



core
WEB
programming

Network Programming: Servers

© 2001-2003 Marty Hall, Larry Brown <http://www.corewebprogramming.com>

Agenda

- **Steps for creating a server**
 1. Create a ServerSocket object
 2. Create a Socket object from ServerSocket
 3. Create an input stream
 4. Create an output stream
 5. Do I/O with input and output streams
 6. Close the socket
- **A generic network server**
- **Accepting connections from browsers**
- **Creating an HTTP server**
- **Adding multithreading to an HTTP server**

Steps for Implementing a Server

1. Create a ServerSocket object

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

2. Create a Socket object from ServerSocket

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

- Note that it is quite common to have doSomethingWith spin off a separate thread

3. Create an input stream to read client input

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

3

Network Programming: Servers

www.corewebprogramming.com

Steps for Implementing a Server

4. Create an output stream that can be used to send info back to the client.

```
// Last arg of true means autoflush stream  
// when println is called  
PrintWriter out =  
    new PrintWriter(server.getOutputStream(), true)
```

5. Do I/O with input and output Streams

- Most common input: readLine
- Most common output: println

6. Close the socket when done

```
server.close();
```

- This closes the associated input and output streams.

4

Network Programming: Servers

www.corewebprogramming.com

A Generic Network Server

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

/** A starting point for network servers. */

public class NetworkServer {
    protected int port, maxConnections;

    /** Build a server on specified port. It will continue
     * to accept connections (passing each to
     * handleConnection) until an explicit exit
     * command is sent (e.g. System.exit) or the
     * maximum number of connections is reached. Specify
     * 0 for maxConnections if you want the server
     * to run indefinitely.
     */

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

5

Network Programming: Servers

www.corewebprogramming.com

A Generic Network Server (Continued)

```
/** Monitor a port for connections. Each time one
 * is established, pass resulting Socket to
 * handleConnection.
 */

public void listen() {
    int i=0;
    try {
        ServerSocket listener = new ServerSocket(port);
        Socket server;
        while((i++ < maxConnections) ||
            (maxConnections == 0)) {
            server = listener.accept();
            handleConnection(server);
        }
    } catch (IOException ioe) {
        System.out.println("IOException: " + ioe);
        ioe.printStackTrace();
    }
}
```

6

Network Programming: Servers

www.corewebprogramming.com

A Generic Network Server (Continued)

```
...
protected void handleConnection(Socket server)
    throws IOException{
    BufferedReader in =
        SocketUtil.getBufferedReader(server);
    PrintWriter out =
        SocketUtil.getPrintWriter(server);
    System.out.println
        ("Generic Network Server:\n" +
         "got connection from " +
         server.getInetAddress().getHostName() + "\n" +
         "with first line '" +
         in.readLine() + "'");
    out.println("Generic Network Server");
    server.close();
}
}
```

- Override `handleConnection` to give *your* server the behavior you want.

Using Network Server

```
public class NetworkServerTest {
    public static void main(String[] args) {
        int port = 8088;
        if (args.length > 0) {
            port = Integer.parseInt(args[0]);
        }
        NetworkServer server = new NetworkServer(port, 1);
        server.listen();
    }
}
```

Network Server: Results

- **Accepting a Connection from a WWW Browser**

- Suppose the above test program is started up on port 8088 of `server.com`:

```
server> java NetworkServerTest
```

- Then, a standard Web browser on `client.com` requests `http://server.com:8088/foo/`, yielding the following back on `server.com`:

```
Generic Network Server:  
got connection from client.com  
with first line 'GET /foo/ HTTP/1.0'
```

HTTP Requests and Responses

- **Request**

```
GET /~gates/ HTTP/1.0  
Header1: ...  
Header2: ...  
...  
HeaderN: ...  
Blank Line
```

- All request headers are optional except for `Host` (required only for HTTP/1.1 requests)
- If you send `HEAD` instead of `GET`, the server returns the same HTTP headers, but no document

- **Response**

```
HTTP/1.0 200 OK  
Content-Type: text/html  
Header2: ...  
...  
HeaderN: ...  
Blank Line  
<!DOCTYPE ...>  
<HTML>  
...  
</HTML>
```

- All response headers are optional except for `Content-Type`

A Simple HTTP Server

- **Idea**

1. Read all the lines sent by the browser, storing them in an array
 - Use `readLine` a line at a time until an empty line
 - Exception: with POST requests you have to read some extra data
2. Send an HTTP response line (e.g. "HTTP/1.0 200 OK")
3. Send a Content-Type line then a blank line
 - This indicates the file type being returned (HTML in this case)
4. Send an HTML file showing the lines that were sent
5. Close the connection

EchoServer

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

/** A simple HTTP server that generates a Web page
 *  * showing all of the data that it received from
 *  * the Web client (usually a browser). */

public class EchoServer extends NetworkServer {
    protected int maxInputLines = 25;
    protected String serverName = "EchoServer 1.0";

    public static void main(String[] args) {
        int port = 8088;
        if (args.length > 0)
            port = Integer.parseInt(args[0]);
        EchoServer echoServer = new EchoServer(port, 0);
        echoServer.listen();
    }

    public EchoServer(int port, int maxConnections) {
        super(port, maxConnections);
    }
}
```

EchoServer (Continued)

```
public void handleConnection(Socket server)
    throws IOException{
    System.out.println(serverName + ": got connection from " +
        server.getInetAddress().getHostName());
    BufferedReader in = SocketUtil.getBufferedReader(server);
    PrintWriter out = SocketUtil.getPrintWriter(server);
    String[] inputLines = new String[maxInputLines];
    int i;
    for (i=0; i<maxInputLines; i++) {
        inputLines[i] = in.readLine();
        if (inputLines[i] == null) // Client closes connection
            break;
        if (inputLines[i].length() == 0) { // Blank line
            if (usingPost(inputLines)) {
                readPostData(inputLines, i, in);
                i = i + 2;
            }
            break;
        }
    }
    ...
}
```

13

Network Programming: Servers

www.corewebprogramming.com

EchoServer (Continued)

```
    printHeader(out);
    for (int j=0; j<i; j++)
        out.println(inputLines[j]);
    printTrailer(out);
    server.close();
}

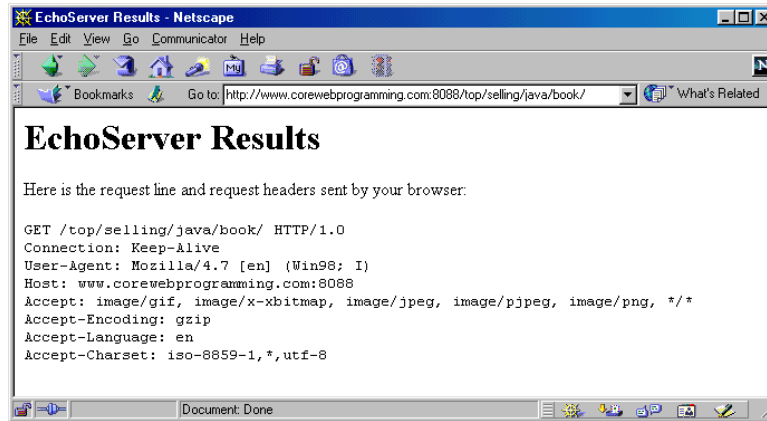
private void printHeader(PrintWriter out) {
    out.println("HTTP/1.0 200 Document follows\r\n" +
        "Server: " + serverName + "\r\n" +
        "Content-Type: text/html\r\n" +
        "\r\n" +
        "<!DOCTYPE HTML PUBLIC " +
        "\"-//W3C//DTD HTML 4.0//EN\">\n" +
        "<HTML>\n" +
        ...
        "</HEAD>\n");
}
...
}
```

14

Network Programming: Servers

www.corewebprogramming.com

EchoServer in Action



EchoServer shows data sent by the browser

Adding Multithreading

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

/** A multithreaded variation of EchoServer. */

public class ThreadedEchoServer extends EchoServer
    implements Runnable {
    public static void main(String[] args) {
        int port = 8088;
        if (args.length > 0)
            port = Integer.parseInt(args[0]);
        ThreadedEchoServer echoServer =
            new ThreadedEchoServer(port, 0);
        echoServer.serverName = "Threaded Echo Server 1.0";
        echoServer.listen();
    }

    public ThreadedEchoServer(int port, int connections) {
        super(port, connections);
    }
}
```


Adding Multithreading (Continued)

```
public void handleConnection(Socket server) {
    Connection connectionThread =
        new Connection(this, server);
    connectionThread.start();
}

public void run() {
    Connection currentThread =
        (Connection)Thread.currentThread();
    try {
        super.handleConnection(currentThread.serverSocket);
    } catch(IOException ioe) {
        System.out.println("IOException: " + ioe);
        ioe.printStackTrace();
    }
}
}
```

Adding Multithreading (Continued)

```
/** This is just a Thread with a field to store a
 * Socket object. Used as a thread-safe means to pass
 * the Socket from handleConnection to run.
 */

class Connection extends Thread {
    protected Socket serverSocket;

    public Connection(Runnable serverObject,
                     Socket serverSocket) {
        super(serverObject);
        this.serverSocket = serverSocket;
    }
}
```

Summary

- **Create a ServerSocket; specify port number**
- **Call `accept` to wait for a client connection**
 - Once a connection is established, a Socket object is created to communicate with client
- **Browser requests consist of a GET, POST, or HEAD line followed by a set of request headers and a blank line**
- **For the HTTP server response, send the status line (HTTP/1.0 200 OK), Content-Type, blank line, and document**
- **For improved performance, process each request in a separate thread**



core
WEB
programming

Questions?