第7章　Javaによるネットワークプログラミング

C や Java 言語で TCP/IP による通信を行なうときは _____と呼ばれるものを用いる。ソケットは TCP/IP のポートに対するプログラムからのインターフェースである。

7.1　クライアントのプログラミング

例題 7.1.1 コネクション型（*TCP*）（受信のみ）

ファイル TCP_RO.java

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

public class TCP_RO {
  public static void main(String[] argv) {
    try {
      Socket readSocket = new Socket(argv[0],
                                     Integer.parseInt(argv[1]));
      InputStream instrm = readSocket.getInputStream();
      while(true) {
          int c = instrm.read();
          if (c==-1) break;
          System.out.write(c);
      }
    } catch (Exception e) {
      e.printStackTrace();
      System.exit(1);
    }
  }
}
```

このプログラムは、サーバから *TCP* を用いてデータを受信し画面に出力するだけのプログラムである。

```
java TCP_RO サーバホスト名 ポート番号
```

という形で使用する。*Socket* クラスのオブジェクトを生成するときの引数は、通信先のホスト名（*IP* アドレスでも可）とポート番号である。

この *Socket* オブジェクトから _____メソッドで、*InputStream* オブジェクトを取り出すことができる。この *InputStream* に対しては、標準入力オブジェクト（*System.in*）と同じ方法で入力が可能である。

ただし、通常は

```
    while(true) {
        int c = instrm.read();
        if (c==-1) break;
        System.out.write(c);
    }
```

の部分は、一文字ずつ入出力を行なうことになって効率が悪いので、この部分は

```
    byte[] buff = new byte[1024];
    while(true) {
        int n = instrm.read(buff);
        if (n==-1) break;
        System.out.write(buff, 0, n);
    }
```

のようにバッファを用いて、一度に大量の文字（この場合は 1024 文字）を読むようにするのが普通である。（InputStream クラスの read() メソッド（無引数）は一文字を読む。read(byte[]) メソッドは引数として与えられた配列に文字を一度に読み込み、読み込んだ文字数を返す。）

このプログラムの main メソッドでは全体を try 〜catch で囲んで、エラーが起ったときはメッセージを表示するようにしている。Exception の _____ メソッドはエラーが起った場所の情報を出力する。（本当はもっとちゃんとしたエラー処理を書くべきだが、ここでは簡略にしている。）

例題 7.1.2 コネクション型（*TCP*）（送受信）

ファイル *TCP_RW.java*

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

public class TCP_RW {
  public static void main(String[] argv) {
    byte[] buff = new byte[1024];
    try {
      Socket rwSocket     = new Socket(argv[0],
                                       Integer.parseInt(argv[1]));
      InputStream  instrm = rwSocket.getInputStream();
      OutputStream outstr = rwSocket.getOutputStream();
      while(true) {     // 標準入力からソケットへ
        int n = System.in.read(buff);
        if (n==-1) break;
        outstr.write(buff, 0, n);
      }
      while(true) {     // ソケットから標準出力へ
        int n = instrm.read(buff);
        if (n==-1) break;
        System.out.write(buff, 0, n);
      }
    } catch (Exception e) {
      e.printStackTrace();
      System.exit(1);
    }
  }
}
```

.....

.....

このプログラムはソケットに対して、まず送信を行なってから受信を行なう。送信を行なうために *Socket* クラスの ＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿メソッドを使って、*OutputStream* クラスのオブジェクト *outstr* を得ている。この *outstr* の *write(byte[], int, int)* メソッドを用いて、ソケットに出力する。*write(buff, i, n)* という呼出しは *buff* という配列の *i* 番目から *n* 文字を出力する。

使用法は次のとおりである。

```
java TCP_RW サーバホスト名 ポート番号
```

例えば *80* 番は *HTTP* のポートなので、*Web* サーバがあるマシンに対して、通信を行なうと次のようになる。

```
> java TCP_RW 133.92.XXX.XXX 80↩
GET /index.html HTTP/1.0↩
↩ （※ ← この改行の後に Ctrl-Z または Ctrl-D）
HTTP1.1 200 OK
Date: Mon, XX Xxx 2XXX XX:XX:XX GMT
Server: Apache/X.X
 ...
```

立字体の部分が、ユーザが入力した部分、斜字体の部分がシステムの反応である。入力の終の印として *Ctrl-Z*（*Windows* の場合）, *Ctrl-D*（*Unix* の場合）を入力すると“標準入力からソケットへ”のループを脱出して、“ソケットから標準出力へ”のループに移る。

.....
.....
.....

問 7.1.3 *URL* をコマンドライン引数として受け取って、*HTTP* サーバと交信し、ページをファイルにダウンロードするプログラム *simpleGet* を書け。

```
simpleGet http://133.92.XXX.XXX/index.html
```

とすると *index.html* ファイルをダウンロードする。

問 7.1.4 *URL* をコマンドライン引数として受け取って、*HTTP* サーバと交信し、ページの中のリンク <a href=" ... ">を表示するプログラムを書け。（ヒント: *Java.lang.String* クラス[1]のメソッドを利用せよ。また、*2* 行にまたがる場合やコメントに含まれている場合などを完全に考慮すると難しくなるので、*100%*完全なプログラムでなくても良い。）

telnet や ftp と交信するときには、パスワードを入力する必要がある。通常 telnet や ftp でパスワードを入力するときは、セキュリティのため、パスワードが画面に現れない（エコーしない）ようになっている。しかし、Java には標準入出力のエコーを止めるための方法が用意されていない（ようである）。ちなみに C 言語では、`stty` あるいは `ioctl` という関数を用いる。そこで、ftp や telnet などパスワードを入力するプログラムを作るときは、その時だけウインドウを作成して、パスワードを入力することにする。

[1] *(JDKDIR)*/docs/api/java/lang/String.html 参照

例題 7.1.5　（参考）パスワードを入力するためのクラス

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

class MyDialog extends JFrame implements ActionListener {
  JPasswordField t;
  String ret;
  boolean suspended;

  private String ShowDialogAux(String message, int len, boolean echo) {
    getContentPane().setLayout(new FlowLayout());
    getContentPane().add(new Label(message));

    t = new JPasswordField("", len);
    t.addActionListener(this);
    if (!echo) {/* echo が false なら画面にエコーしない */
      t.setEchoChar('*');
    }
    getContentPane().add(t);

    JButton b = new JButton("OK");
    b.addActionListener(this);
    getContentPane().add(b);
    pack();     // 画面に表示する

    suspended = true;
    show();
    try {
      synchronized(this) {
        while(suspended) { // OK ボタンが押されるのを待つ
          wait();
        }
      }
    } catch (InterruptedException e) {}
    dispose();  // 画面から消える
    return ret;
  }

  public void actionPerformed(ActionEvent ae) {
    ret = t.getText(); // 万全を期すなら getPassword() を用いる
    suspended = false;
    synchronized(this) {
      notify(); // ShowDialogAux メソッドで待っているのを起こす
    }
  }

  public static String ShowDialog(String message, int len, boolean echo) {
    MyDialog my = new MyDialog();
    return my.ShowDialogAux(message, len, echo);
  }
}
```

次のプログラムはこの *MyDialog* クラスの使用例である。

ファイル *MyDialogTest.java*

```
 ...
public class MyDialogTest {
  // MyDialog クラスの使用例
  public static void main(String[] argv) {
    String result1 = MyDialog.ShowDialog("login:", 8, true);
    String result2 = MyDialog.ShowDialog("password:", 8, false);
    System.out.println(result1);
    if (result1.equals(result2)) {
      System.out.println("OK!");
    }
    System.exit(0);
  }
}
```

文字を入力するためのダイアログは、*MyDialog.ShowDialog(String, int, boolean)* という形で使用する。第 *1* 引数は入力を促すメッセージ、*2* 番目は入力するための *TextField* のに入力可能な文字数、*3* 番目は入力をエコーするかどうか（*true* — エコーする、*false* — 伏せ字にする）である。

この例では、*2* 番目に現れる *"password:"* を入力するためのダイアログでは、入力は、伏せ字（*'*'*）になる。

問 7.1.6 *simpleChmod* モード パスという形で実行すると、*FTP* サーバと交信して、ファイルを *chmod* するプログラム *simpleChmod* を書け。

実行例）

```
java simpleChmod 660 ftp://stfile/home/Report/ ...
```

問 7.1.7 *Tenso* 転送元 転送先という形で実行すると、転送元のローカルファイルを *FTP* サーバ上の転送先に転送するプログラム *Tenso* を書け。（ヒント: *Java.io.File* クラス[2]のメソッドを利用せよ。）

問 7.1.8 *Tenso* をさらにディレクトリ構造をコピーできるようにせよ。

7.2 スレッドを用いた複数の入出力への対処

ソケットを用いたプログラムでは、標準入力とソケットからの入力など複数の入力を待ち受ける必要がある場合が必然的に多くなる。このような場合は、ある入力を待ち受けるためにブロック（ストップ）してしまって、他の入力があるのにそれに反応できない、という状況は避けなくてはならない。

これに対処する方法として考えられるのが、入力があるかどうかいちいち調べる方法（＿＿＿＿＿＿＿＿）である。（Java の場合、InputStream クラスの ＿＿＿＿＿＿＿＿というメソッドを用いる）

```
while (true) {
   if (input1.available() > 0) {
     ...  // input1 に対する処理
   } else if (input2.available() > 0) {
     ...  // input2 に対する処理
   } ...
}
```

[2] *(JDKDIR)*/docs/api/java/io/File.html 参照

この方法は、＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿ので、望ましくない。

Java では次の例のようにスレッドを使って複数の入力を待ち受けることができる。

例題 7.2.1 スレッドを使った例

ファイル *TCPThread.java*

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

public class TCPThread {
  public static void main(String[] argv) {
    try {
      Socket rwSocket   = new Socket(argv[0], Integer.parseInt(argv[1]));
      InputStream  instrm = rwSocket.getInputStream();
      OutputStream outstr = rwSocket.getOutputStream();

      Thread input_thread  = new Thread(new StreamConnector(System.in, outstr));
      Thread output_thread = new Thread(new StreamConnector(instrm, System.out));
      input_thread.start(); output_thread.start();
    } catch (Exception e) {
      e.printStackTrace();
      System.exit(1);
    }
  }
}
```

このクラスでは *StreamConnector* という補助的なクラスを定義している。*StreamConnector* の＿＿＿メソッドがスレッドで実行される。独立したスレッドの中で入力を待つので、標準入力を待っている状態でも、ソケットからの入力に対応することができる。

ファイル *TCPThread.java*（続き）

```
class StreamConnector implements Runnable {
  InputStream src  = null;
  OutputStream dist = null;
    // コンストラクタ 入出力ストリームを受け取る
  public StreamConnector(InputStream in, OutputStream out){
    src = in;
    dist = out;
  }

  // 処理の本体
  // ストリームの読み書きを無限に繰り返す
  public void run(){
    byte[] buff = new byte[1024];
    while (true) {
      try {
        int n = src.read(buff);
        if (n > 0)
          dist.write(buff, 0, n);
      }
      catch(Exception e){
        e.printStackTrace(System.err);
        System.exit(1);
      }
    }
  }
}
```

使用法は、

`java TCPThread` サーバホスト名 ポート番号

である。

問 7.2.2 複数の *HTTP* サーバに同時接続してファイルをダウンロードし、さらにユーザから新規接続の要求も受け取るプログラムを作成せよ。

7.3 サーバのプログラミング

HTTP サーバや Telenet サーバなどのサーバは多数のクライアントからの接続を受け付けなければならないので、クライアント側とは異なる形でソケットを利用する。Java では _____ というクラスを用いる。

例題 7.3.1 コネクション型（*TCP*）（サーバ側）

ファイル *Pphttpd.java*

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

public class Pphttpd{
  public static void main(String args[]){
    // サーバソケットの準備
    ServerSocket servsock = null ;
    Socket sock ;

    BufferedReader in;
    // println メソッドが使えるように PrintStream クラスを用いる
    PrintStream out;

    try {
      // サーバ用ソケットの作成
      servsock = new ServerSocket(Integer.parseInt(args[0]));
      while(true){
        sock = servsock.accept(); // 接続要求の受付
        // 以下の処理は、時間がかかる場合は、
        // 本来はすぐに接続要求の受付に戻れるように、スレッドで行なうべきである。

        // 接続先の表示
        System.out.println("Request from "
                           + (sock.getInetAddress()).getHostName());
        // 効率を考慮してバッファを利用する。
        //  （1 文字ずつではなく、まとめて読めるようにする。）
        in  = new BufferedReader(
                  new InputStreamReader(sock.getInputStream()));
        out = new PrintStream(sock.getOutputStream());
        // とりあえず改行を 2 つ読み飛ばす
        int i ;
        for(i=0; i<2; ) {
          in.readLine();
        }
        out.println("<HTML>");
        out.println("<HEAD><TITLE>Test</TITLE></HEAD>");
        out.println("<BODY>Hello!</BODY>");
        out.println("</HTML>") ;
        // 接続終了
        sock.close() ;
      }
    } catch (Exception e){
      e.printStackTrace();
      System.exit(1) ;
    }
  }
}
```

.....

.....

ServerSocket のコンストラクタの引数はポート番号である。

クライアントからの接続要求の受け付けは、＿＿＿＿＿メソッドで行なう。

```
sock = servsock.accept();
```

このメソッドは新しい *Socket* クラスのインスタンスを返す。クライアントとの通信は、この新しい *Socket* を通じて行なう。*ServerSocket* の方は、次のクライアントからの接続要求のために再び利用する。

このプログラムを例えば、

```
java Pphttpd 8080
```

というように *8080* 番のポートで起動して、*Netscape* などで *URL* を *http://XXX.XXX.XXX.XXX:8080/*（*XXX.XXX.XXX.XXX* の部分は、*Pphttpd* を起動したマシンのホスト名か *IP* アドレス）と指定する。すると *"Hello!"* という内容だけの *Web* ページがあるかのように表示される。

WindowsXP の場合、マシンの *IP* アドレスは *ipconfig* コマンドで調べることができる。また、*IP* アドレス *127.0.0.1* は必ず自分自身を指すので、*Pphttpd* とクライアントを同じマシンで実行する時は、*127.0.0.1* を使うこともできる。

問 7.3.2 接続要求を受け付けると、別の *Web* サーバに要求をそのまま中継して、サーバから受信したデータをそのままクライアントに送るプログラム（超簡易 *proxy* サーバ）を書け。

問 7.3.3 アクセスカウンタ付 *Web* ページを配信する（偽）*HTTP* サーバプログラムを書け。

問 7.3.4 時計付 *Web* ページを配信する（偽）*HTTP* サーバプログラムを書け。

問 7.3.5 （難） オセロや麻雀などのネットワーク対戦型ゲームのサーバとクライアントを作成せよ。

7.4 コネクションレス型のプログラミング

これまで紹介したソケットは _____を使った _____と呼ばれるものである。コネクション型では、最初にソケット間の接続を行ない、通信されるデータの順序が保存されるようになっている。

一方、_____を用いる、最初に接続を行なわない _____のソケットもある。これは、送信のたびに宛先を指定する。コネクションレス型のソケットでは、データの順序は保存されないし、データが失われる場合もある。ただし高速である。

例題 7.4.1 コネクションレス型（*UDP*）（クライアント）

ファイル *UdpClient.java*

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

public class UdpClient {
  public static void main(String[] argv) {
    try {
      // 接続先の IP アドレスとポート番号
      InetAddress addr = InetAddress.getByName(argv[0]);
      int port = Integer.parseInt(argv[1]);

      // 適当な空いているポート番号にソケットを作る
      DatagramSocket dgSock = new DatagramSocket();
      while (true) {
        byte buff1[]= new byte[512];
        int n = System.in.read(buff1);
        // 送信パケットの作成
        DatagramPacket pa1 = new DatagramPacket(buff1, n, addr, port);
        dgSock.send(pa1);  // パケット送出
        System.out.println("Sent!");

        // 受信パケット用データ領域の作成
        byte buff2[] = new byte[512];
        DatagramPacket pa2 = new DatagramPacket(buff2, buff2.length);
        dgSock.receive(pa2); // パケット受信

        System.out.print("received: ");
        System.out.print(new String(pa2.getData()));
      }
    } catch (Exception e) {
      e.printStackTrace();
      System.exit(1);
    }
  }
}
```

.....
.....

使用方法は、

```
java UdpClient サーバホスト名　ポート番号
```

である。

引数なしで *DatagramSocket* クラスのコンストラクタを呼び出すと、適当な空きポートに *UDP* ソケットを作る。また、パケットは *DatagramPacket* というクラスのオブジェクトとして表現される

```
DatagramPacket(byte[] data, int len, InetAddress addr, int port)
```

という形のコンストラクタは、長さ *len* のデータで、宛先の *IP* アドレス *addr*、ポート番号が *port* というパケットのためのデータを用意する。実際にパケットを送るのは *DatagramSocket* クラスの *send(DatagramPacket)* メソッドである。

また、

```
DatagramPacket(byte[] data, int len)
```

という形の 2 引数のコンストラクタは受信したパケットのデータを受け取るためのオブジェクトを用意する。実際にパケットを受信するのは *DatagramSocket* クラスの *receive* メソッドである。*receive* メソッドに、*DatagramPacket* クラスのオブジェクトを引数として与える。*receive* の呼出し後には、このオブジェクトの内容が受信したデータに書き換えられている。

例題 7.4.2 コネクションレス型（*UDP*）（サーバ）

ファイル *UdpServer.java*

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

public class UdpServer {
  public static void main(String[] argv) {
    try {
      // 使用するポート番号
      int port = Integer.parseInt(argv[0]);

      // 指定されたポート番号にソケットを作る
      DatagramSocket dgSock = new DatagramSocket(port);
      while (true) {
        // 受信パケット用データ領域の作成
        byte buff1[]= new byte[512];
        DatagramPacket pa1 = new DatagramPacket(buff1, buff1.length);
        dgSock.receive(pa1);  // パケット受信
        System.out.println("Received!");
        System.out.print(new String(pa1.getData()));
        System.out.println("addr: "+pa1.getAddress());
        System.out.println("port: "+pa1.getPort());

        // 送信パケットの作成
        DatagramPacket pa2 =
          new DatagramPacket(pa1.getData(), pa1.getLength(),
                             pa1.getAddress(), pa1.getPort());
        dgSock.send(pa2);
        System.out.println("Sent!");
      }
    } catch (Exception e) {
      e.printStackTrace();
      System.exit(1);
    }
  }
}
```

.....
.....

サーバ側では、クライアントと違って固定したポート番号にソケットを作成する。そのためポート番号（*int* 型）を *DatagramSocket* のコンストラクタの引数として用いる。

このプログラムは、まず先にクライアントから送られてきたパケットを受信している。受信したパケット（*DatagramPacket*）から *getData* メソッドでデータの部分を取り出すことができる。また *getAddress*, *getPort* メソッドで送り元の *IP* アドレス、ポート番号を知ることができる。

このプログラムでは送られてきたデータを、そのまま何も変更せずにクライアントの送り元のポートに送り返している。

問 7.4.3　（タートルグラフィックスサーバ）
タートルグラフィックスとは、画面上の仮想の亀に指令を与えて、線を描画することである。例えば、次のような指令は一辺が *100* の正三角形を描く。

```
FORWARD 100
RIGHT 120
FORWARD 100
RIGHT 120
FORWARD 100
RIGHT 120
```

FORWARD は前進する命令、*RIGHT* は右に回転する命令である。この “亀” をサーバとして実現して、複数のクライアントから指令を与えることができるようにせよ。サーバとクライアントの間はコネクションレス型で通信を行なうこと。クライアントはサーバから情報を得て、“亀” の軌跡を表示できるようにせよ。（タートルグラフィックスの命令は自由に拡張しても良い。）

キーワード　ソケット、Socket クラス、getInputStream メソッド、 getOutputStream メソッド、ビジーウェイト、スレッド、ServerSocket クラス、accept メソッド、PrintStream クラス、DatagramSocket クラス、DatagramPacket クラス、send メソッド、receive メソッド、getData メソッド、getAddress メソッド、getPort メソッド