Client端Socket網路程式架構

課程大綱

1 Client端Socket應用程式流程

2 建立Client端Socket

3 取得Client端Socket資訊

4 接收與傳送 — Client端

5 關閉連結 — Client端

6 Client端範例

Client端Socket應用程式流程

Client端 v.s. Server端Socket程式

基本上，Client端Socket應用程式與Server端Socket應用程式其流程類似，最大的差別在於：

- Server端Socket應用程式主要在等候及接受Client端的連結，而Client端Socket應用程式則在於嘗試與Server端建立連結。

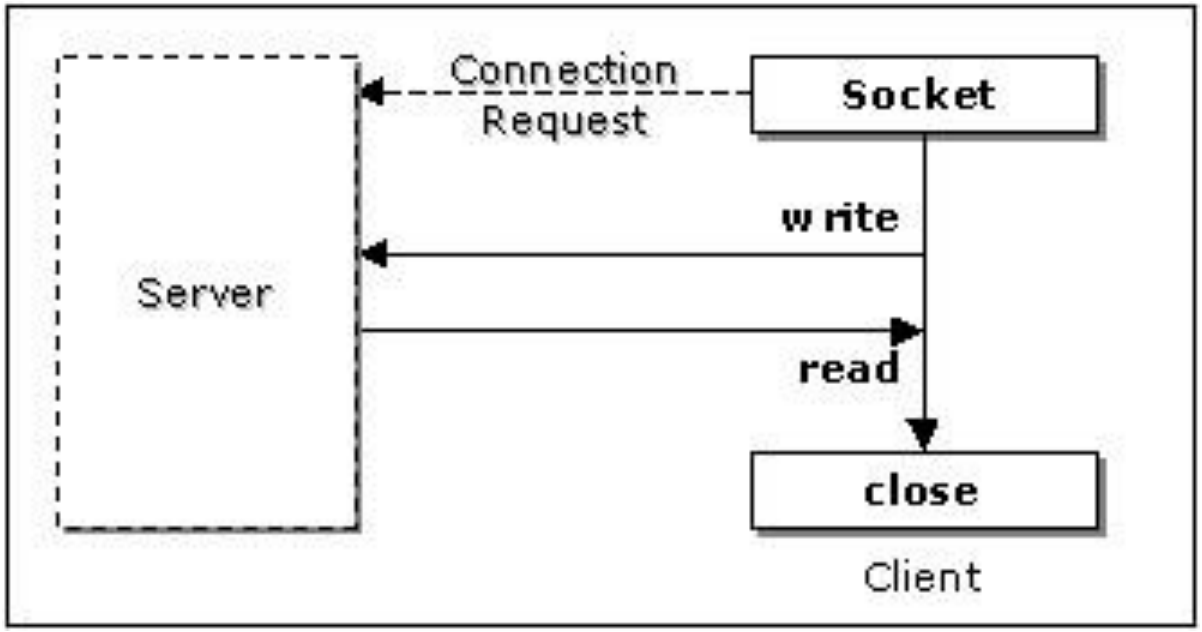
- Client端Socket應用程式傳送訊息指令至Server端以及接收Server端所回傳的結果。而Server端Socket應用程式則在處理指令邏輯並將結果或錯誤訊息傳送至Client端。

Client端應用

常見的Client端應用有：

- Chat Client。
- FTP Client。
- POP3 Client。
- SMTP Client。
- Telnet Client。

Client端Socket程式流程 (1)



Client端Socket程式流程 (2)

建構Client端Socket應用程式，其步驟大致如下：

- 建立Client端Socket，在建立時需指定欲連結Server端的主機名稱（或IP Address）與Internet服務的通訊埠（Port）。
- 傳送特定資訊或指令至Server端。
- 接收Server端回傳的執行結果或錯誤訊息，並以特定格式顯示。例如HTTP通訊協定會以HTML內容顯示。
- 當Client端不需Server端的處理時，便關閉Socket通訊連結。

建立Client端Socket

建立Client端Socket (1)

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

- `public Socket(InetAddress address, int port) throws IOException`
- `public Socket(InetAddress address, int port, InetAddress localAddr, int localPort) throws IOException`
- `public Socket(String host, int port) throws UnknownHostException, IOException`
- `public Socket(String host, int port, InetAddress localAddr, int localPort) throws UnknownHostException, IOException`

建立Server端Socket (2)

須注意的是，Client端Socket與Server端Socket的意義是有所不同，後者為等候Client端之連結，而前者則是連結至Server端，因此Client端Socket在建立時，需指定Server的主機名稱（或IP Address）及Internet服務的通訊埠（Port）。

Socket參數 (1)

- `address`：設定Server端的IP Address，為`InetAddress`形式。

- `port`：設定Server端所提供之Internet服務的通訊埠號。

- `localAddr`：若Client端主機有一個以上的IP Address時，可設定`localAddr`參數指定欲使用的Client端主機IP Address，為`InetAddress`形式。

Socket參數 (2)

- localPort：Client端在連結Server端時，在未指定通訊埠時，Client端Socket會自行尋找Client端尚未被使用的通訊埠，以便建立Socket。但亦可透過設定localPort參數自行指定此通訊埠。不過建議非必要時，不要使用localAddr及localPort參數宣告Client端所需的本機（local）IP Address及通訊埠，因為若通訊埠已被使用，將產生IOException錯誤，不妨由Socket建構子自行決定。

- host：除了可指定Server端的IP Address之外，也可以用主機名稱代替IP Address。使用主機名稱，會經由DNS（Domain Name Server）轉換為相對的IP Address，若DNS未能成功轉換，則代表此主機名稱未被定義或不正確，將產生UnknownHostException錯誤。

建立Client端Socket的程式架構 (1)


```
Socket socket;
int port = <Port Number>;

try {
  socket =
    new Socket(
      InetAddress.getByName("<Host Name>"), port);
  ...
}
catch (IOException ex) {
  ...
}
```

建立Client端Socket的程式架構 (2)

```
Socket socket;
int port = <Port Number>;

try {
  socket = new Socket("<Host Name>", port);
  ...
}
catch (UnknownHostException e) {
  ...
}
catch (IOException ex) {
  ...
}
```

取得Client端Socket資訊

取得Client端Socket資訊

較之於ServerSocket，Socket物件除了提供取得本機（local）所使用的IP Address及通訊埠號的API之外，也提供所連結Server端資訊的API，共有以下的方法：

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

getLocalAddress方法

getLocalAddress方法用以取得Client端Socket所使用的IP Address及主機名稱，並以InetAddress類別形式回傳，因此可使用InetAddress的下列方法取得Client端Socket的相關資訊：

- public byte[] getAddress()
- public static InetAddress[] getAllByName(String host) throws UnknownHostException
- public static InetAddress getByName(String host)throws UnknownHostException
- public String getHostAddress()
- public String getHostName()
- public static InetAddress getLocalHost() throws UnknownHostException

getLocalPort方法

`getLocalPort`用以回傳Client端Socket所使用的通訊埠號。

getInetAddress方法

`getInetAddress`取得所連結Server端的IP Address及主機名稱，並以`InetAddress`類別形式回傳。

getPort方法

`getPort`回傳所連結Server端Internet服務所用的通訊埠號。

範例4-5 ClientInfo.java (1)


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

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

    if (args.length < 2) {
      System.out.println(
        "Usage: java ClientInfo
        [Remote IP/Host] [port]");
      System.exit(1);
    }
    host = args[0];
    port = Integer.parseInt(args[1]);
```

範例4-5 ClientInfo.java (2)


```
    connectServer(host, port);
  }

  public static void connectServer(String host,
    int port) {
    try {
      Socket socket = new
        Socket(InetAddress.getByName(host), port);

      InetAddress addr =
        socket.getLocalAddress().getLocalHost() ;

      // Get Client Socket Information
      System.out.println("Client Information: ");
      System.out.println("  Local Host: " +
        socket.getLocalAddress().getLocalHost());
```

範例4-5 ClientInfo.java (3)


```
System.out.println("  Host Name : " +
  addr.getHostName());
System.out.println("  IP address: " +
  addr.getHostAddress());
System.out.println("  Port      : " +
  socket.getLocalPort());
System.out.println();

InetAddress[] addrs =
  socket.getLocalAddress().getAllByName(
  addr.getHostName()) ;
System.out.println("IP Address(es): ");

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

範例4-5 ClientInfo.java (4)


```
        // Get Server Socket Information
        System.out.println("Connection to Remote: " +
          socket.getInetAddress().getHostAddress() + ":" +
          socket.getPort());
        System.out.println();
      }
      catch (UnknownHostException e) {
        e.printStackTrace();
      }
      catch (IOException ex) {
        ex.printStackTrace();
      }
    }
}
```

範例4-5執行結果

```
C:\>java ClientInfo
Usage: java ClientInfo [Remote IP/Host] [port]

C:\>java ClientInfo localhost 80
Client Information:
  Local Host : leohuang/192.11.17.250
  Host Name  : leohuang
  IP address : 192.11.17.250
  Port       : 1024

IP Address(es):
  192.11.17.250

Connection to Remote: 127.0.0.1:80
```

接收與傳送 — Client端

Client端接收與傳送

當Client端成功建立與Server端的連結之後，便可開始傳送訊息指令至Server端（稱為Request）及接收Server端的處理結果（稱為Response）。

與先前所介紹的方法一樣，Client端所建立的Socket物件代表著Client端與Server端之連結，可使用下列方法取得所連結Server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Server端的輸入資料流（Input Stream），代表Server端傳送至Client端的回應，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Server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Server端資訊的程式片段：

```
try {
   Socket socket = new Socket(
      InetAddress.getByName(host), port);

   DataInputStream in = new DataInputStream(
      socket.getInputStream());
   String inData = in.readUTF();
}
catch (UnknownHostException e) {
   ...
}
catch (IOException ex) {
   …
}
```

getOutputStream方法 (1)

代表Client端傳送資料至Server端的輸出資料流（Output Stream），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Server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Server端的程式片段：

```
try {
   Socket socket = new Socket(
      InetAddress.getByName(host), port);

   DataOutputStream out = new DataOutputStream
      (socket.getOutputStream());
   out.writeUTF(<Data>);
}
catch (UnknownHostException e) {
   ...
}
catch (IOException ex) {
   ...
}
```

關閉連結 — Client端

Client端關閉連結

當Client端結束連結時，需以Socket類別的close方法，關閉Client端與Server端的連結並釋放資源。

- `public void close() throws IOException`

至此便完成整個Client端程式的流程。

Client端範例

範例4-7 SimpleClient.java (1)


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

public class SimpleClient {
  private static Socket socket;

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

    if (args.length < 2) {
      System.out.println(
        "Usage: java SimpleClient [Remote IP/Host] [port]");
      System.exit(1);
    }
```

範例4-7 SimpleClient.java (2)


```
    host = args[0];
    port = Integer.parseInt(args[1]);

    connectServer(host, port);
  }

  public static void connectServer(String host, int port) {
    try {
      socket = new Socket(
        InetAddress.getByName(host), port);

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

範例4-7 SimpleClient.java (3)


```
    byte[] inByte = new byte[1024];
    in.read(inByte);
    String response = new String(inByte, 0,
      inByte.length);

    System.out.println("Message from server: ");
    System.out.println(response.trim());
  }
  catch (UnknownHostException e) {
    e.printStackTrace();
  }
  catch (IOException ex) {
    ex.printStackTrace();
  }
```

範例4-7 SimpleClient.java (4)


```
    finally {
      try {
        socket.close();
      }
      catch (IOException ioe) {
        ioe.printStackTrace();
      }
    }
  }
}
```

範例4-7執行方式

本範例可與範例SimpleServer.java一起執行，可以瞭解資料如何在Client/Server之間傳遞，由於在SimpleServer.java程式中是以byte（位元）方式傳送資料，所以在simpleClient.java程式中亦以byte方式接收資料。

範例4-7執行結果

- Client端執行結果：

```
C:\>java SimpleClient localhost 80
Message from server:
Server Information:
  Local Host: localhost/192.11.17.250
  Port      : 80
```