ネットワーク・プログラミング

拓殖大学工学部情報工学科

平成 30 年 12 月 12 日

現在のコンピュータシステムの多くは，単一の計算機として独立しているのではなく，研究室，事務室や事業所のネットワーク，あるいは世界的なインターネットの一部として互いに通信できる機能を持っている．そのようなシステムの開発では，従来のプログラミングとは異なった，ネットワークを利用したプログラミングの開発技術が要求される．

1　クライアントサーバ型通信

インターネットで使用されている WorldWideWeb の HTTP(Hyper Text Transfer Protocol) や電子メイルの SMTP(Simple Mail Transfer Protocol) などの通信の基本は，ネットワーク経由で要求を送り，結果を受け取るクライアントと，クライアントからの要求を受け取り，処理結果を送り返すサーバからなるサーバクライアント型の通信である．このサーバクライアント型の通信では，クライアントは要求の送り先として，計算機とその計算機上で要求を処理するサーバのプログラムを指定しなければならない．インターネットの通信プロトコルである TCP/IP では，通信先の計算機の指定は **IP アドレス**と呼ばれる 4 オクテット (1 オクテットは 8 ビット) の計算機ごとに異っているアドレスを使用する．また，計算機上のプログラムの指定には，2 オクテットのポート番号を使用する．IP アドレスとポート番号を使ったサーバプログラムの指定の仕組みは次のとおりで，通信の様子を図 1 に示す．

- 各計算機はポート番号が割当てられた仮想的なポートを持っている．
- サーバプログラムは自分の計算機のポートを開き，メッセージが届くのを待つ
- クライアントは相手先の計算機の IP アドレスとポート番号を指定してメッセージを送ることで，特定のプログラムに要求を送ることができる．

この仕組みがうまく働くためには，クライアントは自分の要求を受け付けるサーバがどの番号のポートで待っているかを知らなければならない．そこで通常は，電子メイルならば 25 番というようにサービスの内容ごとにあらかじめ決めた番号を使用する．

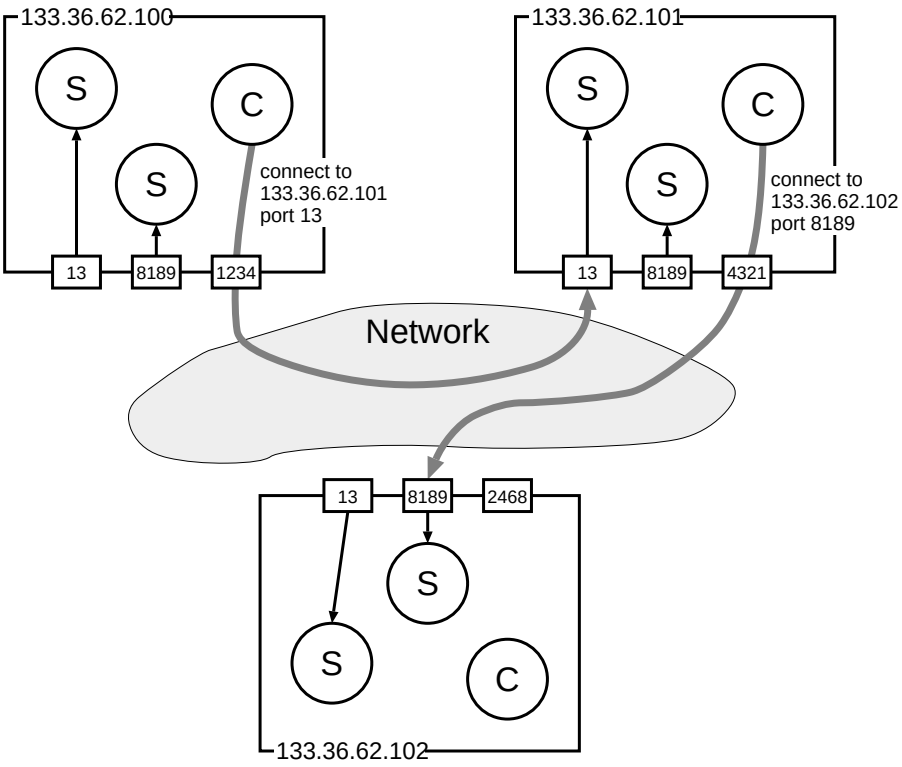

図 1: クライアントサーバ型通信

2　サーバとの接続

2.1　telnet を使った接続

プログラムを作る前に，まずは既存のコマンドを使ってサーバとの接続をやってみよう．UNIX には telnet という他の計算機と通信するためのコマンドがある．telnet の通常の使い方は，相手先計算機の名前 (または IP アドレス) だけを指定し，ポート番号を指定しない．この場合 telnet は通信相手の計算機にログインしてコマンドを実行するためのインターフェース (仲介役) として働く．ポート番号を指定した場合には，相手先計算機のサーバと文字ベースでの通信を行なうことができるので，サーバプログラムの簡単なテストに使うことができる．ポート番号を指定して telnet を実行するには，次のように引数に計算機の名前 (IP アドレスでも可) とポート番号を与える．

telnet 計算機名 ポート番号

具体的な例として例えば，ポート番号 13 は，計算機が持っている現在時刻を知らせるサービスに使われているので，ポート番号を 13 として telnet を実行すると相手先の計算機上での現在時刻が表示される．

課題 1 計算機名を `133.36.50.23`，ポート番号を *13* として *telnet* を実行せよ．

現在時刻のサービスでは接続を行っただけで結果が得られて，接続が切られてしまうが，こちらからの要求を送るような例として World Wide Web の通信を telnet を使ってやってみよう．

課題 2 *WWW* サーバへの接続

1. 計算機名を `133.36.50.23`，ポート番号を *80* として *telnet* を実行せよ．
2. 接続されたら次のとおり正確に入力せよ *(*バックスペースなどを使って修正できない場合がある*)*．

   GET /

3 Java プログラムの作成とコンパイル

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

class SocketTest {
  public static void main(String[] args) {
    try {
      Socket t = new Socket("133.36.50.23", 13);
      BufferedReader is = new BufferedReader(
        new InputStreamReader(t.getInputStream()));
      boolean more = true;
      while ( more ) {
        String str = is.readLine();
        if (str == null)
          more = false;
        else
          System.out.println(str);
      }
    }
    catch ( IOException e ) {
      System.out.println("Error" + e);
    }
  }
}
```

図 2: 簡単なクライアントのプログラム

telnet を使ってやったのと同じことを Java でやってみよう．プログラムの例を図 2 に示す．

課題 3 図 *2* のプログラムのソースファイルを作成し，コンパイルせよ．

3.1 実行手順

図 2 のプログラムをコンパイルすると SocketTest.class というファイルが作成される．これを実行するには，

```
java SocketTest
```

というコマンドを実行する． `java` の引数がクラス名であって，ファイル名でないことに注意すること．

課題 4 図 *2* のプログラムをコンパイルし，実行して動作を確認せよ．

3.2 Java プログラムの説明

このプログラム全体での注意点は次の 2 つ．

- java.net パッケージを import する．
- try/catch で main メソッド内の I/O エラーを全て処理する．

java.net には java でネットワークを扱うためのクラスが提供されている．また，ネットワーク接続に関するメソッドのほとんどは誤った処理の場合に I/O エラー例外を発生させるので，これを処理するように記述しないとコンパイルできない．

ネットワーク接続に関して図 2 のプログラムで最も重要なポイントは次の部分である．

```
Socket t = new Socket("133.36.50.23", 13);

BufferedReader is = new BufferedReader(
  new InputStreamReader(t.getInputStream()));
```

この部分の最初の行は**ソケット**をオープンしている．ソケットはネットワーク接続を抽象化したもので，ソケットに対して入出力を行うことで，ネットワークでの通信を行うことができる．ソケットをオープンするにはソケットのコンストラクタに接続先のホスト名とポート番号を指定する．

ソケットがオープンされると Socket(java.net.Socket) クラスの getInputStream メソッドを使って InputStream が得られる．InputStream はファイルからの入力に使うものと同じであり，あとの部分はファイルからの入力と同じようにプログラムすることができる．実際，この図 2 のプログラムで行なっているのは，

- readLine() を使ってサーバから送られた文字列を読み込む.
- 標準出力に書き出す.

という処理をストリームが終るまで繰り返しているだけである.

Socket クラスはネットワークの複雑な接続手順やデータの読み書きを隠し，ファイル入出力と同じようにプログラムできるようにしている.

参考のため図 3 にファイル入力のプログラムを示す.

```
import java.io.*;

class FileInputTest {
  public static void main(String[] args) {
    try {
      File f = new File( "FileInputTest.java" );

      BufferedReader is;
      is = new BufferedReader(
        new InputStreamReader(new FileInputStream(f)));
      boolean more = true;
      while ( more ) {
        String str = is.readLine();
        if (str == null)
          more = false;
        else
          System.out.println(str);
      }
    }
    catch ( IOException e ) {
        System.out.println("Error" + e);
    }
  }
}
```

図 3: ファイル入力のプログラム

なおソケットという抽象化を行うことでファイル入力と同じようにプログラムするというアイデアは Java 以前から UNIX でのネットワークプログラミングで標準的に使用されている．ただし，ネットワークのコネクションを確立してソケットを使うまでの手順は Java ほど簡単に記述できない．参考として図 2 と同じ処理を行うプログラムを C 言語で記述したものを図 4 に示す.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>
#include <arpa/inet.h>

int main(int ac, char *av[]) {
  struct sockaddr_in sin;
  struct hostent *hp;
  struct servent *sp;
  int t, more = 1;
  char buf[1024];
  FILE *fp;
  sin.sin_family = AF_INET;
  if ((sp = getservbyname("daytime","tcp")) == NULL ) {
    perror("getservbyname");
    exit(1);
  }
  sin.sin_port = sp->s_port;
  if ((hp = gethostbyname("133.36.50.23")) == NULL) {
    perror("gethostbyname");
    exit(1);
  }
  memcpy( &(sin.sin_addr.s_addr), hp->h_addr_list[0], sizeof(sin.sin_addr.s_addr));
  if ( (t = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) == -1 ) {
    perror("socket");
    exit(1);
  }
  if (connect(t, (struct sockaddr *)&sin, sizeof(sin)) == 0)
    while ( more ) {
      if ( recv( t, buf, 1024, 0) <= 0 )
        more = 0;
      else
        printf( "%s", buf );
    }
  else
    perror( "connect:" );
  close(t);
}
```


図 4: 簡単なクライアントのプログラム (C 版)

4　サーバの実現

今度はサーバのプログラムを作ってみよう．図 5 はクライアントから送られた文字列をオウム返しにそのまま返すプログラムである．サーバは特定のポートでクライアントからの接続を待たなければならないが，この例では 8189 を使用することにした．

クライアントプログラムと同様にネットワーク通信にはソケットを使用する．まず，ポート 8189 を監視する ServerSocket を作成する．

```
ServerSocket s = new ServerSocket(8189);
```

次に，クライアントが接続してくるのを待つ．

```
Socket incoming = s.accept();
```

クライアントが接続してくると，このメソッドは確立された通信路に相当する Socket オブジェクトを返してくる．ソケットが確立されてしまえば，InputStrem と OutputStream が得られ，ファイル入出力と同じように処理することができる．ここでは文字ベースで入出力を行うので，Reader，Writer を使用する．

```
BufferedReader in = new BufferedReader(
  new InputStreamReader(incoming.getInputStream()));
PrintWriter out = new PrintWriter(
  incoming.getOutputStream(), true /* autoFlush */);
```

残りの部分は特別なものはなく，メッセージを出力したあとは，「BYE」という文字列が入力されるまで，入力された文字列をそのまま出力している．最後の部分

```
incoming.close();
```

は，ソケットを閉じるための指示である．

ここに示したのは非常に単純なサーバであるが，基本的にサーバはこのプログラムと同様の処理を行う．つまり，

- クラアントから要求のメッセージを受け取る．
- 要求に従って何らかの処理を行う．
- 処理結果をメッセージとしてクライアントに送る

ということを繰り返すのである．

サーバプログラムをテストするには，2.1 と同様に telnet を使用する．まずプログラムをコンパイルして実行するのだが，このときに

```
java EchoServer &
```

のようにバックグラウンドジョブとして起動する．telnet での接続には，自分のホストの名前を指定すればよいのだが，自分自身をあらわす特別なアドレス (127.0.0.1) を使用してもよい．

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

public class EchoServer {
  public static void main(String[] args) {
    try {
        int port = 8189;
        if ( args.length > 0 )
            port = Integer.parseInt(args[0]);
        ServerSocket s = new ServerSocket( port );
        Socket incoming = s.accept();
        BufferedReader in
            = new BufferedReader(
                  new InputStreamReader( incoming.getInputStream()));
        PrintWriter out
            = new PrintWriter(incoming.getOutputStream(),true /* autoFlush */);
        out.println( "Hello! Enter BYE to exit." );
        boolean done = false;
        while (!done) {
            String str = in.readLine();
            if ( str == null )
                done = true;
            else {
                out.println( "Echo: " + str );
                if ( str.trim().equals("BYE") )
                    done = true;
            }
        }
        incoming.close();
    }
    catch (Exception e) {
        System.out.println(e);
    }
  }
}
```

図 5: 簡単なサーバのプログラム

telnet 127.0.0.1 8189

課題 5 図 *5* のプログラムを作成し，実行して動作を確認せよ．

5 複数のクライアントからの接続

前節のサーバプログラムでは，接続できるクライアントの数が 1 つだけに限定されていた．しかし，一般的には 1 つのサーバに対して世界中のユーザから同時に要求が行われることを考慮しなければならない．同時に複数のクライアントが接続できるようにするにはどうしたらよいだろうか．従来の UNIX でのネットワークプログラミングでは，プロセスの複製を作ることで複数のクライアントに対応していた．しかし，Java では言語が標準的に持っているスレッドの機能を使うことで複数の処理を同時に行うことが可能である．

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

public class ThreadedEchoServer {
  public static void main( String[] args ) {
    int i = 1;
    try {
      ServerSocket s = new ServerSocket( 8189 );
      for ( ;; ) {
        Socket incoming = s.accept();
        System.out.println( "Spawining " + i );
        new ThreadedEchoHandler( incoming, i ).start();
        i++;
      }
    }
    catch ( Exception e ) {
      System.out.println( e );
    }
  }
}
```

図 6: スレッド化サーバのプログラム

前節のサーバプログラムを複数のクライアントから接続できるようにスレッドを使って書き直したものを図 6，7 に示す．このプログラムは ThreadedEchoServer と ThreadedE-

```
class ThreadedEchoHandler extends Thread {
  public ThreadedEchoHandler( Socket i, int c ) {
    incoming = i;
    counter = c;
  }

  public void run() {
    try {
      BufferedReader in = new BufferedReader(
        new InputStreamReader( incoming.getInputStream() ) );
      PrintWriter out = new PrintWriter(
        incoming.getOutputStream(), true );

      out.println( "Hello! Enter BYE to exit." );

      boolean done = false;
      while ( !done ) {
        String str = in.readLine();
        if ( str == null )
          done = true;
        else {
          out.println( "Echo (" + counter + "): " + str );
          if ( str.trim().equals( "BYE" ) )
            done = true;
        }
      }
      incoming.close();
    }
    catch ( Exception e ) {
      System.out.println( e );
    }
  }

  private Socket incoming;
  private int counter;
}
```

図 7: スレッド化サーバのプログラム (続き)

choHandler からなり，ThreadedEchoServer では，クライアントからの接続を受け付けるたびに，その接続を引き継いだ新しい ThreadedEchoHandler を生成する．

なお，このプログラムは新しい接続を待ち続けるので，プログラムを終了するには，Ctrl + C を入力するなどして，強制的に終了させる必要がある．

課題 6 図 *6*，*7* を *1* つのファイル *(ThreadedEchoServer.java)* として作成し，コンパイル，実行して動作を確認せよ．