IBD – Intergiciels et Bases de Données

Servlet-based distributed systems

Fabien Gaud, fabien.gaud@inria.fr

http://www-ufrima.imag.fr/ ⇨ Placard électronique ⇨ M1 Info ⇨ IBD

Overview of lectures and practical work

- Lectures
  - Introduction to distributed systems and middleware
  - RMI-based distributed systems
  - **Servlet-based distributed systems**
  - Introduction to multi-tier distributed Internet services
- Practical work
  - Programming distributed systems with RMI
  - **Project on multi-tier Internet services**

Introduction – Web applications

- Communication between client and server
  - In a web application, client and server communicate via the HTTP protocol (HyperText Transfer Protocol)
- Web requests
  - 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

What are Servlets

- Servlets are Java programs which run in a server
  - Need a JVM and a servlet container
- 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
  - The data changes frequently
  - The Web page uses information from corporate databases or other such sources

Advantages of Servlets

- Efficiency
  - One process, the JVM
    - One thread per request (with traditional CGI, one process per request)
    - Can use pool of threads
    - Memory efficiency since servlet code is only loaded one time
- 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, etc.)
- Features
  - User session tracking
  - Database connection pools
  - etc.

Outline

- Introduction
- **HTTP basics**
- Servlet basics
- 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)
  - Guidelines and rules governing interactions between two parties
  - Examples:
    - 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
- Possibilities to maintain a client session
- Request
  - Client request has the following form:
    - a method,
    - target resource address (a URL),
    - HTTP protocol version

HTTP request headers

- When sending the request, the client can send optional header information
  - What software the client is running
  - What content types the client understands
- The request ends with an empty line
- This information does not directly pertains to what was requested, but it could be used by the server to generate its response

HTTP Request Example

```
   GET / HTTP/1.1\r\n
   Host: www.google.fr\r\n
   User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.1.8) Gecko/20100214 Ubuntu/9.10 (karmic)
Firefox/3.5.8\r\n
   Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
   Accept-Language: fr-fr,fr;q=0.8,en-us;q=0.5,en;q=0.3\r\n
   Accept-Encoding: gzip,deflate\r\n
   Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\n
   Keep-Alive: 300\r\n
   Connection: keep-alive\r\n
   [truncated] Cookie:
PREF=ID=e7aabd20dfdb8322:U=56759c536b80012d:FF=4:LD=fr:NR=10:TM=1263406128:LM=12650501
00:S=GGkjwPGnyIWu-wNk;
NID=32=f1DjnOUkft7DZSlODhJeaH84tgcB1lpSwv6Ogo3k5U23DXCYTuHvUyhVkF2HbA0i3vtKwUklyGhd-
BdTm-7ORYbHPZEWEdxOo4dnlgt
   \r\n
```

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

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
  - MIME type
  - Last modification
  - ...
- The server sends a blank line after the header
- If the request was successful, the requested data is sent as part of the response

HTTP Response example

```
HTTP/1.1 200 OK\r\n
Date: Sun, 28 Feb 2010 11:08:41 GMT\r\n
Expires: -1\r\n
Cache-Control: private, max-age=0\r\n
Content-Type: text/html; charset=UTF-8\r\n
Content-Encoding: gzip\r\n
Server: gws\r\n
Content-Length: 4721\r\n
\r\n
[ … ]
```

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://www.google.com/search?hl=fr&q=java+servlet& ....
```

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
- 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 its supports

Example of HTTP protocol

[TCP CONNECTION]

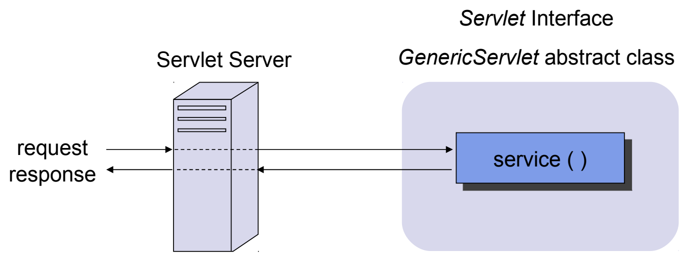
Client	→	Server	GET / HTTP/1.1
Server	→	Client	HTTP/1.1 200 OK (text/html) […]
Client	→	Server	GET /intl/fr_fr/images/logo.gif HTTP/1.1
Server	→	Client	HTTP/1.1 200 OK (GIF89a) […]
Client	→	Server	GET /extern_js/xxx.js HTTP/1.1
Server	→	Client	HTTP/1.1 200 OK (text/javascript) […]

[TCP FIN]

Outline

- Introduction
- HTTP basics
- Servlet basics
  - Generic servlets and HTTP servlets
  - Servlet lifcycle
  - Servlet API
  - A simple example
  - Getting information from requests
  - An HTML form example
- Miscellaneous

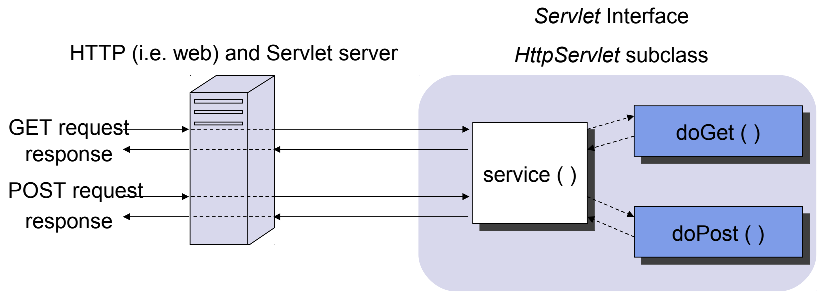
A generic servlet handling a request



"service" method is the GenericServlet's entry point

▨ Implemented by subclass

An HTTP servlet handling GET and POST requests

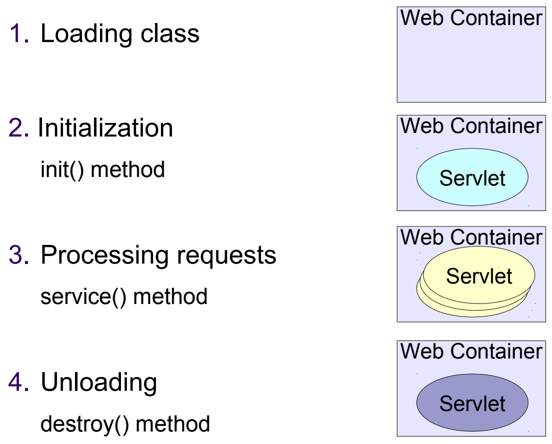


"doGet" method is the HttpServlet's entry point for GET requests

"doPost" method is the HttpServlet's entry point for POST requests

▨ Implemented by subclass

Servlet lifecycle



©D. Donsez

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
  - creating an instance of the Servlet by calling the no-args constructor
  - calling the Servlet's init(ServletConfig config) method
- Servlet's init(ServletConfig config) method
  - 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
  - Is guaranteed to be called only once during the Servlet's lifecycle

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
  - This method is guaranteed to be called only once during the Servlet's lifecycle

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:
      - 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, bind an object to a given attribute, ...)

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

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> <P> Hello World </P> </BODY>");
        out.println("</HTML>");
        out.close();

   }

}
```

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
  - 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
  - http://java.sun.com/webservices/docs/1.5/api/javax/servlet/http/HttpServletRequest.html

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
                ...
        }
        ...
   }
   ...
}
```

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).
        PrintWriter out = res.getWriter();
        // Use "out" to send content to browser
        ...
   }
...
```

Basic HTTP Servlet structure (2)

```
...

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

        doGet(req, res);

   }

}
```

Outline

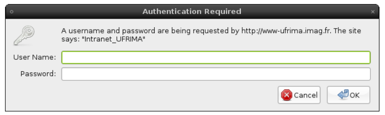
- Introduction
- HTTP basics
- Servlet basics
- Miscellaneous
  - **User authentication**
    - User session based on username
    - User session based on cookies
    - User session based on HttpSession
  - Notes about deployment

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 asks the user for a name and password


Authentication Required

A username and password are being requested by http://www-ufrima.imag.fr. The site says: "Intranet_UFRIMA"

User Name:

Password:

  - 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
  - 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 th user login is not known
  - At this time, the user authentication has already been done by the server

User session based on username

- Username can be used to track a client session
- Once a user has logged in, the browser remembers his username
- A servlet can identify the user through his 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 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)
    - Value can be changed later
- 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.
- Number and size of cookie are restricted

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();
        }
}
```

User session based on HttpSession

- The easiest way to maintain data associated with a client
- Usually maintained using cookies and associated with a timeout
- Get the current HttpSession

HttpSession session = request.getSession();

- Set attributes to a session

session.setAttribute(“name”,(MyObject) value);

- Get attributes from a session

MyObject value = (MyObject)session.getAttribute("name");

- Invalidate a session

session.invalidate();

Outline

- Introduction
- HTTP basics
- Servlet basics
- Miscellaneous
  - User authentication
    - User session based on username
    - User session based on cookies
    - User session based on HttpSession
  - **Notes about deployment**

Tomcat

- Widely used servlet container developed by the Apache Foundation
- Important directories
  - /bin : startup and shutdown scripts
  - /conf : configuration files especially server.xml and tomcat-users.xml
  - /lib : contains needed libraries (for example jdbc drivers)
  - /logs : server log files
  - /webapps : place here your web apps
- Important environment variables
  - $CATALINA_HOME must be set to the root of Tomcat installation

Standard directory layout

- See http://tomcat.apache.org/tomcat-6.0-doc/appdev/deployment.html
- Place at the root all needed resources (images, html pages, …)
- /WEB-INF/lib/ : libraries needed by your web application
- /WEB-INF/classes/ : Contains java classes (both servlet and non-servlet)
- /WEB-INF/web.xml : allow to specify servlet ↔ url mapping as well as initialization parameters

Packaging

- Goal: distributing of web applications
- Package must contains all application needs
  - Libraries, resources, ...
- All contained in a .war
  - A jar with the organization described before
- Example: Use with Tomcat
  - Put war in webapps/
  - Start (or restart) the server

Incoming lectures and practical work on middleware

- Lectures
  - Introduction to distributed systems and middleware
  - RMI-based distributed systems
  - Servlet-based distributed systems
  - **Introduction to multi-tier distributed Internet services**
- Practical work
  - Programming distributed systems with RMI
  - **Project on multi-tier Internet services**

References

This lecture is extensively based on:

- S. Bodoff. *Java Servlet Technology*. 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/