EDA095
HTTP

Pierre Nugues

Lund University
http://cs.lth.se/pierre_nugues/

April 8, 2015

Covers: Chapter 6, *Java Network Programming*, 4rd ed., Elliotte Rusty Harold

Client and Server Communications Using HTTP

The communication between a server and a client in its simplest form starts with a TCP connection from the client to port 80.

Then we have the exchange:

- The client sends a request
- The server sends a response.

Both request and response messages feature a header that consists of parameter-value pairs

In addition:

- The request consists of a command word (HTTP method) to identify the request, parameters, and possibly other data.

- The response consists of a response code and objects to be displayed or rendered by the client

HTTP Request

The request behind the URL `http://cs.lth.se/pierre_nugues` consists of:

1. HTTP method, URL, version

   `GET /pierre_nugues HTTP/1.1`

2. Sequence of parameter names (46 types) followed by ':' and values – pairs `Name: Value`

   `Accept: text/plain`

   `...`
   `Host: cs.lth.se      #Mandatory`
   `User-Agent: Mozilla/4.0`

3. Empty line: `\r\n`

4. Possibly a message body (data) whose size is given by the `Content-Length` attribute

RFC 2616 (`http://www.ietf.org/rfc/rfc2616.txt`)

HTTP Response

Servers send a response: header followed by data

1. Protocol, status code, textual phrase

   `HTTP/1.0 200 OK`

2. Sequence of parameter names followed by ':' and values

   `Date: Wed, 28 Mar 2007 12:12:54 GMT`
   `Server: Apache/2.0.52 (sparc-sun-solaris2.8)`
   `...`
   `Connection: close`

3. Empty line: \r\n

4. Data

   `<html>`
   `...`
   `</html>`

HTTP Development Tools for Firefox, Chrome, etc.

HTTP Response Codes

A few response codes:

- 2xx: Success

- 3xx: Redirection

- 4xx: Client error

- 5xx: Server error

Code	Java	Code	Java
200 OK	HTTP_OK	201 Created	HTTP_CREATED
202 Accepted	HTTP_ACCEPTED	204 No Content	HTTP_NO_CONTENT
301 Moved Permanently	HTTP_MOVED_PERM	301 Moved Temporarily	HTTP_MOVED_TEMP
400 Bad Request	HTTP_BAD_REQUEST	401 Unauthorized	HTTP_UNAUTHORIZED
500 Internal Server Error	HTTP_SERVER_ERROR	502 Bad Gateway	HTTP_BAD_GATEWAY

Parameter: Connection: `keep-alive`

- HTTP 1.0: A new connection for each request
- HTTP 1.1: Possibility to reuse an existing TCP socket with Connection: keep-alive
- Connection: keep-alive means that the client is willing to keep the same socket

Cookies

A cookie is a small piece of information sent by the server and saved by the client

The client will send the cookie back to the server in subsequent accesses

A way to:

- Store server information on the client
- Spy the clients

To set a cookie, the server sends:

```
Set-Cookie: fe_typo_user=4ca926632655c2ecb1a12c66eee5ad8f;
path=/
```

When the client accesses the same server, it sends:

```
Cookie: fe_typo_user=4ca926632655c2ecb1a12c66eee5ad8f
```

Cookie Properties

A cookie has a name and a value.
Some properties:

- domain
- path
- expires
- connection type: http, secure.

In its simplest form, the client does not send data

It just requests the content of the URL file

For HTTP, the corresponding methods are GET or POST.

The method is sent automatically by the URL class and is hidden to the programmer.

We will review the steps to get data from simple to more complex

The URL class uses GET by default, hides the header details, and receives data

Client Methods

The client uses eight possible "methods":

- `GET`: retrieves information identified by the Request-URI
- `POST`: sends data to the identified resource
- `PUT`: stores the resource identified by the Request-URI
- `DELETE`: deletes the resource identified by the Request-URI
- `HEAD`. Same as `GET` but returns a response consisting of headers (without a message body)
- `OPTIONS`: returns the methods supported by the server
- `TRACE`: sends back the header to the client.
- `CONNECT`: reserved name to connect to a TCP/IP tunnel

HTTP servers must implement at least `GET` and `HEAD`.

An Elementary Form

A text box: `My text`

Radio buttons:

- ⊙ FM
- ◯ LW
- ◯ SW

A drop-down list:

`Low ▾`

`Send!` `Cancel`

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Content-Type" content="text/html;
      charset=UTF-8"/>
    <title>Testing HTML forms</title>
  </head>
  <body>
    <h1>An Elementary Form</h1>
    <form action="http://localhost:25001/program.sh"
      method="post">
      <p>A text box: <input type="text" name="name" value=""
        size="30"/></p>
      <hr/>
```

```
<p>Radio buttons:</p>
<ul>
  <li><input type="radio" name="buttons" value="FM"
    checked="checked"/>FM</li>
  <li><input type="radio" name="buttons" value="LW"/>
    LW
  </li>
  <li><input type="radio" name="buttons" value="SW"/>
    SW
  </li>
</ul>
```

```html
        <p>A drop-down list:</p>
        <p>
          <select name="dropdown">
            <option selected="selected">Low</option>
            <option>Medium</option>
            <option>High</option>
          </select>
        </p>
        <hr/>
        <p>
          <input type="submit" value="Send!"/>
          <input type="reset" value="Cancel"/>
        </p>
      </form>
  </body>
</html>
```

HTTP Request with POST

To send data to URL `http://cs.lth.se/pierre_nugues/prog.sh`, the request consists of:

1. HTTP method, URL, version

   `POST /pierre_nugues/prog.sh HTTP/1.0`

2. Sequence of parameter names (46 types) followed by ':' and values – pairs Name: Value

   `Accept: text/plain`
   `...`
   `Host: cs.lth.se`
   `User-Agent: Mozilla/4.0`

3. Empty line: `\r\n`

4. Data length should match the `Content-Length` parameter

RFC 2616 (`http://www.ietf.org/rfc/rfc2616.txt`)

An Example of HTTP Request with POST

```
POST /program.sh HTTP/1.1
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_5_6;
AppleWebKit/528.16 (KHTML, like Gecko) Version/4.0
Safari/528.16
Content-Type: application/x-www-form-urlencoded
Accept: application/xml,application/xhtml+xml,text/html;q=0.9,
text/plain;q=0.8,image/png,*/*;q=0.5
Origin: file://
Accept-Language: fr-fr
Accept-Encoding: gzip, deflate
Content-Length: 36
Connection: keep-alive
Host: localhost:25001

name=My+text&buttons=FM&dropdown=Low
```

```
URL: http://localhost:25001/program.sh?name=My+text&buttons=FM
&dropdown=Low
GET /program.sh?name=My+text&buttons=FM&dropdown=Low HTTP/1.1
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_5_6;
 fr-fr) AppleWebKit/528.16 (KHTML, like Gecko) Version/4.0
 Safari/528.16
Accept: application/xml,application/xhtml+xml,text/html;q=0.9,
text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: fr-fr
Accept-Encoding: gzip, deflate
Connection: keep-alive
Host: localhost:25001
```

Using GET:

```
http://control.lth.se/?mact=Search%2Ccntnt01%2Cdosearch
%2C0&cntnt01returnid=56&cntnt01searchinput=Nugues&submit=Submi

GET /?mact=Search%2Ccntnt01%2Cdosearch%2C0
&cntnt01returnid=56&cntnt01searchinput=Nugues&
submit=Submit HTTP/1.1
Host: control.lth.se
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:29
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,
Accept-Language: fr,fr-fr;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
DNT: 1
Referer: http://control.lth.se/
Cookie: CMSSESSIDafa92467=qfhf9a02nekjn5tc9q69gof0n0
Connection: keep-alive
```

Sending Data to a Server: A Real Example with POST

Using POST:

```
http://cs.lth.se/sok/
```

```
POST /sok/ HTTP/1.1
Host: cs.lth.se
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:29.0)
Gecko/20100101 Firefox/29.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;
q=0.8
Accept-Language: fr,fr-fr;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
DNT: 1
Referer: http://cs.lth.se/
Cookie: fe_typo_user=d9c25aba11b4cdca162da148b69d3511
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 45
s=1&so=1&i=sv&category=cs&q=Nugues&x=S%C3%B6k
```