Java Enterprise Edition

Java Servlets

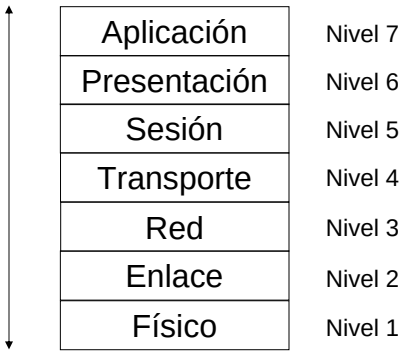
Copyright

Java Servlets

- Los Java Servlets son un tipo de componente definido por la plataforma Java EE.
- Se ubican en la capa web (contenedor web).
- Surgieron para implementar contenido dinámico en la web.
- Aunque la especificación Java Servlet no está atada a un protocolo concreto, su uso mas generalizado y la implementación por defecto usa HTTP.
- El navegador web es el tipo de cliente mas frecuente.

Protocolo HTTP

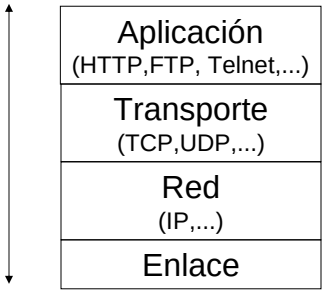
- Un sistema de comunicaciones se compone de una pila de niveles encargados de distintas tareas.



- A este modelo teórico se le llama el Modelo de Referencia OSI.

Protocolo HTTP (cont.)

- En una red TCP/IP (por ejemplo Internet) algunos niveles se fusionan quedando la siguiente pila:



- HTTP: Hipertext Transfer Protocol.
- Estándar definido por el IETF (RFC 2616):
  - http://www.ietf.org/rfc/rfc2616.txt

Protocolo HTTP (cont.)

- Una conversación HTTP es una secuencia de:
  - Una petición (HTTP Request).
  - Una respuesta (HTTP Response).
- Una petición se compone de:
  - Método HTTP: la acción a realizar.
  - Recurso a acceder: una URL.
  - Parámetros: como los argumentos de un método Java.
- Una respuesta se compone de:
  - Código de estado: indica cómo fue la petición.
  - Tipo de contenido: formato de la respuesta (HTML, GIF,...).
  - Contenido: la respuesta como tal.

Protocolo HTTP (cont.)

- Un mensaje HTTP se compone de:
  - Cabecera: incluye información técnica.
  - Cuerpo: incluye el contenido del mensaje como tal.
- El protocolo HTTP define distintos métodos:
  - GET, POST, PUT, DELETE, TRACE, OPTIONS, HEAD.
- Los mas relevantes son GET y POST.
- Método GET:
  - Es el mas simple.
  - Se suele usar para pedir un recurso.
  - Puede enviar parámetros pero los añade como parte de la URL, lo que puede ser un problema.

Protocolo HTTP (cont.)

- Ejemplo de una petición GET:

```
GET /Practica23a/Sumador?param1=12&param2=41 HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-excel, application/vnd.ms-
powerpoint, application/msword, application/x-shockwave-flash, */*
Referer: http://127.0.0.1/Practica23a/index.html
Accept-Language: en-us,es;q=0.5
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322; .NET CLR 2.0.50727)
Host: 127.0.0.1:8080
Connection: Keep-Alive
```

  - Solo incluye la cabecera.
  - Los parámetros se añaden al final con el carácter ?.
  - Los parámetros se separan entre si con el carácter &.

Protocolo HTTP (cont.)

- Ejemplo de una respuesta GET:

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Set-Cookie: JSESSIONID=304CE17CCEB55869A606CE06271F915B; Path=/Practica23a
Content-Type: text/html;charset=ISO-8859-1
Content-Length: 314
Date: Sun, 08 Jun 2008 09:19:19 GMT


<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
      <title>Práctica 23a</title>
  </head>
  <body>
    El resultado de la suma de 12 y 41 es: 53
  </body>
</html>
```

- Incluye tanto la cabecera como el cuerpo.

Protocolo HTTP (cont.)

- Método POST:
- Es el mas usado para el envío de parámetros.
- Los parámetros no forman parte de la URL sino del cuerpo.
- Ejemplo de una petición POST:

```
POST /Practica23a/Sumador HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-excel, application/vnd.ms-
powerpoint, application/msword, application/x-shockwave-flash, */*
Referer: http://127.0.0.1/Practica23a/index.html
Accept-Language: en-us,es;q=0.5
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322; .NET CLR 2.0.50727)
Host: 127.0.0.1:8080
Content-Length: 19
Connection: Keep-Alive
Cache-Control: no-cache
Cookie: JSESSIONID=304CE17CCEB55869A606CE06271F915B


param1=45&param2=21
```

- Incluye tanto la cabecera como el cuerpo.

Protocolo HTTP (cont.)

- Ejemplo de una respuesta POST:

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/html;charset=ISO-8859-1
Content-Length: 314
Date: Sun, 08 Jun 2008 09:27:24 GMT


<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
      <title>Práctica 23a</title>
  </head>
  <body>
    El resultado de la suma de 45 y 21 es: 66
  </body>
</html>
```

- Incluye tanto la cabecera como el cuerpo.

Protocolo HTTP (cont.)

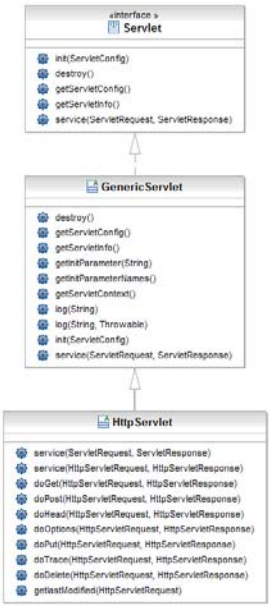
- Los parámetros de una petición HTTP pueden ser:
  - Caracteres ASCII alfanuméricos incluyendo los caracteres: . (punto), - (guión), * (asterisco) y _ (subrayado).
- El resto de caracteres han de ir codificados:
  - El espacio en blanco se sustituye por el carácter + (suma).
  - El resto se sustituye por su valor hexadecimal: %xx (por ejemplo, la ñ se sustituye por %F1)
- Si se usa el método GET, la codificación es responsabilidad del desarrollador.
- Si se usa el método POST, la codificación se realiza automáticamente.

Protocolo HTTP (cont.)

- Una URL es un identificador unívoco de un recurso en Internet.
- Una URL consta de las siguientes partes:
  - Protocolo: http, https, ftp, etc…
  - Servidor: dirección IP o nombre de máquina.
  - Puerto.
  - Path.
  - Recurso.
- Por ejemplo:

  http://127.0.0.1:8080/Practica23a/index.html

Jerarquía de clases



Interface Servlet
(javax.servlet.Servlet)

Clase GenericServlet
(javax.servlet.GenericServlet)

Clase HttpServlet
(javax.servlet.http.HttpServlet)

Ciclo de vida de un Servlet

- El contendor web gestiona el ciclo de vida de un Java Servlet:
  - **public void** init(ServletConfig);
  - **public void** service(HttpServletRequest, HttpServletResponse);
  - **public void** destroy();
- **public void** init(ServletConfig);
  - Es el primer método que se ejecuta una vez que se ha llamado al constructor del servlet.
  - Solo se ejecuta una vez en la vida del servlet.
  - Se suele usar para la inicialización del servlet.

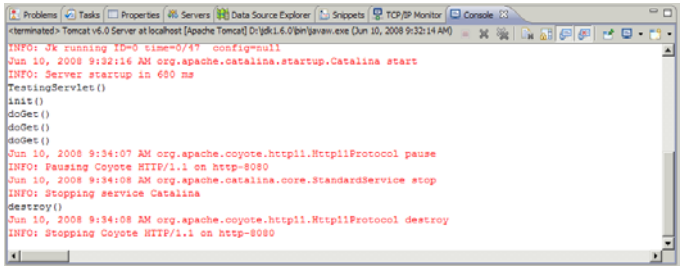
Ciclo de vida de un Servlet

- **public void** service(HttpServletRequest, HttpServletResponse);
  - Se ejecuta una vez por cada petición.
  - La implementación por defecto analiza el método HTTP y redirige la ejecución al método concreto: doGet, doPost,…
  - Normalmente son estos métodos los que se sobrescriben con nuestra lógica particular.
- **public void** destroy();
  - Es el último método del ciclo de vida. Se ejecuta normalmente cuando se para el Servidor de Aplicaciones.
  - Solo se ejecuta una vez en la vida del servlet.
  - Se suele usar para liberar cualquier recurso utilizado.

Ejemplo del Ciclo de Vida

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

public class TestingServlet extends javax.servlet.http.HttpServlet
{
  public TestingServlet()
  {
    System.out.println("TestingServlet()");
  }
  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
  {
    System.out.println("doGet()");
  }
  public void init() throws ServletException
  {
    System.out.println("init()");
  }
  public void destroy()
  {
    System.out.println("destroy()");
  }
}
```

URL http://localhost:8080/TestingWeb/TestingServlet

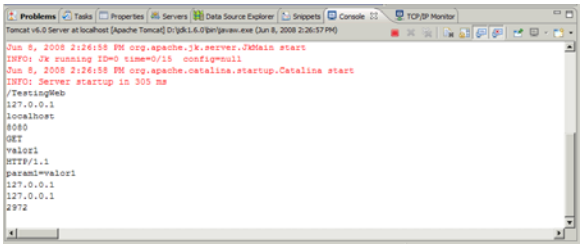


HttpServletRequest

- Es el interface que define una petición HTTP.
- Algunos de sus métodos son:
  - String getParameter(String name);
    Recupera un parámetro de entrada por nombre.
  - **void** setAttribute(String name, Object o);
    Introduce un atributo en la petición.
  - Object getAttribute(String name);
    Recupera un atributo de entrada por nombre.
  - String getHeader(String name);
    Recupera una cabecera HTTP de la petición por nombre.
- Otros métodos:
  - http://java.sun.com/javaee/5/docs/api/javax/servlet/http/HttpServletRequest.html#method_summary

Ejemplo

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class TestingServlet extends javax.servlet.http.HttpServlet
{
  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
  {
    System.out.println(request.getContextPath());
    System.out.println(request.getLocalAddr());
    System.out.println(request.getLocalName());
    System.out.println(request.getLocalPort());
    System.out.println(request.getMethod());
    System.out.println(request.getParameter("param1"));
    System.out.println(request.getProtocol());
    System.out.println(request.getQueryString());
    System.out.println(request.getRemoteAddr());
    System.out.println(request.getRemoteHost());
    System.out.println(request.getRemotePort());
  }
}
```



URL http://localhost:8080/TestingWeb/TestingServlet?param1=valor1

HttpServletResponse

- Es el interface que define una respuesta HTTP.
- Algunos de sus métodos son:
  - ServletOutputStream getOuputStream();
    Accede al stream de escritura enviar la respuesta.
  - PrintWriter getWriter();
    Accede al writer de escritura para enviar la respuesta.
  - **void** setContentType(String type);
    Ajusta el tipo de contenido de la respuesta.
  - **void** setStatus(**int** sc);
    Ajusta el código de la respuesta.
- Otros métodos:
  - http://java.sun.com/javaee/5/docs/api/javax/servlet/http/HttpServletResponse.html#method_summary

Content-Type

- Es una cabecera HTTP que especifica el tipo de contenido del mensaje.
- El cliente es el responsable de saber manejar los distintos tipos de contenidos.
- Los navegadores se suelen extender con plugins para interpretar nuevos tipos que van surgiendo.
- Algunos tipos:
  - text/plain, text/html, text/css, text/xml….
  - image/gif, image/jpg, text/tiff, image/png….
  - application/pdf, video/mpeg, video/quicktime….

Ejemplo

```
import java.io.FileInputStream;
import java.io.IOException;
import java.io.OutputStream;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class TestingServlet extends javax.servlet.http.HttpServlet
{
  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
  {
    response.setContentType("image/gif");

    File file = new File("c:\\duke.gif");
    response.setContentLength((int)file.length());

    FileInputStream in = new FileInputStream(file);
    OutputStream out = response.getOutputStream();

    byte[] buf = new byte[1024];
    int count = 0;
    while((count = in.read(buf)) >= 0)
      out.write(buf, 0, count);

    in.close();
  }
}
```

URL http://localhost:8080/TestingWeb/TestingServlet

HTML

- HTML – HiperText Markup Language
- Es un lenguaje simple utilizado para crear documentos Web de extensión *.html o *.htm
- Se limita a describir la estructura y el contenido de un documento.
- Actualmente se encuentra en su versión 4.01
- Es un lenguaje basado en etiquetas. Cada etiqueta se escribe entre los caracteres: < >.
- Las etiquetas pueden tener atributos.

Estructura básica

- Las etiquetas suelen ir formando bloques: <> y </>.
- Esqueleto básico:

```
<HTML>
  <HEAD>
    <TITLE>Mi primer HTML</TITLE>
  </HEAD>
  <BODY>

  </BODY>
</HTML>
```

Formularios HTML

- La etiqueta <FORM> define un formulario.
- Permite los siguientes atributos:
  - Method: especifica el método HTTP a utilizar.
  - Action: especifica la URL a invocar.
  - Target, Name, OnSubmit…
- Ejemplo:

```
<FORM method="POST" action="/TestingWeb/TestingServlet">
…
…
</FORM>
```

Formularios HTML (cont.)

- La etiqueta <INPUT> define un campo del formulario y se sitúa dentro de este.
- Permite los siguientes atributos:
  - Name: especifica el nombre del campo.
  - Type: especifica el tipo de campo como text, password, submit, reset, checkbox, radio, hidden, image, etc…
  - Value: especifica el valor por defecto del campo.
- Ejemplo:

```
<INPUT name="param1" type="text" value="mi valor">
<INPUT name="param2" type="radio" value="mi valor">
<INPUT name="param3" type="hidden" value="mi valor">
<INPUT type="submit" value="enviar">
```

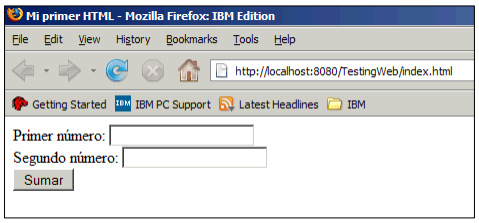
Formularios HTML (cont.)

- La etiqueta <SELECT> define un campo desplegable del formulario y también se sitúa dentro de este.
- Permite los siguiente atributos:
  - Name: especifica el nombre del campo.
  - Size: especifica el número de elementos.
- Incluye tags <OPTION value=“”> para listar los posibles valores.
- Ejemplo:

```
<SELECT name=“param1”>
  <OPTION value=“1” selected>Coche
  <OPTION value=“2”>Moto
</SELECT>
```

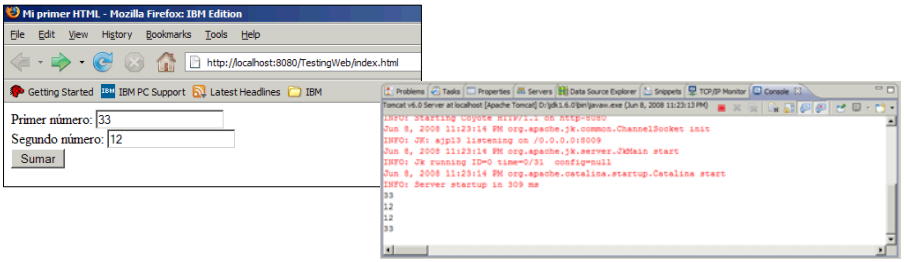
Ejemplo

```
<HTML>
 <HEAD>
  <TITLE>Mi primer HTML</TITLE>
 </HEAD>
 <BODY>

  <FORM method="post" action="/TestingWeb/TestingServlet">
     Primer número:<INPUT name="param1" type="text"><BR>
     Segundo número:<INPUT name="param2" type="text"><BR>
     <INPUT type="submit" value="Sumar">
  </FORM>

 </BODY>
</HTML>
```



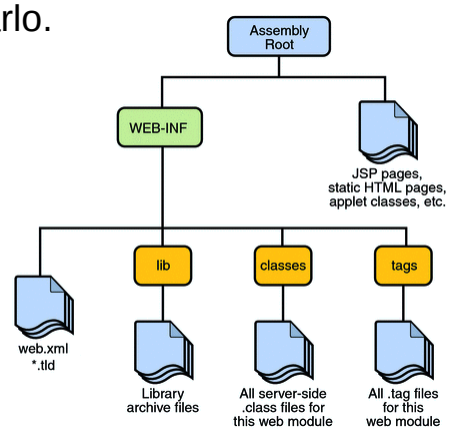
Ejemplo

```
import java.io.IOException;
import java.util.Enumeration;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class TestingServlet extends javax.servlet.http.HttpServlet
{
  protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
  {
    System.out.println(request.getParameter("param1"));
    System.out.println(request.getParameter("param2"));

    Enumeration e = request.getParameterNames();
    while(e.hasMoreElements())
      System.out.println(request.getParameter((String)e.nextElement()));
  }
}
```



Ensamblado

- Los Módulos Web se empaquetan en un archivo WAR (Web ARchive).
- Se puede usar la herramienta jar.exe incluida con la JDK para crearlo.
- Estructura de un archivo WAR:



Ensamblado (cont.)

- El descriptor de despliegue del WAR se llama web.xml y se ubica en el directorio \WEB-INF:
- http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd
- Ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID"
version="2.5">
  <display-name>TestingWeb</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>
  <servlet>
    <description></description>
    <display-name>TestingServlet</display-name>
    <servlet-name>TestingServlet</servlet-name>
    <servlet-class>edu.upco.controller.TestingServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>TestingServlet</servlet-name>
    <url-pattern>/TestingServlet</url-pattern>
  </servlet-mapping>
</web-app>
```

Bibliografía

- Head First Servlets & JSP (2nd edition)
  Bryan Basham, Kathy Sierra y Bert Bates.
  O'Reilly.

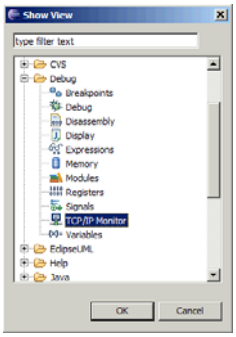
- Java Servlet Programming (2nd edition)
  Jason Hunter.
  O'Reilly.

- Java Servlets 2.3
  Andrew Harbourne-Thomas, Sam Dalton, Simon Brown, Bjarki Holm, Tony Loton, Meeraj Kunnumpurath, Subrahmanyam Allamaraju, John Bell y Sing Li.
  Wrox.

- The Java EE tutorial
  http://java.sun.com/javaee/5/docs/tutorial/doc/

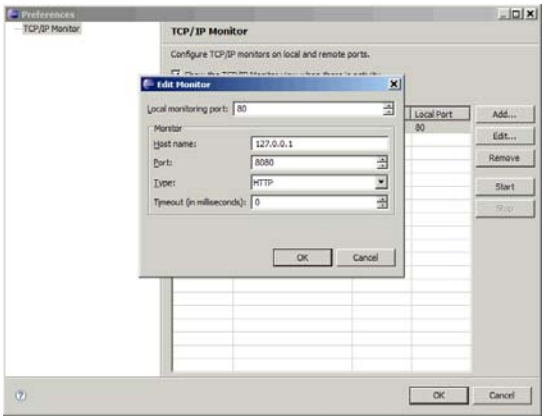
Apéndice A: TCP/IP Monitor

- Las herramientas WTP de Eclipse incluyen un sniffer TCP/IP para analizar el contenido que viaja por la red.
- Para utilizarlo ir al menú:
  Window -> Show View -> Other…



Apéndice A: TCP/IP Monitor

- En sus propiedades, seleccionar el puerto por el que escucha, y la dirección IP y puerto remoto a la que redireccionará el tráfico.



Apéndice A: TCP/IP Monitor

- Una vez lo arranquemos, todas las peticiones que enviemos a través del sniffer quedarán registradas pudiendo analizar las cabeceras y contenidos de los mensajes.