In this chapter's application, the connector parses HTTP request headers and enables a servlet to obtain headers, cookies, parameter names/values, etc.

HTTP request objects are represented by the HttpRequest class, which implements javax.servlet.http.HttpServletRequest.

the connector must parse all values that can be obtained from the HTTP request.

parsing an HTTP request involves expensive string and other operations, and the connector can save lots of CPU cycles if it parses only values that will be needed by the servlet.

if the servlet does not need any request parameter (i.e. it does not call the getParameter, getParameterMap, getParameterNames, or getParameterValues methods of javax.servlet.http.HttpServletRequest), the connector does not need to parse these parameters from the query string and or from the HTTP request body.

Tomcat's default connector tries to be more efficient by leaving the parameter parsing until it is really needed by the servlet.

SocketInputStream class is used for reading byte streams from the socket's InputStream.

An instance of SocketInputStream wraps the java.io.InputStream instance returned by the socket's getInputStream method.

The SocketInputStream class provides two important methods: readRequestLine and readHeader.

readRequestLine returns the first line in an HTTP request.

processing byte stream from the socket's input stream means reading from the first byte to the last (and never moves backwards), readRequestLine must be called only once and must be called before readHeader is called.

The return value of readRequestLine is an instance of HttpRequestLine.

readHeader is called to obtain a header name/value pair each time it is called and should be called repeatedly until all headers are read.

the return value of readHeader is an HttpHeader object.

The HttpProcessor class, using its parse method, parses both the request line and headers in an HTTP request.

The values are then assigned to the fields in the HttpProcessor objects.

NOTE: the parse method does not parse the parameters in the request body or query string. This task is left to the HttpRequest objects themselves. Only if the servlet needs a parameter will the query string or request body be parsed.

**public** **void** process(Socket socket) {

~~SocketInputStream~~ input = **new** SocketInputStream(socket.getInputStream(), 2048);

OutputStream output = socket.getOutputStream();

// create HttpRequest object and parse

request = **new** HttpRequest(input);

response = **new** HttpResponse(output);

response.setRequest(request);

response.setHeader("Server", "Pyrmont Servlet Container");

parseRequest(input, output);

parseHeaders(input);

**if** (request.getRequestURI().startsWith("/servlet/")) {

**new** ServletProcessor().process(request, response);

**else** {

**new** StaticResourceProcessor().process(request, response);

}

socket.close();

}

**private** **void** parseRequest(~~SocketInputStream~~ input, OutputStream output)

**throws** IOException, ServletException {

// Parse the incoming request line

input.readRequestLine(requestLine);

String method = **new** String(requestLine.method, 0, requestLine.methodEnd);

String protocol = **new** String(requestLine.protocol, 0, requestLine.protocolEnd);

**int** question = requestLine.indexOf("?");

String uri = **null**;

**if** (question >= 0) {

request.setQueryString(**new** String(requestLine.uri, question + 1, requestLine.uriEnd - question - 1));

uri = **new** String(requestLine.uri, 0, question);

}

**else** {

request.setQueryString(**null**);

uri = **new** String(requestLine.uri, 0, requestLine.uriEnd);

}

String match = ";jsessionid=";

**int** semicolon = uri.indexOf(match);

**if** (semicolon >= 0) {

String rest = uri.substring(semicolon + match.length());

**int** semicolon2 = rest.indexOf(';');

**if** (semicolon2 >= 0) {

request.setRequestedSessionId(rest.substring(0, semicolon2));

rest = rest.substring(semicolon2);

}

**else** {

request.setRequestedSessionId(rest);

rest = "";

}

request.setRequestedSessionURL(**true**);

uri = uri.substring(0, semicolon) + rest;

}

**else** {

request.setRequestedSessionId(**null**);

request.setRequestedSessionURL(**false**);

}

// Normalize URI (using String operations at the moment)

String normalizedUri = normalize(uri);

// Set the corresponding request properties

((HttpRequest) request).setMethod(method);

request.setProtocol(protocol);

**if** (normalizedUri != **null**) {

((HttpRequest) request).setRequestURI(normalizedUri);

}

**else** {

((HttpRequest) request).setRequestURI(uri);

}

**if** (normalizedUri == **null**) {

**throw** **new** ServletException("Invalid URI: " + uri + "'");

}

}

**Parsing Cookies**

On the client’s first request, the Container generates a unique session ID and gives it back to the client with the response. The client sends back the session ID with each subsequent request. The Container sees the ID, finds the matching session, and associates the session with the request.

How do the Client and Container exchange Session ID info?

The simplest and most common way to exchange the info is through cookies.

HTTP Response

HTTP/1.1 200 OK

Set-Cookie: JSESSIONID=0AAB6C8DE415

Content-Type: text/html

Content-Length: 397

Date: Wed, 19 Nov 2003 03:25:40 GMT

<html> ... </html>

Server's “Set-Cookie” is just another header sent in the response.

HTTP Request

POST /select/selectBeerTaste2.do HTTP/1.1

Host: www.wickedlysmart.com

User-Agent: Mozilla/5.0

Cookie: JSESSIONID=0AAB6C8DE415

Accept: text/xml,application/xml,application/xhtml+xml,text/

html;q=0.9,text/plain;q=0.8,video/x-mng,image/png,image/

Client's “Cookie” is another header sent in the request.

Somewhere in your service method you ask for a session, and everything else happens automatically.

HttpSession session = request.getSession();

The Container does virtually all the cookie work!

the Container gets the session ID from a cookie in the request, matches the session ID with an existing session, and associates that session with the current request.

Cookies

Although cookies were originally designed to help support session state, you can use custom cookies for other things.

You can use cookies to exchange name/value String pairs between the server and the client. The server sends the cookie to the client, and the client sends it back with each subsequent request. Session cookies vanish when the client’s browser quits, but you CAN tell a cookie to persist on the client even after the browser shuts down.

One cool thing about cookies is that the user doesn’t have to get involved—the cookie exchange is automatic.

Everything you need to do with cookies has been encapsulated in the Servlet API in three classes:

**javax.servlet.http.HttpServletRequest<<interface>>**

getCookies()

**javax.servlet.http.HttpServletResponse<<interface>>**

addCookie()

**javax.servlet.http.Cookie**

Cookie(String, String)

String getDomain()

intgetMaxAge()

String getName()

String getPath()

booleangetSecure()

String getValue()

voidsetDomain(String)

voidsetMaxAge(int)

voidsetPath(String)

voidsetValue(String)

Creating a new Cookie

Cookie cookie = **new** Cookie(“username”, name);

Setting how long a cookie will live on the client

cookie.setMaxAge(30\*60);

Sending the cookie to the client

response.addCookie(cookie);

Getting the cookie(s) from the client request

Cookie[] cookies = request.getCookies();

// Notice, There’s no getCookie(String) method...

**for** (**int** i = 0; i<cookies.length; i++) {

Cookie cookie = cookies[i];

**if** (cookie.getName().equals(“username”)) {

String userName = cookie.getValue();

out.println(“Hello “ + userName);

**break**;

}

}

**Cookies and headers!**

When you add a header to a response, you pass the name and value Strings as arguments:

response.addHeader(“foo”, “bar”);

But when you add a Cookie to the response, you pass a Cookie object. You set the Cookie name and value in the Cookie constructor.

Cookie cookie = new Cookie(“name”, name);

response.addCookie(cookie);

There’s both a setHeader() and an addHeader() method.

But there’s NOT a setCookie() method. There’s only an addCookie() method!

Cookie: userName=budi; password=pwd;

**private** **static** Cookie[] parseCookieHeader(String header) {

ArrayList cookies = **new** ArrayList();

**while** (header.length() > 0) {

**int** semicolon = header.indexOf(';');

String token = header.substring(0, semicolon);

**if** (semicolon < header.length())

header = header.substring(semicolon + 1);

**else** header = "";

**int** equals = token.indexOf('=');

String name = token.substring(0, equals).trim();

String value = token.substring(equals+1).trim();

cookies.add(**new** Cookie(name, value));

}

**return** ((Cookie[]) cookies.toArray (**new** Cookie [cookies.size ()]));

}