



Applicazioni client/server in Java

Ing. Nadia Ranaldo

1



Un semplice server Web (1)

- Gestire una richiesta HTTP
- Accettarla ed analizzarla
- Accedere al file richiesto dal file system del server
 - Il client richiede un file che potrebbe non essere presente
- Creare un messaggio di risposta HTTP contenente righe di intestazione ed il file richiesto
- Inviare la risposta direttamente al client
- Non si considera la gestione delle eccezioni
- Una volta inviata la risposta, la connessione viene chiusa
 - HTTP/1.0 connessioni non persistenti

2



Un semplice server Web (2)

- Il server riceve dal client un messaggio di richiesta HTTP
- Il nome del file da restituire è estratto dalla prima linea (linea di richiesta)

GET filename HTTP/1.0

- E' la seconda parola della linea
- Utilizzare un oggetto di tipo **StringTokenizer** che permette di estrarre le singole parole (chiamate **token**) da una stringa passata come parametro al costruttore
- I caratteri che delimitano i token possono essere specificati dall'utente
 - Di default sono ' ' \t \n \r \f
- `public boolean hasMoreTokens();`
- `public String nextToken();`

3



Un semplice server Web (3)

- Si accede al file di nome **filename** creando un'istanza della classe **File**
- Se il file non esiste si deve costruire un messaggio di risposta contenente nella linea di stato il codice di stato 404
 - L'entity body viene utilizzato per contenere un messaggio di errore da visualizzare nel browser
- Se il file esiste, occorre inviare un messaggio di risposta contenente nella linea di stato il codice di stato 200
 - L'entity body viene utilizzato per contenere il file
- Si utilizza un **DataOutputStream** collegato allo stream di output della socket

4

Un semplice server Web (4)

```
import java.io.*;
import java.net.*;
import java.util.*;
public class WebServer {
    final static String CRLF = "\r\n";
    public static void main(String argv[]) throws
    Exception {
        // La porta su cui restare in ascolto è letta dalla linea di comando
        int port = (new Integer(argv[0])).intValue();
        // Definizione della socket su cui rimanere in attesa di connessioni
        ServerSocket socket = new ServerSocket(port);
        Socket connection;
        while (true) {
            // Attende una richiesta di connessione TCP
            connection = socket.accept();
            ... // gestione della richiesta
        }
    }
}
```

5

Un semplice server Web (5)

```
// Ottiene un riferimento agli stream di input ed output della socket
BufferedReader br = new BufferedReader (new
    InputStreamReader(connection.getInputStream()));
DataOutputStream os = new DataOutputStream
    (connection.getOutputStream());
// Legge la linea di richiesta dal messaggio HTTP
String requestLine = br.readLine();
System.out.println(requestLine);
// visualizza le linee di intestazioni del messaggio di richiesta
String headerLine = null;
while ((headerLine = br.readLine()).length() != 0)
    { System.out.println(headerLine); }
// Estrae il nome del file dalla linea di richiesta
StringTokenizer tokens = new StringTokenizer(requestLine);
// Salta il primo token che si suppone sia il metodo GET
tokens.nextToken();
String fileName = tokens.nextToken();
```

6

Un semplice server Web (6)

```
// Pone "." davanti a filename in modo da accedere al file nella
// directory corrente
fileName = "." + fileName;
// Apre il file richiesto
File file = new File(fileName);
// costruisce il messaggio di risposta
boolean exists = false;
String statusLine = null;
String contentTypeLine = null;
String contentLength = null;
String entityBody = null;
byte[] fileBytes = null;
exists = file.exists();
if (!exists) {
    statusLine = "HTTP/1.0 404 Not Found" + CRLF;
    contentTypeLine = "Content-Type: text/html" + CRLF;
    entityBody="<HTML>" + "<HEAD><TITLE>Not Found</TITLE></HEAD>"
        + "<BODY>"+fileName+" Not Found</BODY></HTML>";
} else {
```

7

Un semplice server Web (7)

```
int numofBytes = (int)file.length();
fileBytes = new byte[numofBytes];
FileInputStream fis = new FileInputStream(file);
fis.read(fileBytes);
fis.close();
statusLine = "HTTP/1.0 200 OK" + CRLF;
contentTypeLine ="Content-Type: "+contentType(fileName)+CRLF;
contentLength = "Content-Length: "+numofBytes+CRLF;
}
// Invia la linea di stato
os.writeBytes(statusLine);
// Invia la linea di intestazione content type
os.writeBytes(contentTypeLine);
// Invia la linea di intestazione content length
if (exists) os.writeBytes(contentLength);
```

8

Un semplice server Web (8)

```
// Invia una linea vuota per individuare la fine delle linee
// di intestazione
os.writeBytes(CRLF);
// Invia l'entity body
if (exists) {
    os.write(fileBytes);
} else {
    os.writeBytes(entityBody) ;
}
// chiude gli stream e la socket
os.close();
br.close();
connection.close();
}
```

9

Un semplice server Web (9)

```
private static String contentType(String fileName) {
    if(fileName.endsWith(".htm") || fileName.endsWith(".html")) {
        return "text/html";
    }
    if(fileName.endsWith(".ram") || fileName.endsWith(".ra")) {
        return "audio/x-pn-realaudio";
    }
    if(fileName.endsWith(".jpg")) {
        return "image/jpeg";
    }
    if(fileName.endsWith(".gif")) {
        return "image/gif";
    }
    return "application/octet-stream" ;
}
```

10

Client per testare il Server Web

- Porre nella directory corrente i file html, gif, etc. che si vogliono rendere disponibili
- Due modi per testare il server Web

1. Utilizzare telnet

```
telnet nomeHost porta
```

```
GET /fileName HTTP/1.0
```

2. Utilizzare un browser

```
URL: http://hostname:porta/filename
```

11