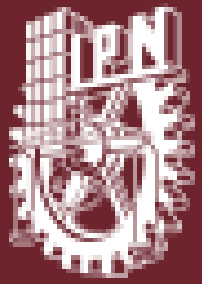


Instituto Politécnico Nacional

Escuela Superior de Cómputo

THEMATIC UNIT: II:
Java Servlets

M. en C. José Asunción Enríquez Zárate

asuncionez@gmail.com

Java Servlets

UNIT OF COMPETENCE

The student builds Web applications based on Java Servlet specification

Contents

Contents

Web Application Technologies

Web Application Technologies

- HTML over HTTP
- HTTP Client Server Architecture

HTTP Client Server Architecture

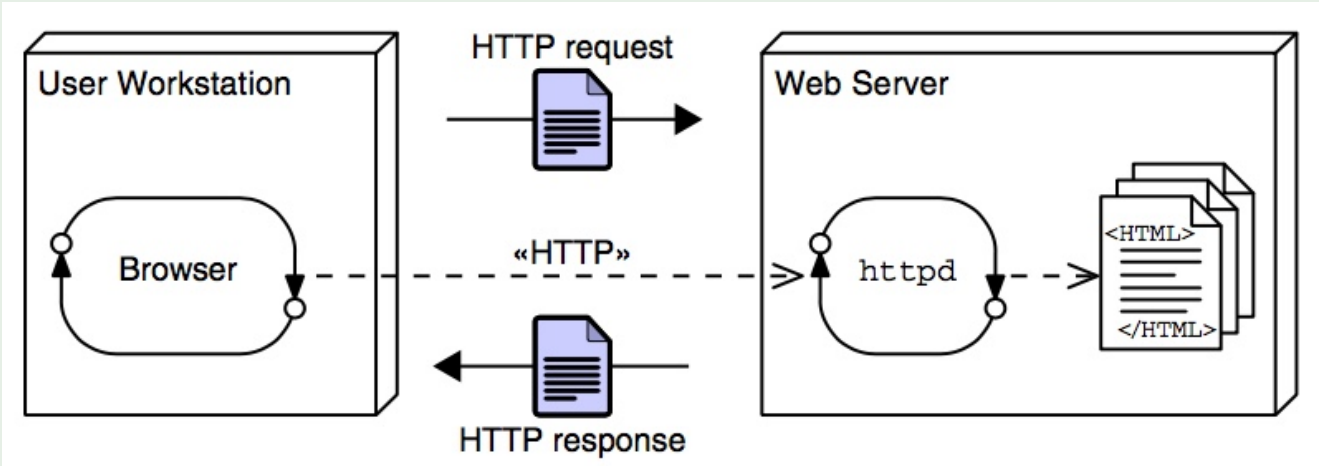
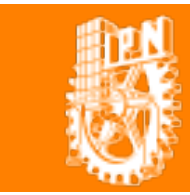


Figure: HTTP Client Server Architecture



Web Application Technologies

Web Site Structure

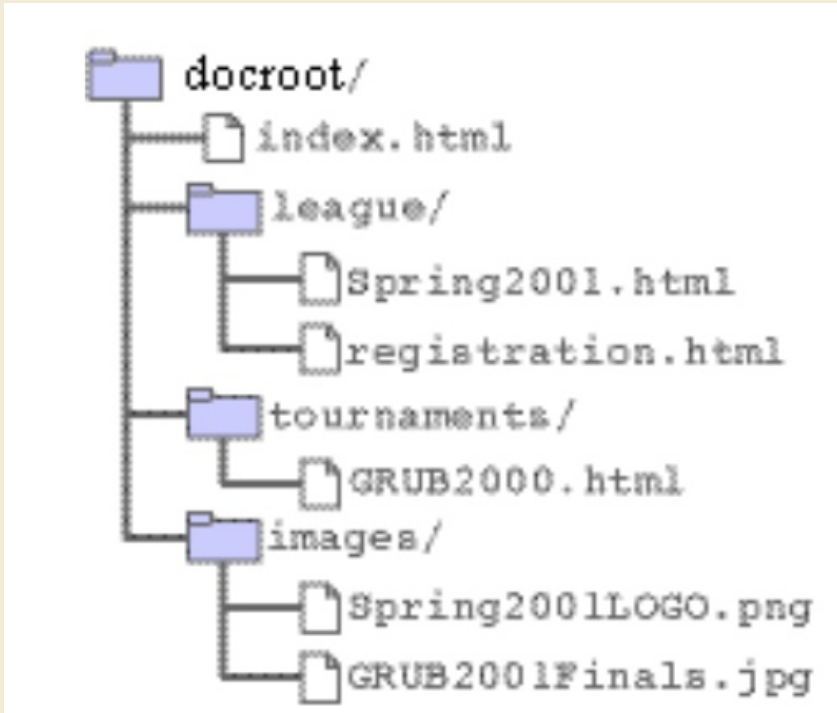


Figure: Web Site Structure

Web Application Technologies

Uniform Resource Locator

- protocol://host:port/path/file
  - *http://148.204.56.120:8080/index.html*

Figure: Uniform Resource Locator

Web Sites and Web Applications

- Browser requests HTML form
- Server responds
- User provides data
- Browser sends second request
- Server processes data using specific computation
- Server sends results to browser
- Web applications are for users, web services for computers

Contents

Introduction to Servlets

Understanding HTTP

HTTP defines the way in which web browsers interact with web servers. HTTP uses TCP/IP, the network protocol of the Internet, to communicate standard messages between machines across the Internet. By using standard protocols such as these, you're able to communicate with any web server from a variety of different web browsers and expect similar behavior.

Introduction to Servlets

Understanding HTTP

At the heart of HTTP lies a request message and a response message. This is the fundamental way in which a web browser communicates with a web server (see Figure). The user types in the location of a document in the URL box, the browser issues a standard HTTP request for the document, and the document is located and returned to the browser as a standard HTTP response.

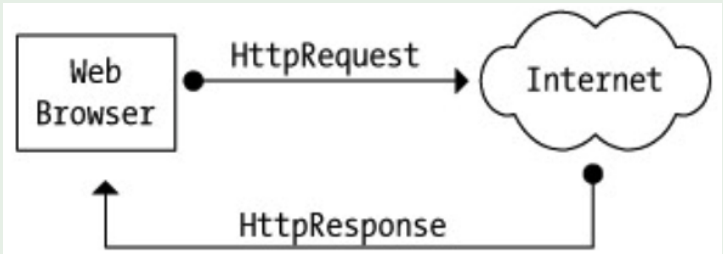


Figure: Basic HTTP exchange.

Introduction to Servlets

Understanding HTTP

The HTTP request consists of a series of standard headers along with any parameters, or form data, necessary to fulfill the request. The web server, for which the request is intended, is able to read these headers and respond accordingly.
There are two common types of HTTP requests, GET and POST.

* GET
* POST

A GET request will append form data to the requested URL and send it as one packet. A POST request will first send a packet containing header information and then send a separate packet containing the form data.

Introduction to Servlets

Understanding HTTP

A common question is, "Which type of request should I use in certain situations?

A good rule of thumb is that you should use:

* POST requests to modify a resource on the server
* GET requests to simply retrieve information from the server

You may find that this doesn't always apply in every situation.

Contents

Introduction to Servlets

Components of a Web Application

A typical web application involves a web server, an application server, and a database server.

Each of these servers listens to a specific TCP/IP port for incoming messages containing requests to carry out.

These listeners are sometimes called daemons. They're threads of execution that wait for TCP/IP messages to appear for a specific port. For instance, the web server will, by default, listen to requests addressed to port 80.

Introduction to Servlets

Components of a Web Application

Because web servers default to this port, there's no need to specify it in the URL, it's just implied.
If you were to request http://www.apress.com, the request would be sent to port 80 of the machine on which the web server is running.
To specify a different port, let's say port 8080, you would add it to the URL like this: http://www.asuncionez.com.mx:8081.

Contents

Introduction to Servlets

Java in the Web

- Introduced with Servlets
- Similar execution model to CGI
- Single process, multiple threads

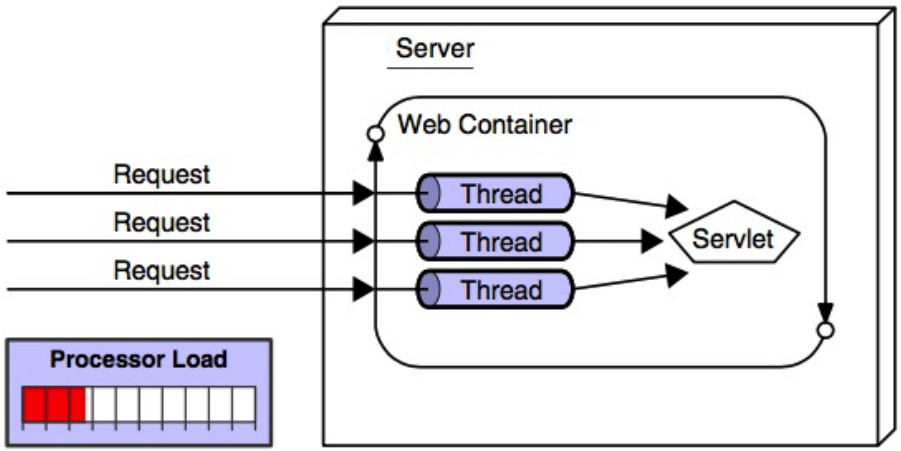


Figure: Java in the Web.

Introduction to Servlets

Advantages and Disadvantages of Java Servlets

- Single process, so faster than CGI
- More scalable than CGI
- Use Java programming language
- Standardized logging
- Error handling and security
- Must use the Java programming language
- Concurrency not limted to the database

Introduction to Servlets

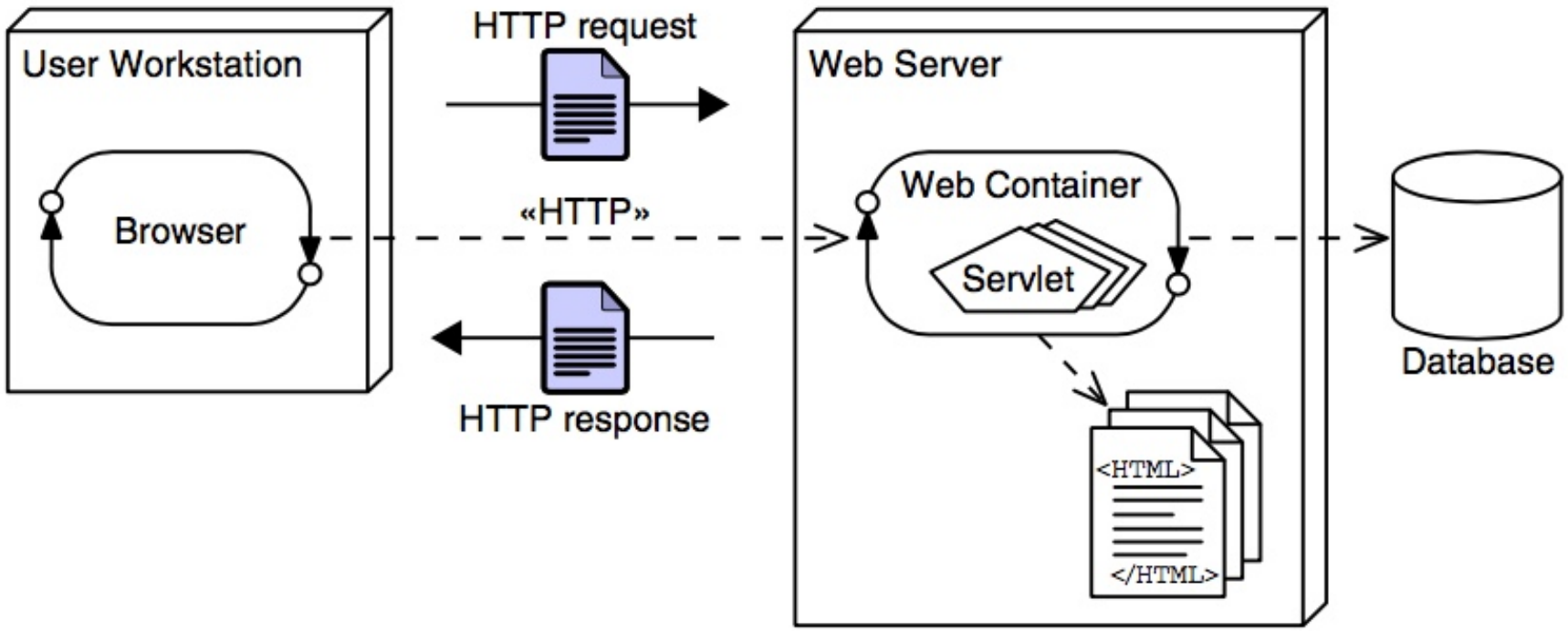


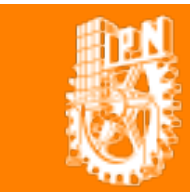
Figure: Java Servlets Architecture.

Introduction to Servlets

```java
package com.darkdestiny.mx;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
/**** @author darkdestiny */
@WebServlet(name = "TablasDeMultiplicar", urlPatterns = {"/TablasDeMultiplicar"})
public class TablasDeMultiplicar extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet TablasDeMultiplicar</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Tablas De Multiplicar </h1>");
            out.println("<table border=1>");
            for (int i = 1; i <= 10; i++) {
                out.println("<tr>");
                for (int j = 1; j <= 10; j++) {
                    out.println("<td>    " + (i * j) + "  </td>");
                }
                out.println("</tr>");
            }
            out.println("</table>");
            out.println("</body>");
            out.println("</html>");
        }
    }
}
```

Contents

Contents



Life Cycle and the Servlet API

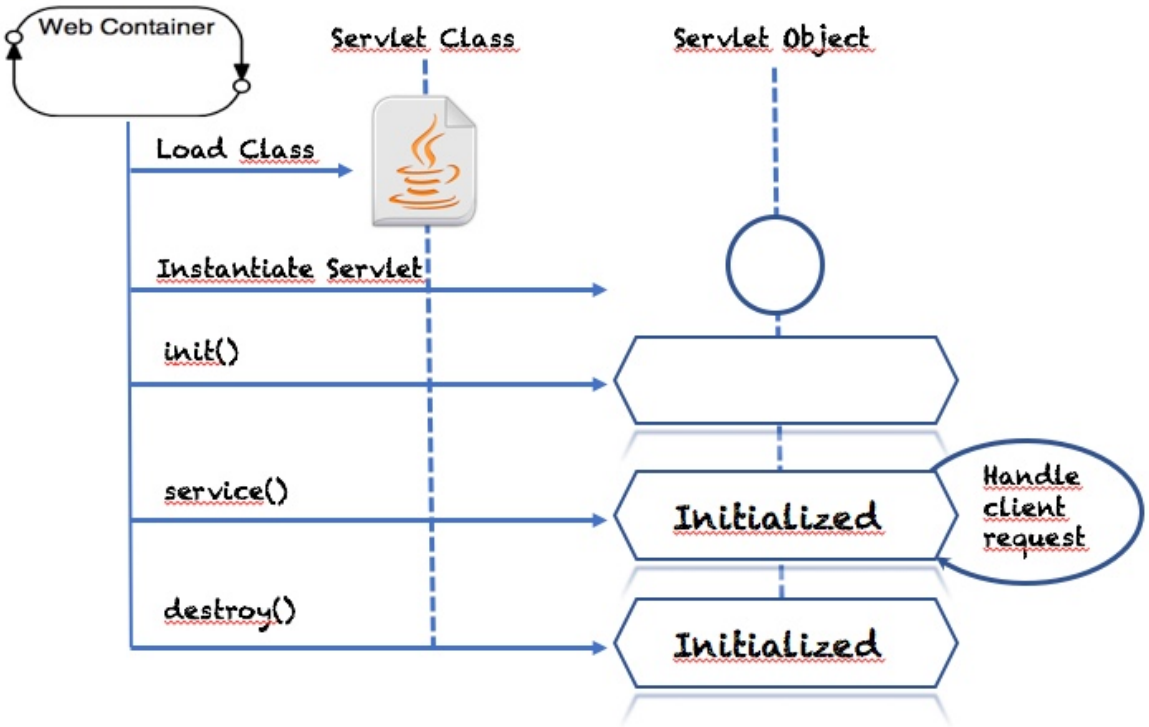
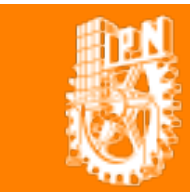


Figure: Servlet life cycle.



Life Cycle and the Servlet API

Figure: Instantiate Servlet.

Instantiate Servlet

- Run the servlet class no-arg constructor.
- Don't need to write a constructor.
- Just use the compiler-supplied default.

Life Cycle and the Servlet API

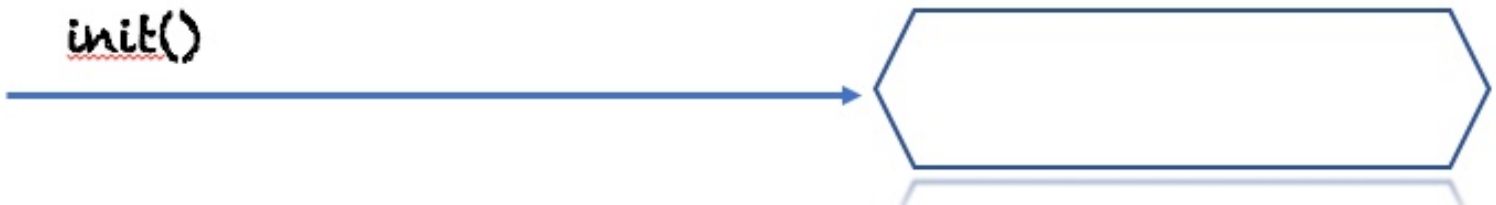
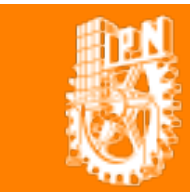


Figure: Servlet init() method.

Init method (*init()*)

- Called only ONCE in the servlet's life.
- Must complete before Container can call service() method.



Life Cycle and the Servlet API

Service Method (*service()*)

- doGet.
- doPost.
- other http methods.
- Each request runs in a separated thread.

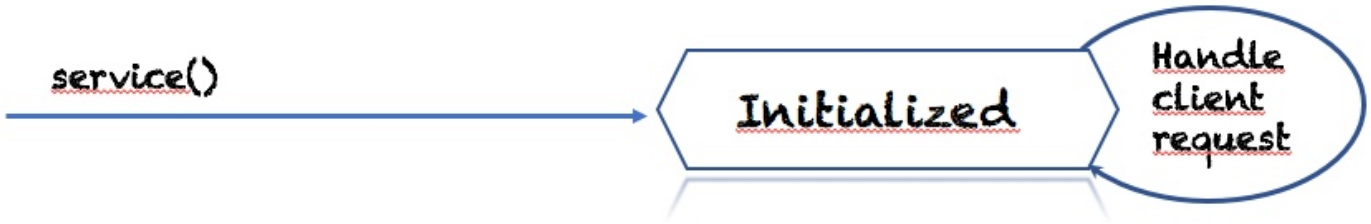
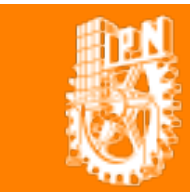


Figure: Service method.



Life Cycle and the Servlet API

Destroy Method (*destroy()*)

- Container calls to give the servlet a chance to clean up before the servlet is killed.
  - Ready for garbage collection.
- Like *init()* method, it's called only once.

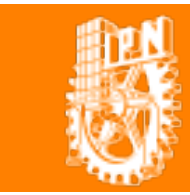
Figure: Destroy method.

Life Cycle and the Servlet API

Tabla de Multiplicar sin Encabezados									
1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

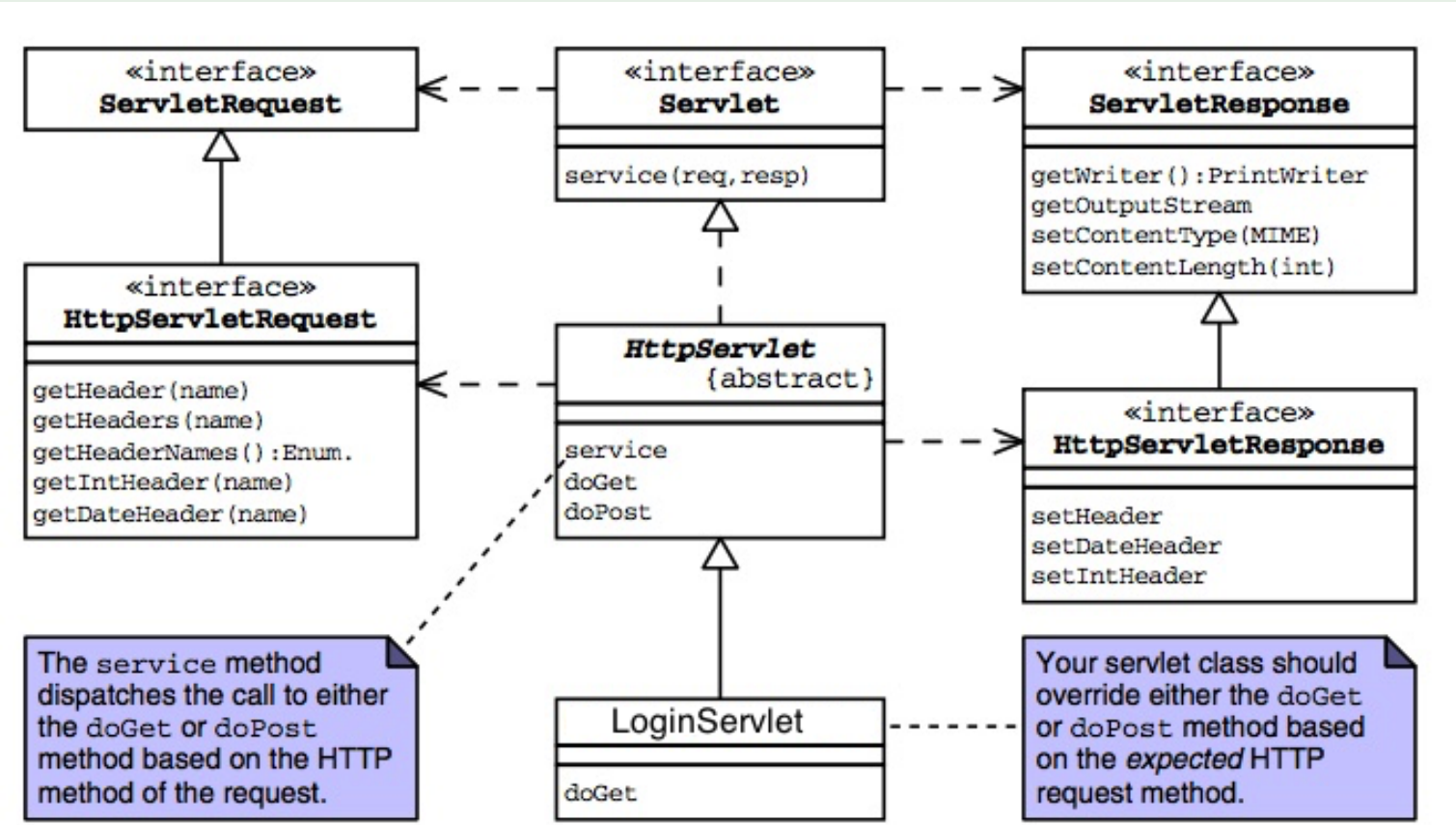
```
--------------------------------------------------
init method has been called and servlet is initialized
--------------------------------------------------
--------------------------------------------------
 service method has been called
--------------------------------------------------
--------------------------------------------------
destroy method has been called and servlet is destroyed
--------------------------------------------------
```

Contents



Context of HttpServlets

Context of HttpServlets



Contents

Sessions

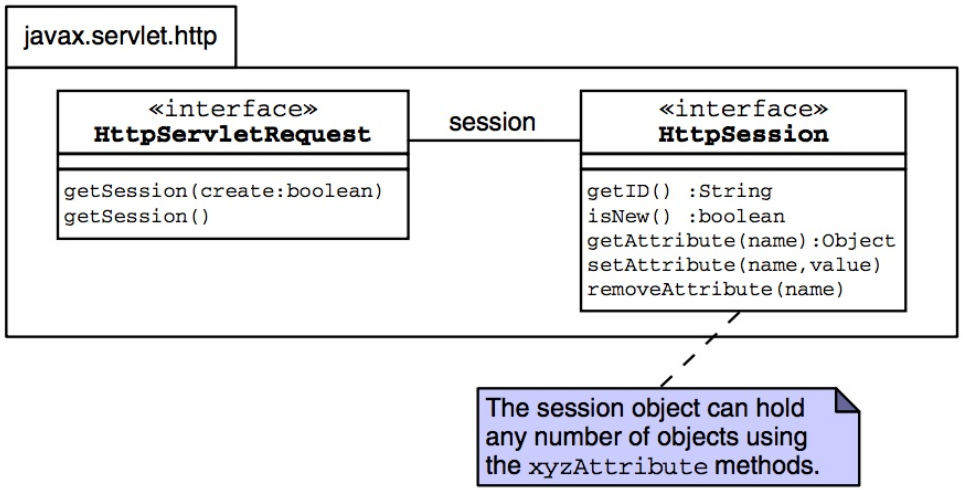
Sessions

- The servlet API provides a convenient wrapper
  - A hashtable-like interface named javax.servlet.http.HttpSession
- has two important methods
  - setAttribute()
  - getAttribute()
- Store and retrieve objects by name.
- Provides a session ID key that a participating client stores and returns on subsequent requests in the same session. The servlet engine looks up the appropriates session object and makes it available to the current request.

Sessions

Sessions

- HTTP is stateless, good for clustering.
- Sessions are not directly supported.
- Browser must return identification data with every request.
- HttpSession class addresses needs.



Sessions

Sessions Configuration

- web.xml can specify default session timeout and preferred session tracking.

```
<web-app ...>
    <session-config>
        <session-timeout>30</session-timeout>
        <tracking-mode>SSL</tracking-mode>
    </session-config>
</web-app>
```

Sessions

Method	Description
Object getAttribute (String name)	Stores an object in the session under the specified name, or returns or removes an object by that name that was previously stored.
void setAttribute (String name, Object value) void removeAttribute (String name)	
Enumeration getAttributeNames()	Returns an Enumeration of the names of all attributes currently bound to the session

Table: Methods in the HttpSession Interface.

Sessions

Method	Description
long getCreationTime()	Returns a long integer representing the date and time at which the session was created or last accessed. The integer is in the form used by the java.util.Date() constructor.
long getLastAccessedTime()	
String getId()	Returns the session ID, a unique key assigned by the servlet engine.
void invalidate()	Causes the session to expire and unbinds any objects in it.

Table: Methods in the HttpSession Interface.

Sessions

Method	Description
lint getMaxInactiveInterval()	Sets or returns the maximum number of seconds the session will be kept alive if no interaction occurs with the client.
void setMaxInactiveInterval (int seconds)	
boolean isNew()	Returns true if the client hasn't yet joined the session. This is true when the session is first created and the session ID is passed to the client, but the client hasn't made a second request that includes the session ID..

Table: Methods in the HttpSession Interface.

Contents

Cookies

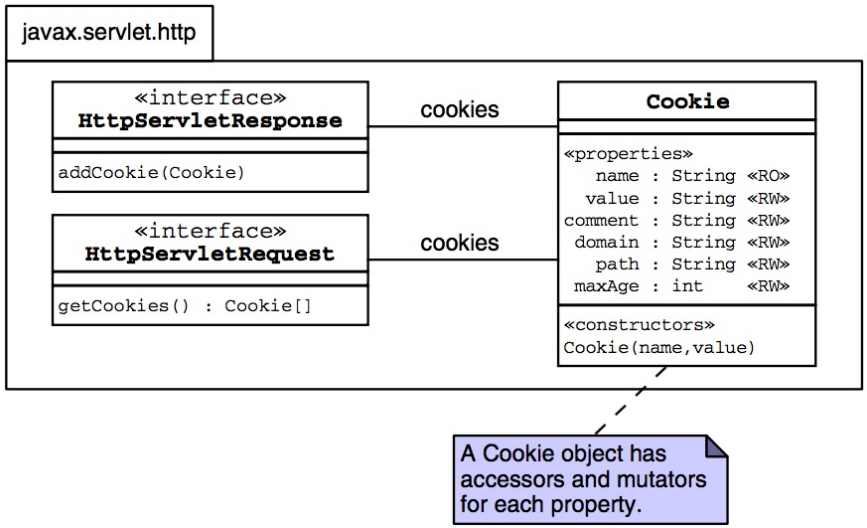
Cookies

- Cookies are key-value data pairs stored on the browser.
- Cookies are created and updated in a server response to the browser.
- Cookies are stored by the browser in the client system.
- Cookies can be partitioned by server and path.
- All relevant cookies are sent by browser to server with every request.
- Cookies can have life-span limits

Cookies

Cookies

- Create cookies with new Cookie(name, value).
- Use response.addCookie to set the stored value.
- Use request.getCookies to read cookies.



Cookies

Using Cookies

- To store a username from a form for future use.

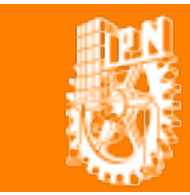
```
String name = request.getParameter("firstName");
Cookie c = new Cookie("yourname", name);
response.addCookie(c);
```

Using Cookies

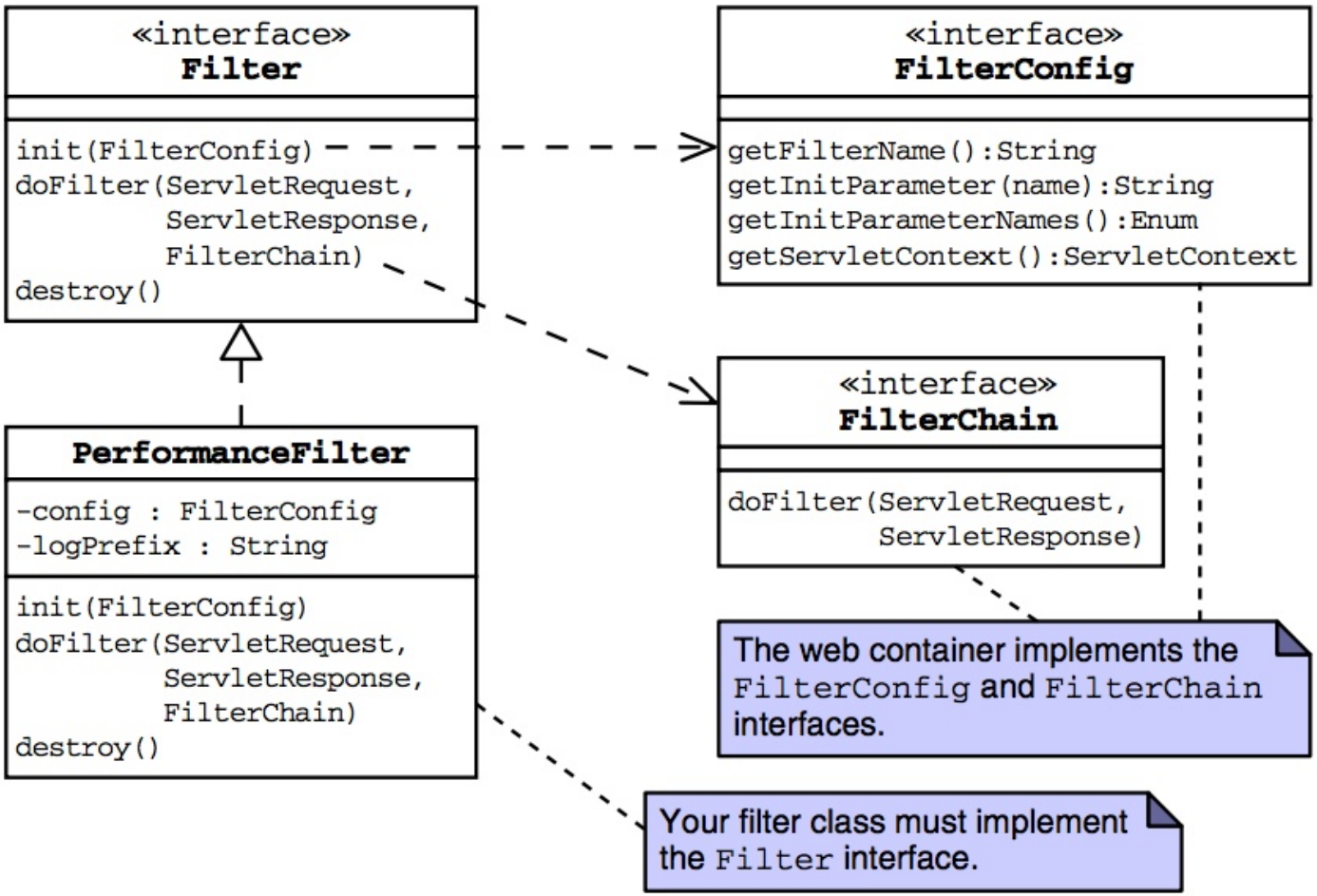
- To retrieve username on subsequent request.

```
Cookie[] allCookies = request.getCookies();
for ( int i=0; i < allCookies.length; i++ ) {
  if ( allCookies[i].getName().equals(“yourname”) ) {
    name = allCookies[i].getValue();
  }
}
```

Contents



Filters

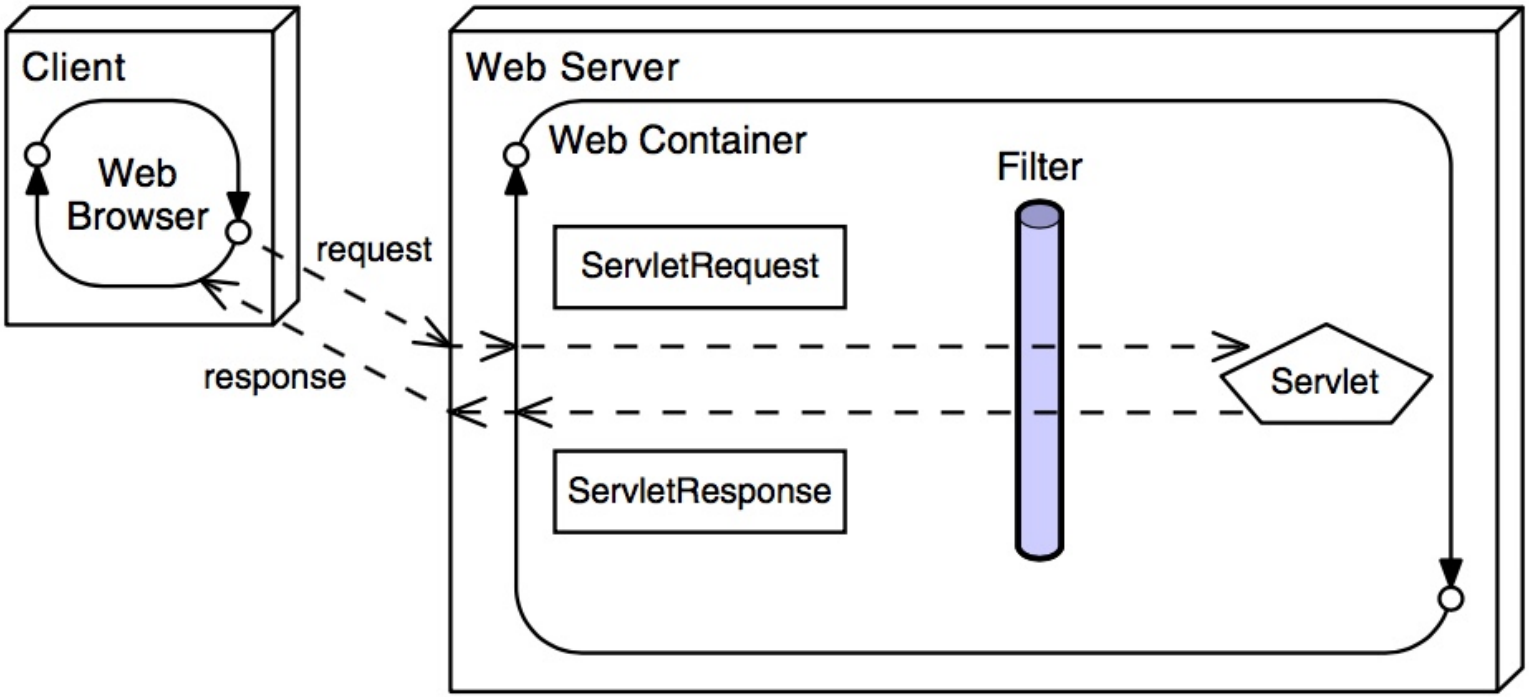

«interface»
Filter
init(FilterConfig)
doFilter(ServletRequest,
ServletResponse,
FilterChain)
destroy()
«interface»
FilterConfig
getFilterName():String
getInitParameter(name):String
getInitParameterNames():Enum
getServletContext():ServletContext
«interface»
FilterChain
doFilter(ServletRequest,
ServletResponse)
PerformanceFilter
-config : FilterConfig
-logPrefix : String
init(FilterConfig)
doFilter(ServletRequest,
ServletResponse,
FilterChain)
destroy()
The web container implements the FilterConfig and FilterChain interfaces.
Your filter class must implement the Filter interface.

Filters

Applying Filters

- Container private pre-processing.
- Container checks for filter requirements for this URL
  - Locate filter and execute.
  - Repeat for other defined filters.
- If no errors pass request to target servlet.

Filters


Client
Web Browser
request
response
Web Server
Web Container
Filter
ServletRequest
ServletResponse
Servlet

Filters

Filter Applicability

- Blocking access to a resource based on user identity or role membership.
- Auditing incoming requests.
- Compressing the response data stream.
- Transforming the response.
- Measuring and logging servlet performance.

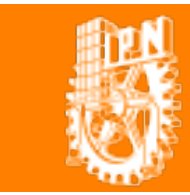
Contents

Connection pool

Connection pool

- java.sql.DriverManager.getConnection
  - obtains an exclusive connection.
  - Poor scaling behavior.
- javax.sql.DataSource
  - Simplifiespoolingandis portable across application servers and deployments.
  - is obtained from a JNDI lookup.

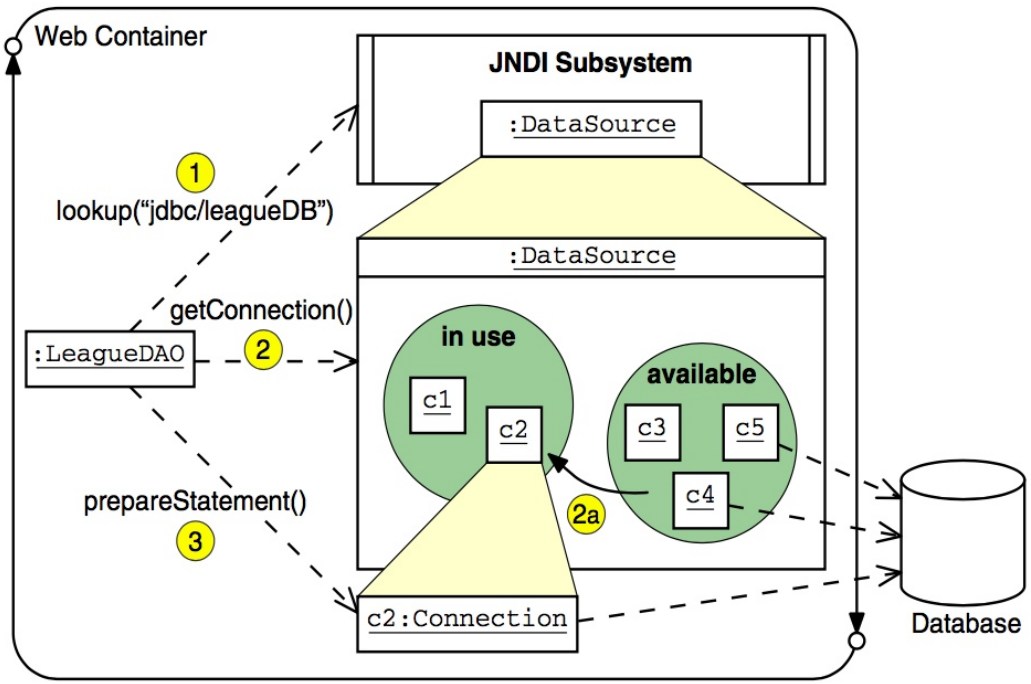
```
Context ctx = new InitialContext();
ds = (DataSource)ctx.lookup("java:comp/env/
jdbc/leagueDB");
```

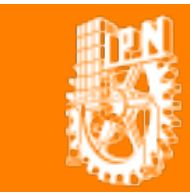


Connection pool

Obtaining a Pooled Connection

- DataSource.getConnection usually returns a connection from a pool.

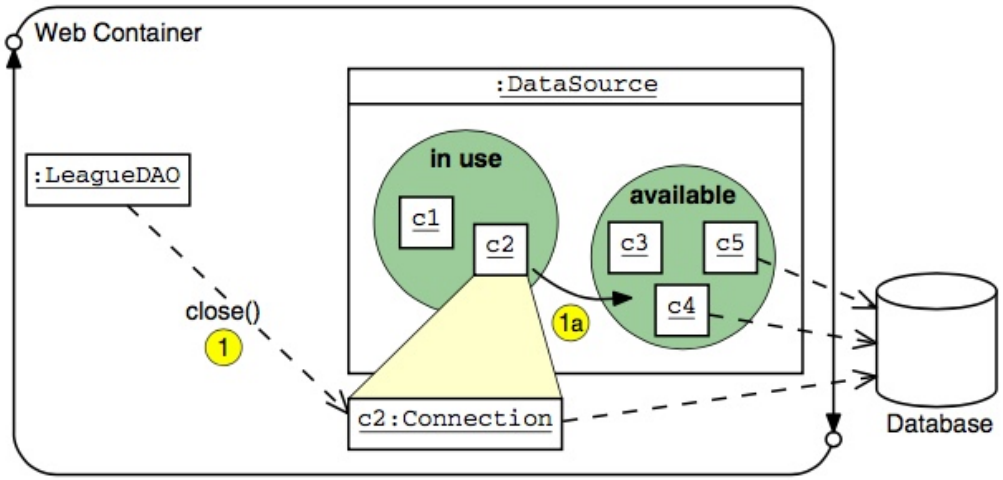




Connection pool

Obtaining a Pooled Connection

- Closing the connection returns it to the pool.



Connection pool

Configuring a DataSource

- The database and tables must exist.
- A connection pool must be prepared for that database.
- A JNDI entry must refer to the pool.

Contents

Referencias

Oracle.
Web Component Development With Servlet and JSP Technologies
Oracle.

Edgar Martinez, Tom McGinn, Eduardo Moranchel, Anjana Shenoy, Michael Williams.
Java EE 7: Back-end Server Application Development
Oracle, 2016.

Eric Jendrock, Ricardo Cervera-Navarro, Ian Evans, Kim Haase, William Markito.
Java Platform, Enterprise Edition (Java EE) 8 The Java EE Tutorial
Oracle, 2017.