Lập trình mạng (Networking) trong Java

Khái niệm *lập trình mạng (network programming)* nói về viết các chương trình mà thực thi qua nhiều thiết bị (máy tính), trong đó các thiết bị này được kết nối mạng với nhau.

Để hiểu sâu hơn các khái niệm được trình bày trong chương này, mời bạn tham khảo loạt bài: **Ví dụ về Lập trình mạng (Networking) trong Java**.

Gói java.net của J2SE API chứa một tập hợp các class và interface mà cung cấp chi tiết về giao tiếp tầm thấp, cho phép bạn viết các chương trình mà trọng tâm vào giải quyết vấn đề trong tầm tay.

Gói java.net cung cấp sự hỗ trợ cho hai giao thức mạng phổ biến:

- **TCP:** TCP là viết tắt của *Transmission Control Protocol*, cho phép sự giao tiếp đáng tin cậy giữa hai ứng dụng. TCP đặc trưng được sử dụng qua Internet Protocol, được xem như là TCP/IP.
- **UDP:** UDP là viết tắt của *User Datagram Protocol*, một giao thức connection-less, cho phép các gói nhỏ dữ liệu được truyền tải giữa các ứng dụng.

Chương này giúp bạn hiểu sâu về hai chủ đề trong Java:

- **Lập trình Socket**: Đây là khái niệm được sử dụng rộng rãi nhất trong lập trình mạng và nó được giải thích rất chi tiết.
- **Tiến trình URL**: Nó sẽ được trình bày riêng. Bạn click vào link sau để học về **Tiến trình URL trong Java**.

Lập trình Socket trong Java

Các Socket cung cấp kỹ thuật giao tiếp giữa hai máy tính sử dụng TCP. Một chương trình Client tạo một socket trên đầu cuối của giao tiếp và cố gắng để kết nối socket đó tới một Server.

Khi kết nối được tạo, Server tạo một đối tượng Socket trên đầu cuối của giao tiếp. Client và Server bây giờ có thể giao tiếp bằng việc đọc và ghi từ Socket.

Lớp java.net.Socket biểu diễn một Socket, và lớp java.net.ServerSocket cung cấp một kỹ thuật cho chương trình Server để nghe thông tin từ các Client và thành lập các kết nối với chúng.

Các bước sau xảy ra khi thành lập một kết nối TCP giữa hai máy tính sử dụng Socket:

- Server thuyết minh một đối tượng ServerSocket, biểu thị số hiệu cổng (port) nào để xuất hiện giao tiếp.
- Server gọi phương thức accept() của lớp ServerSocket. Phương thức này đợi tới khi một Client kết nối với Server trên cổng đã cho.
- Sau khi Server đang đợi, một Client khởi tạo một đối tượng Socket, xác định tên Server và số hiệu cổng để kết nối tới.
- Constructor của lớp Socket cố gắng để kết nối Client tới Server và số hiệu cổng đã xác định. Nếu giao tiếp được thành lập, bây giờ Client có một đối tượng Socket có khả năng giao tiếp với Server.
- Trên Server-side, phương thức accept() trả về một tham chiếu tới một socket mới trên Server mà được kết nối với socket của Client.

Sau khi các kết nối được thành lập, giao tiếp có thể xảy ra bởi sử dụng I/O stream. Mỗi Socket có cả một OutputStream và InputStream. OutputStream của Client được kết nối với InputStream của Server, và InputStream của Client được kết nối với OutputStream của Server.

TCP là một giao thức giao tiếp hai chiều, vì thế dữ liệu có thể được gửi qua cả hai luồng tại cùng một thời điểm. Các lớp hữu ích sau đây cung cấp đầy đủ các phương thức để triển khai các Socket.

Các phương thức lớp ServerSocket trong Java

Lớp **java.net.ServerSocket** trong Java được sử dụng bởi các ứng dụng Server để thu nhận một cổng và nghe các yêu cầu từ Client.

Lớp ServerSocket có 4 constructor sau:

STT	Phương thức và miêu tả
1	**public ServerSocket(int port) throws IOException** Cố gắng để tạo một Server Socket giới hạn với số hiệu cổng đã xác định. Một

	exception xuất hiện nếu cổng này thuộc phạm vi của ứng dụng khác.
2	**public ServerSocket(int port, int backlog) throws IOException** Tương tự như constructor trước, tham số backlog xác định có bao nhiêu Client đang vào để lưu giữ trogn một hàng đợi (wait queue)
3	**public ServerSocket(int port, int backlog, InetAddress address) throws IOException** Tương tự như constructor trước, tham số InetAddress xác định địa chỉ IP nội bộ để kết nối tới. InetAddress này được sử dụng cho các Server mà có thể có nhiều địa chỉ IP, cho phép Server xác định địa chỉ IP nào của nó để chấp nhận yêu cầu của Client
4	**public ServerSocket() throws IOException** Tạo một Server Socket không giới hạn. Khi sử dụng constructor này, sử dụng phương thức bind() khi bạn đã kết nối với Server Socket

Nếu ServerSocket constructor không ném một exception, nghĩa là ứng dụng của bạn đã thành công kết nối tới cổng đã xác định và sẵn sàng cho các yêu cầu của Client.

Bảng dưới liệt kê các phương thức phổ biến của lớp ServerSocket trong Java:

STT	Phương thức và miêu tả
1	**public int getLocalPort()** Trả về cổng mà Server Socket đang nghe trên đó. Phương thức này hữu dụng nếu bạn truyền số hiệu cổng là 0 trong một constructor và để Server tìm một cổng cho bạn
2	**public Socket accept() throws IOException** Đợi cho một Client đang đến. Phương thức này block tới khi một Client kết nối tới Server trên cổng đã xác định hoặc Socket là trễ (timeout), giả sử rằng giá trị time-out đã được thiết lập với phương thức setSoTimeout(). Nếu không thì, phương thức này

	block vô hạn
3	**public void setSoTimeout(int timeout)** Thiết lập giá trị timeout cho Server Socket đợi một Client trong bao lâu, trong khi phương thức accept() gọi
4	**public void bind(SocketAddress host, int backlog)** Nối kết Socket tới Server và cổng đã xác định trong đối tượng SocketAddress. Sử dụng phương thức này nếu bạn đã thuyết minh đối tượng ServerSocket với constructor không có tham số

Khi ServerSocket gọi phương thức accept(), phương thức này không trả về giá trị tới khi một Client kết nối. Sau khi một Client kết nối, ServerSocket tạo một Socket mới trên một cổng chưa được xác định và trả về một tham chiếu tới Socket mới này. Bây giờ, một kết nối TCP tồn tại giữa Client và Server, và giao tiếp có thể bắt đầu.

Các phương thức lớp Socket trong Java

Lớp **java.net.Socket** biểu diễn socket mà cả Client và Server sử dụng để kết nối với nhau. Client thu nhận một đối tượng Socket bằng việc thuyết minh nó, trong khi Server thu nhận một đối tượng Socket từ giá trị trả về của phương thức accept().

Lớp Socket có 5 constructor mà một Client sử dụng để kết nối tới một Server:

STT	Phương thức và Miêu tả
1	**public Socket(String host, int port) throws UnknownHostException, IOException.** Phương thức này cố gắng kết nối tới Server đã xác định tại cổng đã xác định. Nếu constructor này không ném một exception, kết nối là thành công và Client được kết nối tới Server
2	**public Socket(InetAddress host, int port) throws IOException** Phương thức này tương tự constructor trước, ngoại trừ host được biểu thị bởi một đối

	tượng InetAddress
3	**public Socket(String host, int port, InetAddress localAddress, int localPort) throws IOException.** Kết nối tới host và cổng đã xác định, tạo một Socket trên host nội bộ tại địa chỉ và cổng đã xác định
4	**public Socket(InetAddress host, int port, InetAddress localAddress, int localPort) throws IOException.** Phương thức này tương tự constructor trước, ngoại trừ host được biểu thị bởi một đối tượng InetAddress thay vì một String
5	**public Socket()** Tạo một Socket rời rạc. Sử dụng phương thức connect() để kết nối Socket này tới một Server

Khi Socket constructor trả về giá trị, nó không đơn giản chỉ khởi tạo một đối tượng Socket mà nó còn thực sự cố gắng kết nối tới Server và cổng đã xác định.

Một số phương thức của lớp Socket đáng quan tâm được liệt kê dưới đây. Chú ý rằng, cả Server và Client đều có một đối tượng Socket, vì thế những phương thức này có thể được gọi bởi cả Client và Server.

STT	Phương thức và Miêu tả
1	**public void connect(SocketAddress host, int timeout) throws IOException** Phương thức này kết nối Socket tới host đã xác định. Phương thức này chỉ cần thiết khi bạn đã thuyết minh Socket bởi sử dụng constructor không có tham số
2	**public InetAddress getInetAddress()** Phương thức này trả về địa chỉ của máy tính khác mà Socket này được kết nối tới

3	**public int getPort()** Trả về cổng mà Socket được kết nối trên thiết bị từ xa
4	**public int getLocalPort()** Trả về cổng mà Socket được kết nối trên thiết bị nội bộ
5	**public SocketAddress getRemoteSocketAddress()** Trả về địa chỉ của Socket từ xa
6	**public InputStream getInputStream() throws IOException** Trả về input stream của Socket. Input stream được kết nối tới output stream của Socket từ xa
7	**public OutputStream getOutputStream() throws IOException** Trả về output stream của Socket. Output stream được kết nối tới input stream của Socket từ xa
8	**public void close() throws IOException** Đóng Socket, mà làm đối tượng Socket này không còn khả năng kết nối lại với bất kỳ Server nào

Các phương thức lớp InetAddress trong Java

Lớp này biểu diễn một địa chỉ Internet Protocol (IP). Dưới đây liệt kê một số phương thức hữu ích mà bạn sẽ cần trong khi lập trình Socket.

STT	Phương thức và Miêu tả
1	**static InetAddress getByAddress(byte[] addr)** Trả về một đối tượng InetAddress đã cung cấp địa chỉ IP thô

2	**static InetAddress getByAddress(String host, byte[] addr)** Tạo một InetAddress dựa trên tên host và địa chỉ IP đã cung cấp
3	**static InetAddress getByName(String host)** Xác định địa chỉ IP của một host, đã cung cấp tên host
4	**String getHostAddress()** Trả về chuỗi địa chỉ IP dạng biểu diễn nguyên văn
5	**String getHostName()** Nhận tên host cho địa chỉ IP này
6	**static InetAddress InetAddress getLocalHost()** Trả về host nội bộ
7	**String toString()** Biến đổi địa chỉ IP này thành một String

Ví dụ về Socket Client trong Java

GreetingClient sau là một chương trình Client kết nối tới một Server bởi sử dụng một Socket và gửi một lời chào, và sau đó đợi một phản hồi.

```
// File Name GreetingClient.java


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


public class GreetingClient
{
   public static void main(String [] args)
   {
```

```
      String serverName = args[0];
      int port = Integer.parseInt(args[1]);
      try
      {
         System.out.println("Connecting to " + serverName
                            + " on port " + port);
         Socket client = new Socket(serverName, port);
         System.out.println("Just connected to "
                     + client.getRemoteSocketAddress());
         OutputStream outToServer = client.getOutputStream();
         DataOutputStream out =
                      new DataOutputStream(outToServer);

         out.writeUTF("Hello from "
                     + client.getLocalSocketAddress());
         InputStream inFromServer = client.getInputStream();
         DataInputStream in =
                       new DataInputStream(inFromServer);
         System.out.println("Server says " + in.readUTF());
         client.close();
      }catch(IOException e)
      {
         e.printStackTrace();
      }
   }
}
```

Ví dụ về Socket Server trong Java

Chương trình GreetingServer sau là ví dụ về một ứng dụng Server sử dụng lớp Socket để lắng nghe các Client trên một số hiệu cổng đã xác định bởi một tham số dòng lệnh.

```
// File Name GreetingServer.java


import java.net.*;
```

```
import java.io.*;


public class GreetingServer extends Thread
{
   private ServerSocket serverSocket;


   public GreetingServer(int port) throws IOException
   {
      serverSocket = new ServerSocket(port);
      serverSocket.setSoTimeout(10000);
   }


   public void run()
   {
      while(true)
      {
         try
         {
            System.out.println("Waiting for client on port " +
            serverSocket.getLocalPort() + "...");
            Socket server = serverSocket.accept();
            System.out.println("Just connected to "
                  + server.getRemoteSocketAddress());
            DataInputStream in =
                  new DataInputStream(server.getInputStream());
            System.out.println(in.readUTF());
            DataOutputStream out =
                 new DataOutputStream(server.getOutputStream());
            out.writeUTF("Thank you for connecting to "
              + server.getLocalSocketAddress() + "\nGoodbye!");
            server.close();
         }catch(SocketTimeoutException s)
         {
```

```
            System.out.println("Socket timed out!");
            break;
         }catch(IOException e)
         {
            e.printStackTrace();
            break;
         }
      }
   }
   public static void main(String [] args)
   {
      int port = Integer.parseInt(args[0]);
      try
      {
         Thread t = new GreetingServer(port);
         t.start();
      }catch(IOException e)
      {
         e.printStackTrace();
      }
   }
}
```

Biên dịch Client và Server và sau đó bắt đầu Server như sau:

```
$ java GreetingServer 6066
Waiting for client on port 6066...
```

Kiểm tra chương trình Client như sau:

```
$ java GreetingClient localhost 6066
Connecting to localhost on port 6066
Just connected to localhost/127.0.0.1:6066
Server says Thank you for connecting to /127.0.0.1:6066
Goodbye!
```