Server端Socket網路程式架構

課程大綱

1 Server端Socket應用程式流程

2 建立Server端Socket

3 取得Server端Socket資訊

4 接受Client端連結

5 接收與傳送 — Server端

6 關閉連結 — Server端

7 Server端範例

Server端Socket應用程式流程

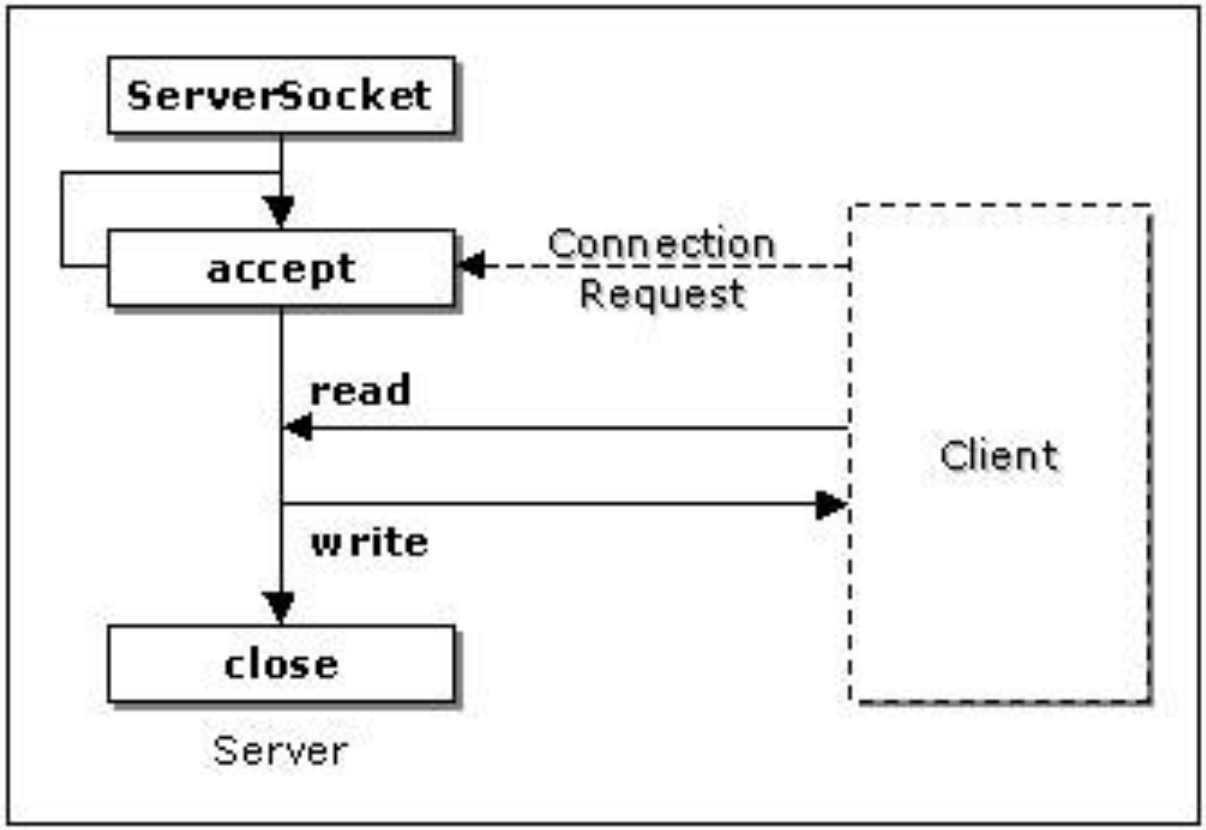
Server端應用

Socket是網路應用程式的核心，不論Server端或Client端網路應用程式，Socket皆為不可或缺之要素。

以Server端而言，常見的應用有：

- FTP Server (File Transfer Protocol)。
- Mail Server (SMTP、POP3、IMAP4 Protocol)。
- Web Server (HTTP Protocol)。

Server端Socket程式流程 (1)



Server端Socket程式流程 (2)

建構Server端Socket應用程式，其步驟大致如下：

- 建立Server端的Socket，並且以此等候Client端之連結請求（Connectoin Request）。

- 當Server端偵測到來自Client端的連結請求時，則接受此請求並藉此建立Client端之Socket，此Socket將做為此Client端連線及後續處理傳送及接收資料的依據，至此則完成Server端與Client端的Socket通訊連結。

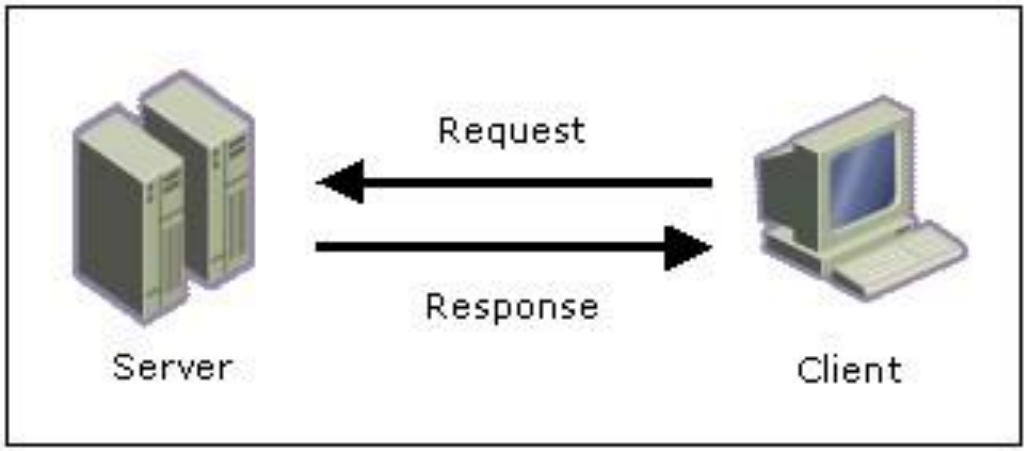
Server端Socket程式流程 (3)

- 處理來自Client端的資訊，一般稱為Request（請求），可視為Client端的指令需求，例如HTTP通訊協定的URL Request或FTP通訊協定的FTP指令（如get、put）等，皆可視為Client端Request。

- 根據Client端傳來的Request請求，Server端需經過程式邏輯處理之後，回傳相對應的執行結果或錯誤訊息至Client端，例如HTTP Server須回傳HTML網頁內容，而FTP Server則回傳FTP指令的結果。

- 當程式完成資料或指令的處理之後，便關閉Socket通訊連結。

Server端Socket程式流程 (4)



建立Server端Socket

建立Server端Socket (1)

欲建立Server端Socket，可使用以下java.net的建構子（Constructor）：

- `public ServerSocket(int port) throws IOException`
- `public ServerSocket(int port, int backlog) throws IOException`
- `public ServerSocket(int port, int backlog, InetAddress bindAddr) throws IOException`

建立Server端Socket (2)

第一種方法最為簡單，只要指定通訊埠號（port），`ServerSocket`便會以本機（local）的IP Address為Server端Socket所需使用的IP Address，並且預設最大的Client端連結數目為50，以此建立Server端Socket。

ServerSocket參數 (1)

- port：Server端所需使用的通訊埠，可為1至65535之間未被使用的埠號，若使用已被使用的通訊埠，則會產生IOException錯誤。若port參數設為0，則表示由ServerSocket自動尋找未被使用的通訊埠使用之。

- backlog：ServerSocket其預設最大Client端連結數為50，欲改變此數目，可自行設定backlog參數。

ServerSocket參數 (2)

- bindAddr：ServerSocket在建立時，會以本機（local）的IP Address為Server端Socket所需使用的IP Address。但有時local主機會有一個以上的IP Address時（例如：Sun Microsystems所推出的E10000或IBM的xServer等），可利用bindAddr參數指定欲用的local主機IP Address，bindAddr為InetAddress形式。

建立ServerSocket的程式架構


```
void startServer() {
  ServerSocket serverSocket;
  int port = <Port Number>;

  try {
    serverSocket = new ServerSocket(port);
    ...
  }
  catch (IOException ex) {
    ...
  }
}
```

取得Server端Socket資訊

取得Server端Socket資訊

Server端以ServerSocket建立Socket之後，可使用以下方法取得Server端Socket的相關資訊：

- `public InetAddress getInetAddress()`
- `public int getLocalPort()`

InetAddress 類別 (1)

getInetAddress方法用以取得Server端Socket的IP位址及主機名稱等資訊，並以InetAddress類別形式回傳，因此可使用InetAddress的下列方法取得Server端Socket的相關資訊：

- `public byte[] getAddress()`
- `public static InetAddress[] getAllByName(String host) throws UnknownHostException`

InetAddress 類別 (2)

- `public static InetAddress getByName(String host) throws UnknownHostException`
- `public String getHostAddress()`
- `public String getHostName()`
- `public static InetAddress getLocalHost()throws UnknownHostException`

InetAddress 類別 (3)

為確保主機有一個以上的IP Address時，可先使用getAllByName方法取得主機的所有IP內容，再以getHostName及getHostAddress方法分別取得某IP的主機名稱及IP位址。

```
ServerSocket serverSocket = new ServerSocket(<port>);

InetAddress[] addrs =
    serverSocket.getInetAddress().getAllByName("<hostname>") ;

for (int i=0; i < addrs.length; i++){
  System.out.println(addrs[i].getHostName());
  System.out.println(addrs[i].getHostAddress());
}
```

getLocalHost方法

getLocalHost方法則回傳有關主機名稱及IP Address資訊，並以下列方式表示：

```
Host Name/IP Address
```

例如：

```
leohuang/192.11.17.250
```

getLocalPort方法

用以回傳`ServerSocket`所使用的通訊埠號。

範例4-1 ServerInfo.java (1)


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

public class ServerInfo {
  public static void main(String[] args) {
    int port;

    if (args.length == 0) {
      System.out.println(
        "Usage: java ServerInfo [port]");
      System.exit(1);
    }
    port = Integer.parseInt(args[0]) ;
    startServer(port) ;
  }
```

範例4-1 ServerInfo.java (2)


```
public static void startServer(int port) {
  ServerSocket ss;

  try {
    ss = new ServerSocket(port);
    InetAddress addr =
      ss.getInetAddress().getLocalHost() ;

    System.out.println("Server Information: ");
    System.out.println("  Local Host: " +
      ss.getInetAddress().getLocalHost());
    System.out.println("  Host Name : " +
      addr.getHostName());
    System.out.println("  IP address: " +
      addr.getHostAddress());
    System.out.println("  Port      : " +
      ss.getLocalPort());
```

範例4-1 ServerInfo.java (3)


```
        InetAddress[] addrs =
          ss.getInetAddress().getAllByName(
          addr.getHostName()) ;

        System.out.println("IP Address(es): ");

        for (int i=0; i < addrs.length; i++){
          System.out.println("  " +
            addrs[i].getHostAddress());
        }
      }
      catch (IOException ex) {
        ex.printStackTrace();
      }
    }
}
```

範例4-1執行結果

```
C:\>java ServerInfo
Usage: java ServerInfo [port]

C:\>java ServerInfo 80
Server Information:
  Local Host : leohuang/192.11.17.250
  Host Name  : leohuang
  IP address : 192.11.17.250
  Port       : 80

IP Address(es):
  192.11.17.250
```

接受Client端連結

接受Client端連結

以ServerSocket建立Server端的Socket之後，基本上Server的雛型已經具備了，接著便是等候（listen）及接受來自Client端的連線。

處理Client端連線，可使用ServerSocket類別的accept方法：

- `public Socket accept() throws IOException`

accept方法 (1)

`accept`方法用以接受Client端的連線請求（Connection Request），並且建立`Socket`類別物件，此物件用以代表此連結Client端，並藉此作為Server端與Client端連線及資料傳送接收之依據。

accept方法 (2)

程式片段：

```
while (true) {
  try {
    Socket clientSocket = serverSocket.accept();
    ...
  }
  catch (IOException ex) {
    ...
  }
}
```

while迴圈在此的用意表示Server端Socket等候（listen）Client端之連結。

處理多人連線 (1)

上述的程式片段有以下問題存在：

- 由於程式以`while`迴圈判斷是否有來自Client端的連線請求，程式會在迴圈中停頓（無限迴圈）至Client端連線結束為止，因此若程式中有其它的功能如圖形化使用者界面，將無法正常運作。

- 當Client端建立`Socket`物件之後，下一個Client端必須等此Client端結束連線，方能再產生連線，無法達到多人連線的目的。

為處理多人連線，Server端程式會以執行緒（Thread）為每一個Client端建立各自的執行緒。

處理多人連線 (2)


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

Thread thread = new Thread(new listenClient(serverSocket));
thread.start();
...

class listenClient implements Runnable {
 private ServerSocket serverSocket ;
 private Socket clientSocket;

 public listenClient (ServerSocket serverSocket) {
   this.serverSocket = serverSocket;
 }
```

處理多人連線 (3)

```
public void run() {
  try {
    while (true) {
      Socket clientSocket = serverSocket.accept();
      ...
    }
  }
  catch (IOException ex) {
    ...
  }
 }
}
```

範例4-2 AcceptClient.java (1)


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

public class AcceptClient {
  public static void main(String[] args) {
    int port;

    if (args.length == 0) {
      System.out.println(
        "Usage: java AcceptClient [port]");
      System.exit(1);
    }
    port = Integer.parseInt(args[0]) ;
    startServer(port) ;
  }
```

範例4-2 AcceptClient.java (2)


```
public static void startServer(int port) {
  ServerSocket ss;
  try {
    ss = new ServerSocket(port);

    while (true) {
      Socket cs = ss.accept();
      System.out.println("Connection from Client IP: " +
        cs.getInetAddress().getHostAddress());
    }
  }
  catch (IOException ex) {
    ex.printStackTrace();
  }
 }
}
```

範例4-2執行結果

- C:\>java AcceptClient 80

- C:\>telnet localhost 80

Connection from Client IP: 127.0.0.1

接收與傳送 — Server端

Server端接收與傳送

Server端最重要的目的是接受自Client端的資訊或指令，經過邏輯處理之後，再將執行結果回傳至Client端。

當Server端以accept方法建立Client端Socket物件之後，便可使用Socket物件所提供的下列方法取得所連結Client端的輸出入資料流（Input/Output Stream）：

- `public InputStream getInputStream() throws IOException`
- `public OutputStream getOutputStream() throws IOException`

getInputStream方法 (1)

getInputStream用以取得所連結Client端的輸入資料流（Input Stream），代表Client端傳送至Server端的資訊，並以java.io.InputStream形式表示，通常會以DataInputStream或BufferedReader類別承接InputStream資料流。

- java.io.DataInputStream
- java.io.BufferedReader

getInputStream方法 (2)

當輸入資料流建立之後，便可使用輸入資料流的read方法，讀取Client端所傳送來的資訊。

DataInputStream:

- read
- readByte
- readChar
- readDouble
- readFloat
- readFully
- readInt
- readLong
- readShort
- readUnsignedByte
- readUnsignedShort
- readUTF

BufferedReader:

- read
- readLine

getInputStream方法 (3)

以下為接收Client端資訊的程式片段：

```
try {
  Socket cs = ss.accept();

  DataInputStream in =
    new DataInputStream (cs.getInputStream());
  String inData = in.readUTF();
}
catch (IOException ex) {
  ...
}
```

getOutputStream方法 (1)

欲傳送資料至Client端，需先以getOutputStream取得Client端的輸出資料流（Output Stream），此方法會以java.io.OutputStream形式表示，通常會以DataOutputStream或BufferedWriter類別轉承OutputStream資料流：

- java.io.DataOutputStream
- java.io.BufferedWriter

getOutputStream方法 (2)

當輸出資料流建立之後，便可使用輸出資料流的write方法，傳送資料至Client端。

DataOutputStream:

- write
- writeBoolean
- writeByte
- writeBytes
- writeChar
- writeChars
- writeDouble
- writeFloat
- writeInt
- writeLong
- writeShort
- writeUTF

Bufferedwriter:

- write

getOutputStream方法 (3)

以下為傳送資料至Client端的程式片段：

```
try {
   Socket clientSocket = serverSocket.accept();

   DataOutputStream out =
   new DataOutputStream(clientSocket.getOutputStream());

   out.writeUTF(<Data>);
   ...
}
catch (IOException ex) {
   ...
}
```

關閉連結 — Server端

Server端關閉連結

當Server端程式結束時，需以ServerSocket類別的close方法關閉Server端Socket並釋放資源。

- `public void close() throws IOException`

另外，當Client端結束連結時，Server端程式同樣需以Socket類別的close方法，關閉Client端Socket並釋放資源。

- `public void close() throws IOException`

至此便完成整個Server端程式的流程。

Server端範例

範例4-3 SimpleServer.java (1)


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

public class SimpleServer {
  private static ServerSocket serverSocket;
  private static listenClient listen ;

  public static void main(String[] args) throws Exception {
    int port;

    if (args.length == 0) {
      System.out.println("Usage: java SimpleServer [port]");
      System.exit(1);
    }
```

範例4-3 SimpleServer.java (2)


```
    port = Integer.parseInt(args[0]) ;
    startServer(port) ;
  }

  public static void startServer(int port) throws Exception{
    try {
      serverSocket = new ServerSocket(port);

      Thread thread = new Thread(
        new listenClient(serverSocket));
      thread.start();
    } catch (IOException ex) {
      ex.printStackTrace();
    }
  }
}
```

範例4-3 SimpleServer.java (3)


```
class listenClient implements Runnable {
  private ServerSocket serverSocket;
  private Socket clientSocket;

  DataInputStream  in;
  DataOutputStream out;
  // constructor
  public listenClient(ServerSocket serverSocket)
    throws Exception {
    this.serverSocket = serverSocket;
  }

  public void run() {
    try {
      while(true) {
        clientSocket = serverSocket.accept();
```

範例4-3 SimpleServer.java (4)


```
System.out.println("Connection from " +
  clientSocket.getInetAddress().getHostAddress());

in  = new DataInputStream(
  clientSocket.getInputStream());
out = new DataOutputStream(
  clientSocket.getOutputStream());

// Line Separator
String lineSep =
  System.getProperty("line.separator");

InetAddress addr =
  serverSocket.getInetAddress().getLocalHost() ;
```

範例4-3 SimpleServer.java (5)

```
            String outData = "Server Information: " + lineSep +
              "   Local Host: " +
              serverSocket.getInetAddress().getLocalHost() +
              lineSep + "   Port       : " +
              serverSocket.getLocalPort();

            byte[] outByte = outData.getBytes();
            out.write(outByte, 0, outByte.length);
          }
        }
        catch (Exception ex) {
          ex.printStackTrace();
        }
      }
    }
```

範例4-3執行方式

- 執行以下的指令（假設使用Port 1024）：

```
java SimpleServer 1024
```

- 利用Telnet程式連結至上述電腦（假設Server端主機名稱為localhost）：

```
telnet localhost 1024
```

範例4-3執行結果

- Server端執行結果：

```
C:\>java SimpleServer 1024
Connection from 192.11.17.251
Connection from 127.0.0.1
```

- Client端（Telnet）顯示結果：

```
Server Information:
  Local Host: localhost/192.11.17.250
  Port      : 1024
```

本章結論

在本章中，主要介紹Server端Socket應用程式的架構及流程，基本上，Server程式大致上皆依照此架構開發，主要的不同在於不同型態的通訊協定，如SMTP、POP3、FTP、HTTP等，所處理的邏輯不儘相同而已，因此有了以上的程式架構，已可開發Server端程式了。