Sistem Terdistribusi

TIK-604

Husni.trunojoyo.ac.id

Pemrograman Socket dengan Java: Server

Topik Praktik (Belajar Mandiri)

Husni

husni@trunojoyo.ac.id

Garis Besar Bahasan

- Langkah-langkah Pembuatan Server
  1. Membuat obyek ServerSocket
  2. Membuat obyek Socket dari ServerSocket
  3. Membuat input stream
  4. Membuat output stream
  5. Melakukan operasi I/O dengan aliran (*stream*) input dan output
  6. Menutup socket
- Server jaringan generik
  - Berthread satu (*single threaded*)
  - Berthread banyak (*multithreaded*)
- Menerima koneksi dari browser
- Server HTTP sederhana

Dasar-dasar

Langkah-langkah Implementasi Server

1. Membuat obyek ServerSocket

```
ServerSocket listenSocket = new ServerSocket(portNumber);
```

2. Membuat obyek Socket dari ServerSocket

```
while(someCondition) {
  Socket server = listenSocket.accept();
  doSomethingWith(server);
}
```

Biasanya **doSomethingWith** digilirkan untuk thread-thread terpisah

3. Membuat input stream untuk membaca input dari client

```
BufferedReader in = new BufferedReader(new
   InputStreamReader(server.getInputStream()));
```

Langkah-langkah Implementasi Server

4. Membuat output stream yang dapat digunakan untuk mengirimkan info balik ke client

```
// Argumen terakhir true berarti autoflush stream
// ketika println dipanggil
PrintWriter out = new PrintWriter(server.getOutputStream(), true);
```

5. Lakukan operasi I/O dengan input dan output streams
   - Biasanya membaca di dalam loop
   - Biasanya memberikan respon dalam thread terpisah
   - Cara paling sering membaca input: `lines` atau `readLine`
   - Cara paling umum mengirimkan output: `printf`
6. Menutup socket saat selesai

```
server.close();    // atau gunakan try-with-resources
```

   - Ini menutup stream input dan output yang berasosiasi.

Ingat Kelas Bantuan: SocketUtils

- Gagasan
  - Cara lama dengan membuat `BufferedReader` dan `PrintWriter` dari Socket. SocketUtils sedikit menyederhanakan sintaks
- Tanpa SocketUtils (untuk Socket s)
  - `PrintWriter out = new PrintWriter(s.getOutputStream(), true);`
  - `BufferedReader in = new BufferedReader(`
    `new InputStreamReader(s.getInputStream()));`
- Dengan SocketUtils (untuk Socket s)
  - `PrintWriter out = SocketUtils.getWriter(s);`
  - `BufferedReader in = SocketUtils.getReader(s);`

Eksepsi

- IOException
  - Interupsi atau masalah tidak diharapkan lainnya
    - Client menutup koneksi dikarenakan error penulisan (*writing*) , tetapi tidak sebabkan error saat pembacaan (*reading*): Stream<String> dari lines hanya selesai, dan null dikembalikan dari readLine
- Catatan
  - ServerSocket mengimplementasikan **AutoCloseable**, sehingga dapat digunakan ide *try-with-resources* (sebagaimana dibahas pada bagian IO file)
    - `try(ServerSocket listener = new ServerSocket(…)) { … }`

Pemanasan:
Server Ber-Thread Tunggal

Kelas Basis Server Jaringan Berthread Tunggal

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

/** Titik permulaan bagi server jaringan: Server Generik. */

public abstract class NetworkServer {
  private int port;

  /** Membangun server pada port tertentu. Akan terus menerima koneksi,
   *  mengirimkan masing-masing ke handleConnection sampai server dimatikan
   *  (misal Control-C di jendela startup) atau memanggil System.exit()
   *  dari handleConnection atau di tempat lain dalam kode Java).
   */

  public NetworkServer(int port) {
    this.port = port;
  }
```

Server Jaringan Generik (Lanj.)

```
// Monitor port untuk koneksi. Setiap kali koneksi terbangun,
// serahkan Socket yang dihasilkan ke handleConnection.

public void listen() {
  try(ServerSocket listener = new ServerSocket(port)) {
    Socket socket;
    while(true) {  // Jalan sampai mati
      socket = listener.accept();
      handleConnection(socket);
    }
  } catch (IOException ioe) {
    System.out.println("IOException: " + ioe);
    ioe.printStackTrace();
  }
}
```

Server Jaringan Generik (Lanj.)

```
  /** Inilah metode yang menyediakan perilaku server karena itu
   *  menentukan apa yang dikerjakan dengan socket yang dihasilkan.
   *  Override metode ini dalam server yang dibuat.</b>
   */

  protected abstract void handleConnection(Socket socket) throws IOException;

  /** Dapatkan port dimana server mendengarkan. */

  public int getPort() {
    return(port);
  }
}
```

Menggunakan Server Jaringan

```
public class NetworkServerTest extends NetworkServer {
  public NetworkServerTest(int port) { super(port); }

  @Override
  protected void handleConnection(Socket socket) throws IOException{
    PrintWriter out = SocketUtils.getWriter(socket);
    BufferedReader in = SocketUtils.getReader(socket);
    System.out.printf("Server Generik: Ada koneksi dari %s%n" +
                      "dengan baris pertama '%s'.%n",
                      socket.getInetAddress().getHostName(), in.readLine());
    out.println("Server Generik");
    socket.close();
  }
```

Menggunakan Server Jaringan (Lanj.)

```
  public static void main(String[] args) {
    int port = 8080;

    try {
      port = Integer.parseInt(args[0]);
    } catch(NumberFormatException|ArrayIndexOutOfBoundsException e) {}

    NetworkServerTest tester = new NetworkServerTest(port);
    tester.listen();
  }
}
```

Server Jaringan: Hasil

- Menerima koneksi dari web browser
  - Misalnya program di atas berjalan pada port 80 di mesin server.com:

```
> Java NetworkServerTest 80
```

  - Kemudian, menggunakan Web browser standard pada client.com request ke http://server.com/foo/bar, menghasilkan teks berikut pada server.com:

```
Generic Network Server:
got connection from client.com
with first line 'GET /foo/bar HTTP/1.1'
```

Kelas Dasar untuk Server Ber-Thread Banyak

(Multithreaded Server)

Kelas Dasar Server Ber-Thread Banyak

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

public class MultithreadedServer {
  private int port;

  public MultithreadedServer(int port) { this.port = port; }

  public int getPort() {  return(port);  }
```

MultithreadedServer.java (Lanj.)

```
public void listen() {
  int poolSize = 50 * Runtime.getRuntime().availableProcessors();
  ExecutorService tasks = Executors.newFixedThreadPool(poolSize);
  try(ServerSocket listener = new ServerSocket(port)) {
    Socket socket;
    while(true) {  // Jalan sampai mati
      socket = listener.accept();
      tasks.execute(new ConnectionHandler(socket));
    }
  } catch (IOException ioe) {
    System.err.println("IOException: " + ioe);
    ioe.printStackTrace();
  }
}
```

Inner class yang memiliki metode run memanggil balik handleConnection dari kelas ini.

EchoServer, contoh berikutnya, akan memperluas kelas ini untuk membuat server HTTP.

MultithreadedServer.java (Lanj.: Inner Class)

```
private class ConnectionHandler implements Runnable {
  private Socket connection;

  public ConnectionHandler(Socket socket) {
    this.connection = socket;
  }
  public void run() {
     try { handleConnection(connection); }
     catch(IOException ioe) { System.err.println("IOException: " + ioe); }
  }
}
```

MultithreadedServer.java (Lanj.)

```
/** Inilah metode yang menyediakan perilaku server karena ia
 *  menentukan apa yang dikerjakan dengan socket yang dihasilkan.
 *  Override metode ini dalam server yang dibuat.</b>
 */

protected abstract void handleConnection(Socket connection)
    throws IOException;
```

Server HTTP Berthread Banyak “Simpel”

Request & Respon HTTP

Request

```
GET /~gates/ HTTP/1.1
Host: www.mainhost.com
Connection: close
Header3: ...
...
HeaderN: ...
Blank Line
```

- Semua header request opsional kecuali untuk Host (diperlukan bagi HTTP/1.1)
- Jika kita mengirimkan HEAD bukan GET, server mengembalikan header HTTP yang sama, tetapi bukan dokumen.

Response

```
HTTP/1.1 200 OK
Content-Type: text/html
Header2: ...
...
HeaderN: ...
Blank Line
<!DOCTYPE ...>
<html>
…
</html>
```

- Semua header respon opsional kecuali Content-Type

HTTP Server Sederhana

- Gagasan
  1. Baca baris yang dikirim oleh browser, simpan ke dalam List
     - Gunakan readLine per baris sampai baris kosong
       - Eksepsi: dengan request POST kita harus membaca baris tambahan
  2. Kirimkan baris respon HTTP (misal "HTTP/1.1 200 OK")
  3. Kirimkan baris Content-Type kemudian baris kosong (*blank line*)
     - Ini mengindikasikan jenis file yang akan dikembalikan (HTML dalam kasus ini)
  4. Kirimkan file HTML yang menunjukkan baris-baris yang dikirimkan
     - Letakkan input dalam seksi <pre> di dalam body
  5. Tutup koneksi

EchoServer.java

```
 /** Server HTTP simpel yang membangkitkan suatu halaman web
  *  menampilkan semua data yang diterima dari
  *  client (biasanya web browser). */

public class EchoServer extends MultithreadedServer {
  public EchoServer(int port) {  super(port); }

  public static void main(String[] args) {
    int port = 8080;
    try {  port = Integer.parseInt(args[0]); }
    catch(NumberFormatException|ArrayIndexOutOfBoundsException e) { }
    EchoServer server = new EchoServer(port);
    server.listen();
  }
```

EchoServer.java: Membaca Request

```
@Override
public void handleConnection(Socket socket) throws IOException{
  String serverName = "Multithreaded EchoServer";
  PrintWriter out = SocketUtils.getWriter(socket);
  BufferedReader in = SocketUtils.getReader(socket);
  List<String> inputLines = new ArrayList<>();
  String line;

  while((line = in.readLine()) != null) {
    inputLines.add(line);
    if (line.isEmpty()) { // Blank line.
      if (WebUtils.isUsingPost(inputLines)) {
        inputLines.add(WebUtils.postData(in));
      }
      break;
    }
  }
```

EchoServer.java: Mengirimkan Balasan

```
    WebUtils.printHeader(out, serverName);
    for (String inputLine: inputLines) {
      out.println(inputLine);
    }
    WebUtils.printTrailer(out);
    socket.close();
  }
```

WebUtils.java

```
public static void printHeader(PrintWriter out, String serverName) {
  out.println ("HTTP/1.1 200 OK\r\n" +
     "Server: " + serverName + "\r\n" +  "Content-Type: text/html\r\n" +
     "\r\n" + "<!DOCTYPE html>\n" +  "<html lang=\"en\">\n" +
     "<head>\n" + "  <meta charset=\"utf-8\"/>\n" +
     "  <title>" + serverName + " Results</title>\n" +
     "</head>\n" + "\n" +
     "<body bgcolor=\"#fdf5e6\">\n" +
     "<h1 align=\"center\">" + serverName + " Results</h1>\n" +
     "Here are the request line and request headers\n" +
     "sent by your browser:\n" +
     "<pre>");
}
```

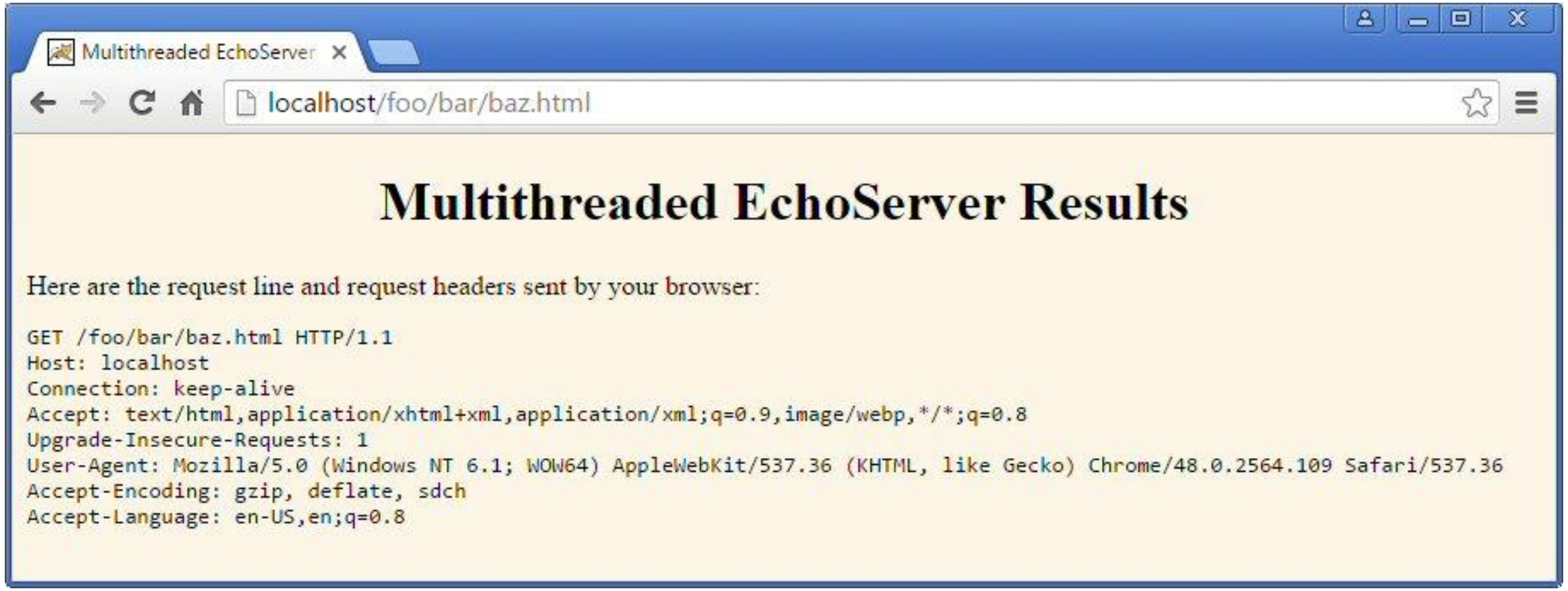
WebUtils.java (Lanj.)

```
public static void printTrailer(PrintWriter out) {
  out.println("</pre></body></html>\n");
}

public static boolean isUsingPost(List<String> inputs) {
  return(inputs.get(0).toUpperCase().startsWith("POST"));
}

/**  submisi POST mempunyai satu baris ekstra di ujung, setelah baris kosong,
 *   dan TIDAK dihentikan oleh CR. Abaikan post bnyk baris, seperti file upload.
 */
public static String postData(BufferedReader in) throws IOException {
  char[] data = new char[1000];  // Anggap maks. 1000 karakter
  int chars = in.read(data);
  return(new String(data, 0, chars));
}
```

Aksi EchoServer



Pertanyaan?

Rangkuman

- Membuat ServerSocket; menentukan nomor port
  - Panggil **accept** untuk menunggu koneksi dari client
  - Menerima kembalian dari obyek Socket (kelas sama yang digunakan di aplikasi client)
- Request browser:
  - Baris GET, POST atau HEAD
  - 0 atau lebih header request
  - Baris kosong
  - Satu baris tambahan (data query) untuk request POST saja
- Respon server HTTP:
  - Baris status (HTTP/1.1 200 OK),
  - Content-Type (dan header respon lainnya)
  - Baris kosong
  - Dokumen
- Selalu buat server ber-thread banyak (*multi-threaded*)
  - Gunakan MultithreadedServer sebagai titik awal (*starting point*) = template ☺

Contoh: Server Membalik String

```
1  import java.net.*;
2  import java.io.*;
3
4  public class RevServer
5  {
6      public static void main(String[] args)throws Exception {
7          int count=1;
8          System.out.println("Server is running.....................");
9          ServerSocket ss=new ServerSocket(10000);
10         while(true) {
11             new RevThread(ss.accept(),count).start();
12             System.out.println(count+" client connected");
13             count++;
14         }
15     }
16 }
```

```
18 class RevThread extends Thread {
19     Socket s=null;
20     int n;
21
22     public RevThread(Socket socket,int count) {
23         s=socket;
24         n=count;
25     }
26
27     public void run() {
28         try {
29             while(true) {
30                 System.out.println("receiving from client "+n);
31                 DataInputStream din=new DataInputStream(s.getInputStream());
32                 String str=din.readUTF();
33                 System.out.println("processing data of Client "+n);
34                 StringBuffer rev=new StringBuffer();
35                 rev=rev.append(str);
36                 rev=rev.reverse();
37                 String revStr=new String(rev);
38                 System.out.println("sending to client "+n);
39                 DataOutputStream dout=new DataOutputStream(s.getOutputStream());
40                 dout.writeUTF(revStr);
41             }
42         }
43         catch(IOException e) { System.out.println(e); }
44     }
45 }
```

Contoh: Client dari Server Membalik String

```
public class RevClient {
 public static void main(String[] args) throws Exception {
  Socket s=new Socket("127.0.0.1",10000);
  if(s.isConnected()) {  System.out.println("Connected to Server....");  }
  while(true)  {
   System.out.println("Enter String to reverse:");
   DataInputStream in=new DataInputStream(System.in);
   String str=in.readLine();
   DataOutputStream dout=new DataOutputStream(s.getOutputStream());
   dout.writeUTF(str);

   DataInputStream din=new DataInputStream(s.getInputStream());
   String rev=din.readUTF();
   System.out.println("Reversed String:\t"+rev);
  }
 }
}
```