그림 15-5] 소켓을 이용한 웹 서버와 웹 클라이언트 사이의 통신

● 서버에서 클라이언트 소켓들의 포트 공유

포트 공유
운영체제

[그림 15-5]에서 서버 쪽의 통신프로그램은 각각 독립된 소켓을 이용하여 클라이언트와 통신을 수행한다. 한편, 그림의 서버 쪽을 자세히 들여다보면, '동일한 포트(80)를 여러 클라이언트 소켓들이 공유하고 있으면, 여러 클라이언트들로부터 전송받은 데이터를 서버 내 어떤 소켓으로 전달해야 하는지 어떻게 판단할까?' 하는 의문이 들 것이다. 이것은 운영체제에 의해 처리된다. 클라이언트가 처음 서버 소켓에 연결될 때, 운영체제는 연결된 클라이언트 IP 주소와 포트 번호를 저장하고 기억해둔다. 그 후 서버 컴퓨터의 운영체제는 클라이언트로부터 데이터 패킷을 받게 되면, 패킷 속에 들어 있는 클라이언트의 IP 주소와 포트 번호를 참고하여, 서버에 있는 클라이언트 소켓을 찾아 그 곳으로 데이터를 보낸다.

● 소켓을 이용한 서버 클라이언트 통신 프로그램 구성

자바로 작성하는 서버 응용프로그램과 클라이언트 응용프로그램의 구조를 전체적으로 알아보자. 구체적인 코드들은 다음 절에서 설명한다. 서버 클라이언트의 전형적인 구조는 [그림 15-6]과 같으며 동작하는 과정은 다음과 같다.

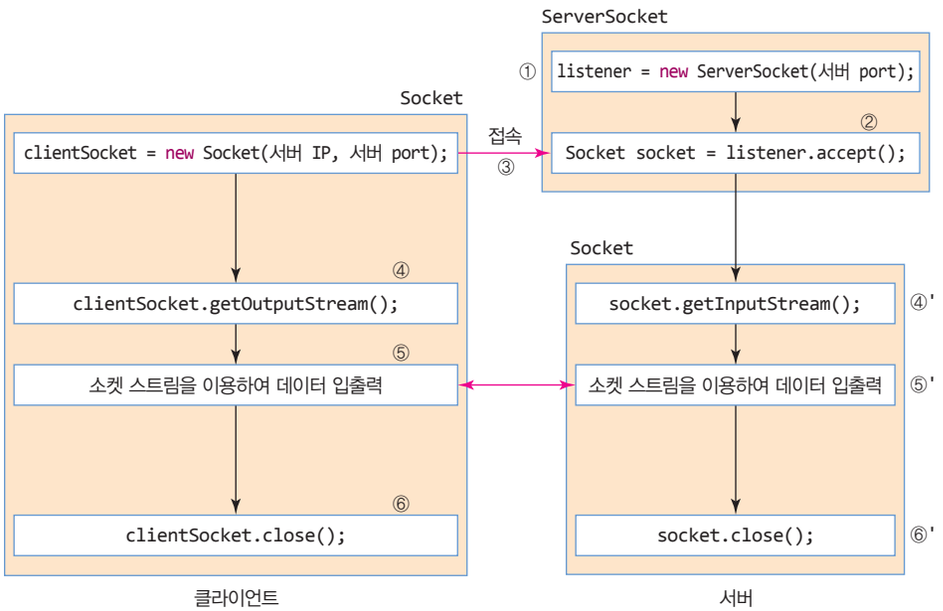
서버 소켓 객체
접속 기다린다

1. 서버 응용프로그램은 ServerSocket 클래스를 이용하여 서버 소켓 객체를 생성하고(new SeverSocket(서버 port)①) 클라이언트의 접속을 받기 위해 기다린다(②). 서버 소켓을 생성할 때 포트 번호를 주어 해당 포트로 접속해 오는 클라이언트 기다리게 한다.
2. 클라이언트 응용프로그램은 Socket 클래스를 이용하여 클라이언트 소켓 객체를

서버에 접속
클라이언트 소켓을 따로 생성
소켓을 닫는다

생성하고(new Socket(서버 IP, 서버 port)) 서버에 접속을 시도한다(③). 소켓 객체를 생성할 때, 접속할 서버 소켓의 IP 주소와 포트 번호를 지정한다.

3. 서버는 클라이언트로부터 접속 요청을 받으면, accept() 메소드에서 접속된 클라이언트와 통신하도록 전용 클라이언트 소켓을 따로 생성한다.
4. 서버와 클라이언트 모두 소켓으로부터 입출력 스트림을 얻어내고 데이터를 주고받을 준비를 한다(④, ④').
5. 서버에 생성된 클라이언트 전용 소켓과 클라이언트의 소켓이 상호 연결된 채 스트림을 이용하여 양방향으로 데이터를 주고받는다(⑤, ⑤').
6. 서버는 클라이언트가 접속해 올 때마다 accept() 메소드에서 따로 전용 클라이언트 소켓을 생성하여 클라이언트와 통신하도록 한다. 통신이 끝나면 소켓을 닫는다(⑥, ⑥').



[그림 15-6] 소켓을 이용한 서버 클라이언트 통신 프로그램의 전형적인 구조

Socket 클래스, 클라이언트 소켓

Socket
클라이언트 소켓

Socket은 java.net 패키지에 포함되어 있는 클래스로서 클라이언트 소켓을 구현한다. 즉, 서버와 통신하기 위해 클라이언트 응용프로그램에서 사용하는 소켓이다. Socket의 생성자와 메소드는 각각 〈표 15-1〉, 〈표 15-2〉와 같다.

Socket의 생성자는 연결할 서버의 IP 주소(또는 도메인 주소)와 포트 번호를 인자로 받아서 Socket 객체를 생성한다. 클라이언트 자신의 주소와 포트 번호가 아님에 주

의하라. 이제 Socket 클래스를 이용하여 클라이언트 응용프로그램을 작성하는 방법을 하나씩 알아보자.

● 클라이언트 소켓 생성 및 서버 접속

IP 주소가 128.12.1.1이고 포트 번호가 5550인 서버에 연결하기 위해 다음과 같이 클라이언트 소켓 객체를 생성한다. 이때 클라이언트의 포트(local port)는 사용되지 않는 포트 중에서 자동으로 선택된다.

```
Socket clientSocket = new Socket("128.12.1.1", 5550); // 128.12.1.1 서버에 접속
```

Socket 객체가 생성되면 곧바로 128.12.1.1 주소의 5550번 포트로 자동 접속이 이루어진다. 다음과 같이 빈 소켓 객체를 생성하고 서버에 접속해도 된다.

```
Socket clientSocket = new Socket(); // 연결되지 않는 소켓 생성
clientSocket.bind(new InetSocketAddress("192.168.1.21", 1234));
                    // 소켓에 자신의 IP 주소(192.168.1.21)와 로컬 포트(1234)를 결합한다.
clientSocket.connect(new InetSocketAddress("128.12.1.1", 5550));
                      // IP 주소가 128.12.1.1이고 포트가 5550인 서버 응용프로그램에 접속
```

● 네트워크 입출력 스트림 생성

소켓이 만들어지고 서버와 연결이 된 후에는 Socket 클래스의 getInputStream()과 getOutputStream() 메소드를 이용하여 서버와 데이터를 주고받을 소켓 스트림을 얻어내고 이를 버퍼 스트림에 연결한다.

getInputStream()
getOutputStream()

```
BufferedReader in = new BufferedReader(new
                        InputStreamReader(clientSocket.getInputStream()));
BufferedWriter out = new BufferedWriter(new
                        OutputStreamWriter(clientSocket.getOutputStream()));
```

지금부터는 in, out 스트림 객체를 이용하여 네트워크 데이터를 보내고 받으면 된다. 이 코드에서 만들어진 in, out은 문자만 보내고 받을 수 있는 문자 입출력 스트림이다.

● 서버로 데이터 전송

이제 버퍼 출력 스트림 out을 통해 데이터를 전송해보자. 다음은 "hello" 문자열을 서버로 전송하는 코드이다.

```
out.write("hello"+"\n");
out.flush();
```

out.flush()

"hello"에 "\n"을 덧붙여 보내는 이유는 서버 쪽에서 라인 단위('\n' 문자가 올 때까지 한 번에 읽는)로 수신한다고 가정하였기 때문이다. 스트림 out은 버퍼 입출력 스트림이므로 버퍼가 차기 전까지 데이터를 보내지 않기 때문에 강제로 out.flush()를 호출하여 스트림 속의 데이터를 모두 즉각 전송하도록 하였다.

● 서버로부터 데이터 수신

버퍼 입력 스트림 in을 이용하면 서버로부터 문자 데이터를 수신할 수 있다. 다음은 클라이언트로부터 한 개의 문자를 입력받는 코드이다.

```
int x = in.read(); // 클라이언트로부터 한 개의 문자 수신
```

한 행의 문자열을 입력받는 코드는 다음과 같다.

```
String line = in.readLine(); // 클라이언트로부터 한 행의 문자열 수신
```

in.readLine() 메소드는 '\n' 문자가 올 때까지 계속 읽고 '\n'이 도착하면 그때까지 읽은 문자열을 리턴한다. 이 문자열 속에는 '\n'이 삽입되지 않는다.

● 데이터 송수신 종료

데이터 송수신을 모두 수행하고 소켓 연결을 끊고자 하면 다음과 같이 한다.

```
socket.close();
```

〈표 15-1〉
Socket 클래스의 생성자

생성자	설명
Socket	연결되지 않은 상태의 소켓을 생성
Socket(InetAddress address, int port)	소켓을 생성하고, 지정된 IP 주소(addresss)와 포트 번호(port)에서 대기하는 원격 응용프로그램의 소켓에 연결
Socket(String host, int port)	소켓을 생성하여 지정된 호스트(host)와 포트 번호(port)에 연결한다. 호스트 이름이 null인 경우는 루프백(loopback) 주소로 가정

〈표 15-2〉
Socket 클래스의 주요 메소드

메소드	설명
void bind(SocketAddress bindpoint)	소켓에 로칼 IP 주소와 로컬 포트 지정(결합)
void close()	소켓을 닫는다.
void connect(SocketAddress endpoint)	서버에 연결
InetAddress getInetAddress()	소켓에 연결된 서버 IP 주소 반환
InputStream getInputStream()	소켓의 입력 스트림 반환. 이 스트림을 이용하여 소켓이 상대편으로부터 받은 데이터를 읽을 수 있음
InetAddress getLocalAddress()	소켓의 로컬 주소 반환
int getLocalPort()	소켓의 로컬 포트 번호 반환
int getPort()	소켓에 연결된 서버의 포트 번호 반환
OutputStream getOutputStream()	소켓의 출력 스트림 반환. 이 스트림에 출력하면 소켓이 서버로 데이터 전송
boolean isBound()	소켓이 로컬 주소와 결합되어 있으면 true 반환
boolean isConnected()	소켓이 서버에 연결되어 있으면 true 반환
boolean isClosed()	소켓이 닫혀있으면 true 반환
void setSoTimeout(int timeout)	데이터 읽기 타임아웃 시간 지정. 0이면 타임아웃 해제

ServerSocket 클래스, 서버 소켓

ServerSocket 클래스는 서버 소켓을 구현한다. ServerSocket 클래스는 java.net 패키지에 포함되어 있으며 ServerSocket 클래스의 생성자와 메소드는 각각 〈표 15-3〉, 〈표 15-4〉와 같다. ServerSocket은 클라이언트로부터 연결 요청을 기다리는 목적으로만 사용되며, 서버가 클라이언트의 연결 요청을 수락하면 Socket 객체를 별도로 생성하고, 이 Socket 객체가 클라이언트와 데이터를 주고받는다. ServerSocket은 데이터의 송수신에 사용되지 않는다.

ServerSocket
서버 소켓
연결 요청을 기다리는 목적

● 서버 소켓 생성

ServerSocket 생성자는 포트 번호를 인자로 받아서 ServerSocket 객체를 생성한다. 이 포트 번호는 클라이언트의 접속을 기다릴 자신의 포트 번호이다. 이미 사용 중인 포트 번호를 지정하면 오류가 발생한다. 9999번 포트를 사용하는 서버 소켓을 생성하는 예를 들면 다음과 같다.

```
ServerSocket listener = new ServerSocket(9999);
```

● 클라이언트로부터 접속 대기

accept()

ServerSocket 클래스의 accept() 메소드를 이용하여 클라이언트로부터의 연결 요청을 기다린다. accept() 메소드가 연결을 수락하면 다음과 같이 Socket 객체를 하나 별도로 생성하여 리턴한다.

```
Socket socket = listener.accept();
```

서버에서 지금 접속된 클라이언트와의 데이터 통신은 새로 만들어진 socket을 이용하여 이루어진다. 새로 만들어진 socket은 서버 소켓과 동일하게 9999번 포트를 통해 데이터를 주고받는다.

● 네트워크 입출력 스트림 생성

getInputStream()
getOutputStream()

클라이언트로 데이터를 주고 받기 위한 스트림 객체는, ServerSocket의 accept() 메소드로부터 얻은 socket 객체의 getInputStream()과 getOutputStream() 메소드를 이용하여 얻어낸다. 다음과 같이 소켓 스트림을 버퍼 입출력 스트림에 연결하여 사용한다.

```
BufferedReader in = new BufferedReader(new
                        InputStreamReader(socket.getInputStream()));
BufferedWriter out = new BufferedWriter(new
                        OutputStreamWriter(socket.getOutputStream()));
```

버퍼 입출력 스트림 in, out을 이용하여 클라이언트와 데이터를 주고받으면 된다. in, out은 모두 문자만 입출력하는 스트림이다.

● 클라이언트로부터 데이터 수신

앞서 만들어진 버퍼 스트림 in을 이용하여 클라이언트로부터 문자 데이터를 수신할 수 있다. 다음은 클라이언트로부터 한 개의 문자를 입력받는 코드이다.

```
int x = in.read(); // 클라이언트로부터 한 개의 문자 수신
```

한 행의 문자열을 입력받는 코드는 다음과 같다.

```
String line = in.readLine(); // 클라이언트로부터 한 행의 문자열 수신
```

in.readLine() 메소드는 '\n' 문자가 올 때까지 계속 읽고 '\n'이 도착하면 그때까지 읽은 문자열을 리턴한다. 이 문자열 속에는 '\n'이 삽입되지 않는다. 현재 서버에서 라인 단위로 읽기 때문에 클라이언트에서는 송신하는 데이터의 끝에 '\n'을 덧붙여 보내야 한다.

● 클라이언트로 데이터 전송

앞서 만들어진 문자 버퍼 스트림 out을 통해 클라이언트로 데이터를 전송할 수 있다. 다음은 "Hi!, Client" 문자열을 클라이언트로 전송하는 코드이다.

```
out.write("Hi!, Client"+"\n");
out.flush();
```

"Hi!, Client"에 "\n"을 덧붙여 보내는 이유는 클라이언트 쪽에서 라인 단위('\n' 문자가 올 때까지 한 번에 읽는)로 수신한다고 가정하였기 때문이다. out.flush()를 호출하면 버퍼 스트림 속의 데이터를 모두 즉각 클라이언트로 전송한다.

out.flush()

● 데이터 송수신 종료

데이터 송수신을 모두 수행하고 소켓 연결을 끊고자 하면 다음과 같이 한다.

```
socket.close();
```

● 서버 응용프로그램 종료

더 이상 클라이언트의 접속을 받지 않고 서버 응용프로그램을 종료하고자 하는 경우 다음과 같이 ServerSocket을 종료시킨다.

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

〈표 15-3〉
ServerSocket 클래스의 주요 생성자

생성자	설명
ServerSocket(int port)	지정된 포트 번호(port)와 결합된 소켓 생성

〈표 15-4〉
ServerSocket 클래스의 주요 메소드

메소드	설명
Socket accept()	클라이언트로부터 연결 요청을 기다리다 요청이 들어오면 수락하고 클라이언트와 데이터를 주고받을 새 Socket 객체를 반환
void close()	서버 소켓을 닫는다.
InetAddress getInetAddress()	서버 소켓의 로컬 IP 주소 반환
int getLocalPort()	서버 소켓의 로컬 포트 번호 반환
boolean isBound()	서버 소켓이 로컬 주소와 결합되어 있으면 true 반환
boolean isClosed()	서버 소켓이 닫혀있으면 true 반환
void setSoTimeout(int timeout)	accept()가 대기하는 타임아웃 시간 지정. 0이면 무한정 대기

15.3 서버-클라이언트 채팅 프로그램 만들기

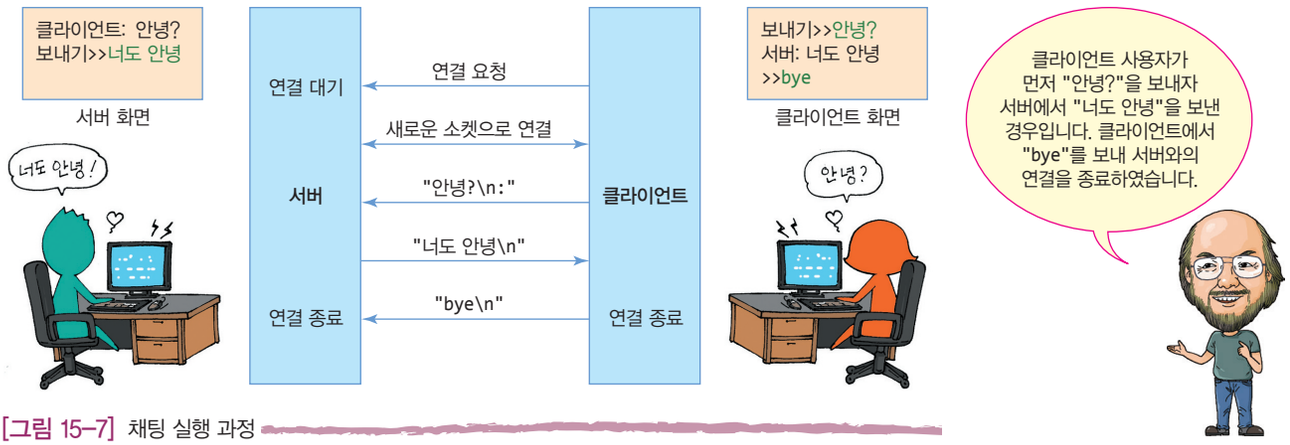
예제 개요

채팅 예제

서버와 클라이언트가 번갈아 한 번씩 채팅 문자를 보내는 간단한 채팅 예제를 만들어 보자. 채팅 프로그램의 개념은 다음과 같다.

- 서버와 클라이언트가 1:1로 채팅한다.
- 클라이언트와 서버가 서로 한번씩 번갈아 가면서 문자열 전송 및 수신한다. 클라이언트가 먼저 문자열을 보내면, 서버가 받아 출력하고 서버가 다시 문자열을 보내는 식이다.
- 서버나 클라이언트는 사용자로부터 문자열을 입력받아 보낸다. 이때 문자열 끝에 "\n"을 덧붙여 보내고 받는 쪽에서는 라인 단위로 수신한다.
- 클라이언트가 "bye"를 보내면 서버 클라이언트 모두 종료한다.

간단한 채팅 프로그램 예제를 통해 자바에서 소켓을 이용하여 어떻게 클라이언트와 서버 간에 데이터를 주고받는 지 실습해보기로 한다. 채팅 프로그램의 실행 과정은 [그림 15-7]과 같다.



[그림 15-7] 채팅 실행 과정

코드 부분 설명

채팅 프로그램의 전체 소스 코드는 다음 절에서 보이며, 여기서는 소스 코드 중 두 부분만 간단히 설명한다. 나머지는 15.2절에서 설명한 바와 거의 같다.

● 클라이언트의 소켓 생성

서버와 클라이언트를 동일한 컴퓨터에서 실행해보고자 한다. 그러므로 클라이언트에서 소켓을 생성할 때, 다음과 같이 서버의 IP 주소 부분을 "localhost"로 하면 된다. 서버가 다른 컴퓨터라면 서버의 IP 주소를 주어야 한다.

```
socket = new Socket("localhost", 9999); // 클라이언트 소켓 생성. 서버의 9999번 포트에 연결
```

● 사용자로부터 라인 입력

다음 절의 소스 코드에는 15.2절에서 설명한 내용에 클라이언트나 서버에서 사용자가 입력한 문자열을 읽어 출력하는 부분이 추가되어 있다. 다음은 서버 쪽에서 사용자가 입력한 문자열을 전송하는 부분 코드이다.

```
Scanner scanner = new Scanner(System.in);
String outputMessage = scanner.nextLine(); // 키보드로부터 한 줄을 읽는다.
out.write(outputMessage + "\n");
out.flush();
```

서버-클라이언트 채팅 소스 코드

전체 예제 코드는 다음과 같다.