

# Servlet-Based Distributed Systems

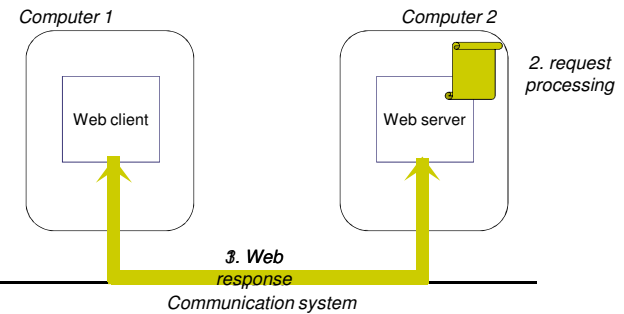
Sara Bouchenak

Sara.Bouchenak@imag.fr

<http://membres-liglab.imag.fr/bouchenak/teaching/>



## Introduction – Web applications



© S. Bouchenak

Distributed systems & Middleware

2

## Introduction – Web applications

- Communication between client and server
  - In a web application, client and server communicate via the HTTP protocol (HyperText Transfer Protocol)
- Web request
  - Client wants to access a remote “resource” available on the server
  - A resource in the WWW is identified and located using a URL
  - A resource can be:
    - a file or a directory
    - a reference to a more complicated object, e.g. a query to a database, a query to a search engine, a program to run
  - Examples of URLs to resources:
    - `http://serverhost/index.html` *a file*
    - `http://serverhost/program?arg1=val1&arg2=val2` *a program*

© S. Bouchenak

Distributed systems & Middleware

3

## What are Servlets

- Servlets are Java programs which run in a server (e.g. a web server)
- They can be remotely requested (e.g. by web clients)
- Servlets that run on a web server build web pages on the fly, and return them to clients
- Building web pages on the fly is useful for a number of reasons:
  - The Web page is based on data submitted by the user
    - Examples: results pages from search engines, programs that process orders for e-commerce sites
  - The data changes frequently
    - Example: news headlines page might build the page dynamically
  - The Web page uses information from corporate databases or other such sources
    - Examples: an on-line store that lists current prices and number of items in stock

© S. Bouchenak

Distributed systems & Middleware

4

## Advantages of Servlets

- Efficiency
  - With traditional CGI, a new process is started for each HTTP request, the overhead of starting the process can dominate the execution time.
  - With servlets, the Java Virtual Machine stays up, and each request is handled by a lightweight Java thread, not a heavyweight operating system process.
  - In traditional CGI, if there are  $N$  simultaneous requests to the same CGI program, then the code for the CGI program is loaded into memory  $N$  times.
  - With servlets, there are  $N$  threads but only a single copy of the servlet class
- Portability
  - Servlets are written in Java and follow a well-standardized API.
  - Servlets can run virtually unchanged on any Servlet server (e.g. Apache Tomcat, IBM's WebSphere Application Server, BEA WebLogic Application Server, etc.)
- Power
  - User session tracking
  - Database connection pools

## Outline

1. Introduction
2. **HTTP basics**
3. Servlet basics
4. Miscellaneous

## HTTP basics

- HTTP: HyperText Transfer Protocol
  - A communication protocol
  - Used to transfer hypertext data on the World Wide Web (WWW)
- A protocol (in the general sense)
  - A set of guidelines and rules that help in governing interactions between two parties
  - Examples:
    - In diplomacy: standards of behavior and ceremony to be observed by diplomats and heads of state in relation to each other
    - Tests and experiments: clinical trial protocol, the method used in a clinical trial of a drug or medical treatment
    - Computing: a set of rules governing communication between computing endpoints

## HTTP basics (2)

- HTTP protocol specifies
  - Requests
  - Responses
  - Headers
- Requests invoke a particular method within the set of HTTP methods
  - HTTP GET method
  - HTTP POST method
  - Other HTTP methods

## HTTP requests

- HTTP: a simple stateless communication protocol
  - An HTTP client (e.g. a web browser) makes a request to an HTTP server
  - The HTTP server (e.g. a web server) responds
  - And the transaction is done.
- Request
  - Client request has the following form:
    - a *method*,
    - target resource address (a URL),
    - HTTP protocol version
  - Example:  
GET /intro.html HTTP/1.0

## HTTP request headers

- After sending the request, the client can send optional header information
- The header tells the server extra information about the request such as:
  - What software the client is running
  - What content types the client understands
- The request ends with an empty line
- This information does not directly pertain to what was requested, but it could be used by the server in generating its response
- Example:  
User-Agent: Mozilla/4.0 (compatible; MSIE 4.0; Windows 95)  
Accept: image/gif, image/jpeg, text/\*, \*/\*

## HTTP responses

- After the server processes the request, it sends an HTTP response
- The first line of the response specifies the following:
  - server's HTTP protocol version
  - a status code (e.g. 200 for successful, 404 for "Not Found")
  - a description of the status code
- Example:  
HTTP/1.0 200 OK

## HTTP response headers

- After sending the status line, the server sends header information
- The header tells the client extra information about the response such as:
  - What software the server is running
  - What content types the server understands
- The server sends a blank line after the header
- Example:  
Date: Saturday, 20-October-2007 03:25:12 GMT  
Server: JavaWebServer/1.1.1  
MIME-version: 1.0  
Content-type: text/html  
Content-length: 1029  
Last-modified: Thursday, 18-October-2007 12:15:35 GMT
- If the request was successful, the requested data is sent as part of the response

## HTTP GET method



- GET method is designed for getting a resource
  - Examples:
    - an HTML/image file,
    - a chart
    - the result of a database query
- GET method can have parameters that better describe what to get
  - Example: an x, y scale for a dynamically created chart
  - Parameters are passed as a sequence of characters appended to the request URL (i.e. a query string)

## HTTP POST method



- POST method is designed for posting information
  - Examples:
    - a credit card number
    - some new chart data
    - information to be stored in a database
- POST method passes all its data as part of the HTTP request body
  - It may need to send megabytes of information
- POST requests should not be bookmarked or emailed (or reloaded)

## Other HTTP methods



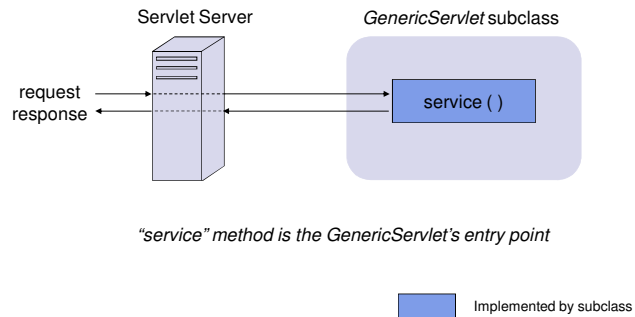
- HEAD method
  - Sent by a client when it wants to see only the headers of the response
  - Examples of use:
    - determine the document's size
    - determine the document's modification time, etc.
- PUT method
  - Place documents directly on the server
- DELETE method
  - Delete documents from the server
- TRACE method
  - Return to the client the exact contents of its request (used for debugging)
- OPTIONS method
  - Ask the server which methods it supports

## Outline



1. Introduction
2. HTTP basics
3. **Servlet basics**
  - **Generic servlets and HTTP servlets**
  - Servlet lifecycle
  - Servlet API
  - A simple example
  - Getting information from requests
  - An HTML form example
4. Miscellaneous

## A generic servlet handling a request

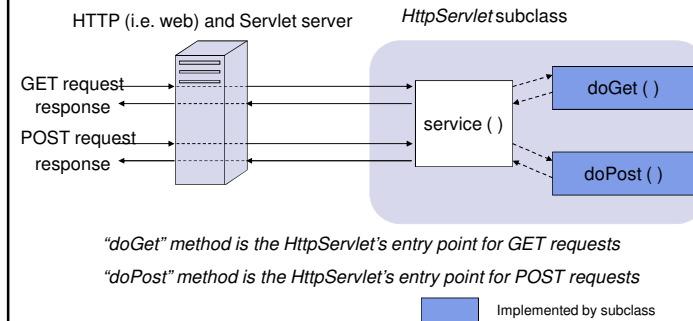


© S. Bouchenak

Distributed systems & Middleware

17

## An HTTP servlet handling GET and POST requests

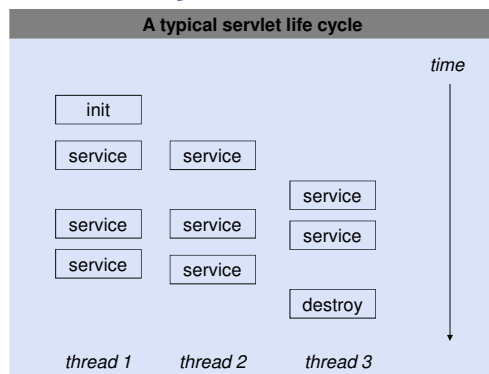


© S. Bouchenak

Distributed systems & Middleware

18

## Servlet lifecycle



© S. Bouchenak

Distributed systems & Middleware

19

## Servlet lifecycle (2)

- A Servlet is an instance of a class which implements the *javax.servlet.Servlet* interface
- A Servlet server initializes a Servlet by
  - loading the Servlet class, and
  - creating an instance of the Servlet by calling the no-args constructor, then
  - calling the Servlet's *init(ServletConfig config)* method
- Servlet's *init(ServletConfig config)* method
  - It performs any necessary initialization of the Servlet and stores the *ServletConfig* object
  - The *ServletConfig* object contains Servlet parameters and a reference to the Servlet's *ServletContext*
  - The *init* method is guaranteed to be called only once during the Servlet's lifecycle

© S. Bouchenak

Distributed systems & Middleware

20

## Servlet lifecycle (3)

- Servlet's *service* method
  - When the Servlet is initialized, its *service(ServletRequest req, ServletResponse res)* method is called for every request to the Servlet
  - The method is called concurrently (i.e. multiple threads may call this method at the same time)
  - It should be implemented in a thread-safe manner
- Servlet's *destroy* method
  - Sometimes, a Servlet may need to be unloaded (e.g. because a new version should be loaded or the server is shutting down)
  - When the Servlet needs to be unloaded, the *destroy()* method is called
  - There may still be threads that execute the *service* method when *destroy* is called, so *destroy* has to be thread-safe
  - All resources which were allocated in *init* should be released in *destroy*
  - This method is guaranteed to be called only once during the Servlet's lifecycle

## Outline

1. Introduction
2. HTTP basics
3. **Servlet basics**
  - Generic servlets and HTTP servlets
  - Servlet lifecycle
  - **Servlet API**
  - A simple example
  - Getting information from requests
  - An HTML form example
4. Miscellaneous

## Servlet API

- Package *javax.servlet*
  - Contains classes to support generic, protocol-independent servlets
  - Some elements of the package:
    - *Servlet* interface:
      - defines methods that all servlets must implement
    - *GenericServlet* abstract class:
      - defines a generic, protocol-independent servlet
    - *ServletRequest* interface:
      - defines an object to provide client request information to a servlet
    - *ServletResponse* interface:
      - defines an object to assist a servlet in sending a response to the client
    - *ServletConfig* interface:
      - defines the information used by a servlet container to pass to a servlet during initialization
    - *ServletContext* interface:
      - defines a set of methods that a servlet uses to communicate with its servlet container, (e.g. write to a log file)

## Servlet API (2)

- Package *javax.servlet.http*
  - Contains classes to support HTTP-based servlets
  - Some elements of the package:
    - *HttpServlet* abstract class:
      - subclass of *GenericServlet*, provides an abstract class to be subclassed to create an HTTP servlet suitable for a Web site
    - *HttpServletRequest* interface:
      - extends the *ServletRequest* interface to provide request information for HTTP servlets
    - *HttpServletResponse* interface:
      - extends the *ServletResponse* interface to provide HTTP-specific functionality in sending a response

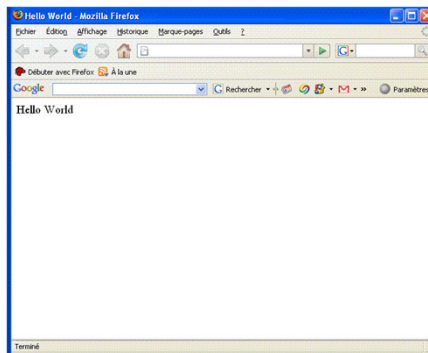
## Outline

1. Introduction
2. HTTP basics
3. **Servlet basics**
  - Generic servlets and HTTP servlets
  - Servlet lifecycle
  - Servlet API
  - **A simple example**
  - Getting information from requests
  - An HTML form example
4. Miscellaneous

## HTML basics

- The most basic type of HTTP servlet generates HTML pages
- HTML (HyperText Markup Language)
  - The predominant markup language for web pages
  - Provides a means to describe the structure of text-based information in a document
  - Denotes certain text as headings, paragraphs, lists, etc.
  - Supplements the text with interactive forms, embedded *images*, and other objects

## An HTML page – A simple example



## An HTML source page

```
<HTML>
  <HEAD>
    <TITLE> Hello World
  </TITLE>
</HEAD>
  <BODY>
    <BIG> Hello World
  </BIG>
</BODY>
</HTML>
```

## A simple HTTP Servlet

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorldServlet extends HttpServlet {

    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {

        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.println("<HTML>");
        out.println("<HEAD> <TITLE> Hello World </TITLE> </HEAD>");
        out.println("<BODY> <BIG> Hello World </BIG> </BODY>");
        out.println("</HTML>");
        out.close();
    }
}
```

## Outline

1. Introduction
2. HTTP basics
3. **Servlet basics**
  - Generic servlets and HTTP servlets
  - Servlet lifecycle
  - Servlet API
  - A simple example
  - **Getting information from requests**
  - An HTML form example
4. Miscellaneous

## Getting information from requests

- A request contains data passed between a client and the servlet
- All requests implement the *ServletRequest* interface
- This interface defines methods for accessing information such as:
  - *String getParameter(String name)*:
    - returns the value of a request parameter as a String, or null if the parameter does not exist
  - *String getProtocol()*:
    - returns the name and version of the protocol the request uses
  - *String getRemoteAddr()*:
    - returns the Internet Protocol (IP) address of the client that sent the request
  - etc.

## Getting information from requests (2)

- Example:
  - A customer wishes to get information about a book.
  - He calls *BookInfoServlet* and includes the identifier of the book in his request
  - For example: `http://host:port/servlets/BookInfoServlet?bookId=1234`

```
public class BookInfoServlet extends HttpServlet {

    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {

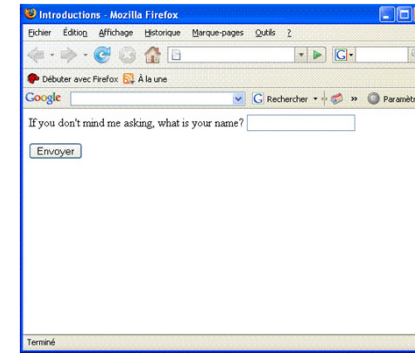
        ...
        String bookId = req.getParameter("bookId");
        if (bookId != null) {
            // Retrieve information about that book
            ...
        }
        ...
    }
    ...
}
```



## Outline

1. Introduction
2. HTTP basics
3. **Servlet basics**
  - Generic servlets and HTTP servlets
  - Servlet lifecycle
  - Servlet API
  - A simple example
  - Getting information from requests
  - **An HTML form example**
4. Miscellaneous

## An HTML form – A simple example



## The HTML source form

```
<HTML>
  <HEAD>
    <TITLE>
      Introductions
    </TITLE>
  </HEAD>
  <BODY>
    <FORM METHOD=GET ACTION="servlet/HelloWorldServlet" >
      If you don't mind me asking, what is your name?
      <INPUT TYPE=TEXT NAME="name">
      <P>
      <INPUT TYPE=SUBMIT>
    </FORM>
  </BODY>
</HTML>
```

## A simple HTTP Servlet handling a form

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorldServlet extends HttpServlet {

    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {

        res.setContentType("text/html");
        PrintWriter out = res.getWriter();

        String name = req.getParameter("name");

        out.println("<HTML>");
        out.println("<HEAD> <TITLE> Hello," + name + "</TITLE></HEAD>");
        out.println("<BODY>");
        out.println("Hello, " + name);
        out.println("</BODY>");
        out.println("</HTML>");
        out.close();

    }
}
```

## Basic HTTP Servlet structure

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class MyServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        // Use "request" to read incoming HTTP headers and HTML form data
        // (e.g. data the user entered and submitted)
        ...
        // Perform any internal processing for generating dynamic results
        ...
        // Use "response" to specify the HTTP response line and headers
        // (e.g. specifying the content type).
        res.setContentType("text/html");
        // Use "out" to send content to browser
        PrintWriter out = response.getWriter();
        ...
    }
    ...
}
```

## Basic HTTP Servlet structure (2)

```
...
    public void doPost(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        doGet(req, res);
    }
}
```

## Outline

1. Introduction
2. HTTP basics
3. Servlet basics
4. **Miscellaneous**
  - User authentication
  - User session based on username
  - User session based on cookies

## User authentication

- Objective
  - Restrict access to some of resources of the web application
- Example
  - A magazine is published online
  - Only paid subscribers can read the articles
- Principles
  - An HTTP server has a built-in capability to restrict access to some or all of its resources to a given set of registered users.
  - How to set up restricted access depends on the server, but here are the underlying principles
  - The first time a web client (e.g. Browser) attempts to access one of these resources, the HTTP server replies that it needs special user authentication

## User authentication (2)

- Principles (cont.)
  - When the browser receives this response, it usually pops open a window asking the user for a name and password appropriate for the resource



- Once the user enters his information, the browser again attempts to access the resource, this time attaching the user's name and password along with the request
- If the server accepts the name/password pair, it happily handles the request.
- If, on the other hand, the server doesn't accept the name/password pair, the browser is denied

## Servlets and user authentication

- When access to a servlet has been restricted by the server, the servlet can get the name of the user that was accepted by the server
- To do so, the servlet uses the *getRemoteUser()* method
- This information is retrieved from the servlet's *HttpServletRequest* object
  - `public String HttpServletRequest.getRemoteUser()`
- This method returns the name of the user making the request as a String, or null if access to the servlet was not restricted

## Outline

1. Introduction
2. HTTP basics
3. Servlet basics
4. **Miscellaneous**
  - User authentication
  - **User session based on username**
  - User session based on cookies

## User session based on username

- Username can be used to track a client session
- Once a user has logged in, the browser remembers her username
- A servlet can identify the user through her username and thereby track her session
- Example
  - if the user adds an item to her virtual shopping cart, that fact can be remembered (e.g. in a shared class or external database)
  - This can be used later by another servlet when the user goes to the check-out page

## User session based on username (2)



- Example:
  - A servlet utilizes user authorization to add items to a user's shopping cart

```
String name = req.getRemoteUser();
if (name == null) {
    // Explain that the server administrator should
    // protect this resource
} else {
    String[] items = req.getParameterValues("item");
    if (items != null) {
        for (int i = 0; i < items.length; i++) {
            addItemToCart(name, items[i]);
        }
    }
}
```

## User session based on username (3)



- Example:
  - Another servlet can then retrieve the items from a user's cart

```
String name = req.getRemoteUser();
if (name == null) {
    // Explain that the server administrator should protect
    // this page
} else {
    String[] items = getItemsFromCart(name);
    ...
}
```

## Outline



1. Introduction
2. HTTP basics
3. Servlet basics
4. **Miscellaneous**
  - User authentication
  - User session based on username
  - **User session based on cookies**

## User session based on cookies



- Servlet API provides the *javax.servlet.http.Cookie* class for working with cookies
- A cookie is created with the *Cookie()* constructor
  - `public Cookie(String name, String value)`
- A servlet can send a cookie to the client by passing a *Cookie* object to the *addCookie()* method of *HttpServletResponse*
  - `public void HttpServletResponse.addCookie(Cookie cookie)`
- Because cookies are sent using HTTP headers, they should be added to the response before you send any content.
- Browsers are only required to accept
  - 20 cookies per site,
  - 300 total per user, and
  - they can limit each cookie's size to 4096 bytes.

## User session based on cookies (2)



- A servlet sets a cookie like this:  

```
Cookie cookie = new Cookie("ID", "123");  
res.addCookie(cookie);
```
- A servlet retrieves cookies by calling the `getCookies()` method of `HttpServletRequest`:  

```
public Cookie[] HttpServletRequest.getCookies()
```
- A servlet fetches cookies looks like this:  

```
Cookie[] cookies = req.getCookies();  
if (cookies != null) {  
    for (int i = 0; i < cookies.length; i++) {  
        String name = cookies[i].getName();  
        String value = cookies[i].getValue();  
    }  
}
```

## References



This lecture is extensively based on:

- S. Bodoff. *Java Servlet Technology*.  
[http://java.sun.com/j2ee/tutorial/1\\_3-fcs/doc/Servlets.html](http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Servlets.html)
- M. Boger. *Java in Distributed Systems: Concurrency, Distribution and Persistence*. Wiley, 2001.
- M. Hall. *Servlets and Java ServerPages: A Tutorial*.  
<http://www.apl.jhu.edu/~hall/java/Servlet-Tutorial/>
- J. Hunter, W. Crawford. *Java Servlet Programming*. O'Reilly, 1998.
- S. Zeiger. *Servlet Essentials*.  
<http://www.novocode.com/doc/servlet-essentials/>

