

Desarrollo de aplicaciones para Internet



Baltasar Fernández Manjón

Dpto. de Sistemas Informáticos y Programación,
Universidad Complutense de Madrid
Avda. Complutense s/n, 28040, Madrid, Spain,

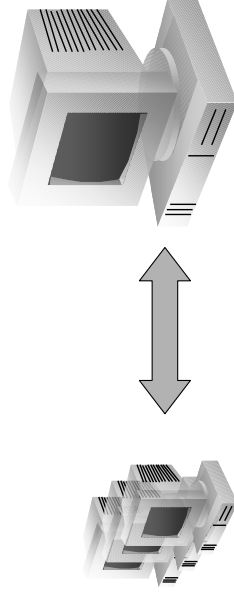
<http://bogart.sip.ucm.es/~balta>

Aplicaciones en Internet

- El desarrollo de aplicaciones en Internet es un ejemplo de desarrollo de aplicaciones distribuidas
- Soluciones y evolución
 - Aplicaciones monolíticas: grandes ordenadores (mainframes) y terminales
 - toda la lógica de proceso (negocio), el acceso a datos y la lógica de presentación están en el mainframe y los terminales sólo sirven para acceder y comunicarse con él.
 - Arquitectura de dos niveles
 - Proceso distribuido entre una máquina que proporciona el acceso mediante una interfaz de usuario (cliente) y una máquina que centraliza el servicio y los datos (servidor)
 - Arquitectura multinivel o multicapa
 - Se añade un nivel intermedio entre la interfaz de usuario y el servidor de datos. Este nivel se encarga de la lógica de negocio.

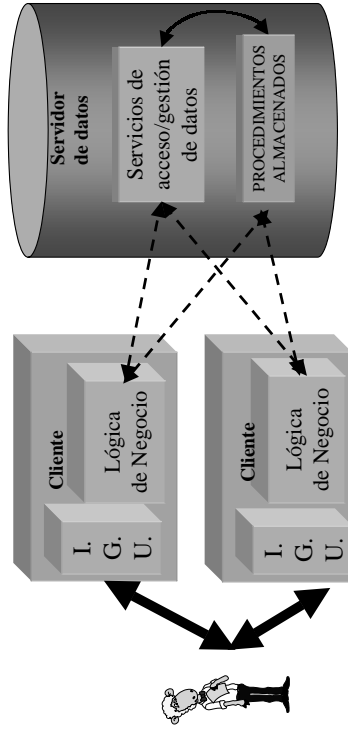
Aplicaciones monolíticas

- El acceso a los datos, la presentación (interfaz) y la lógica de proceso está en una única aplicación monolítica
- Existe un alto grado de acoplamiento entre las distintas partes de la aplicación lo que dificulta la reutilización de código y su mantenimiento.
- No se puede distribuir el código entre diversas máquinas y no es escalable



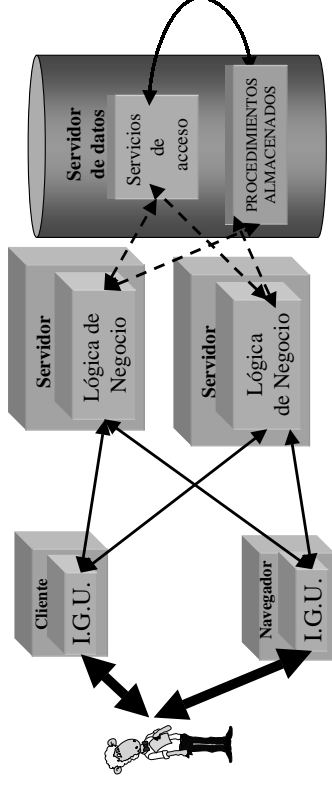
Arquitectura de dos niveles

- Se aprovecha la capacidad de proceso del cliente. El cliente es capaz de procesar datos (fat client).
- La distribución de la lógica de negocio entre el cliente y el servidor dificulta su actualización y mantenimiento
 - Hay que instalar nuevas versiones en el cliente
- Dificultades de escalado y alto tráfico de datos



Arquitectura multinivel

- Cliente ligero que se ocupa únicamente de la interfaz gráfica de usuario
- Proporciona una alta escalabilidad y posibilidad de distribución entre varias máquinas



5

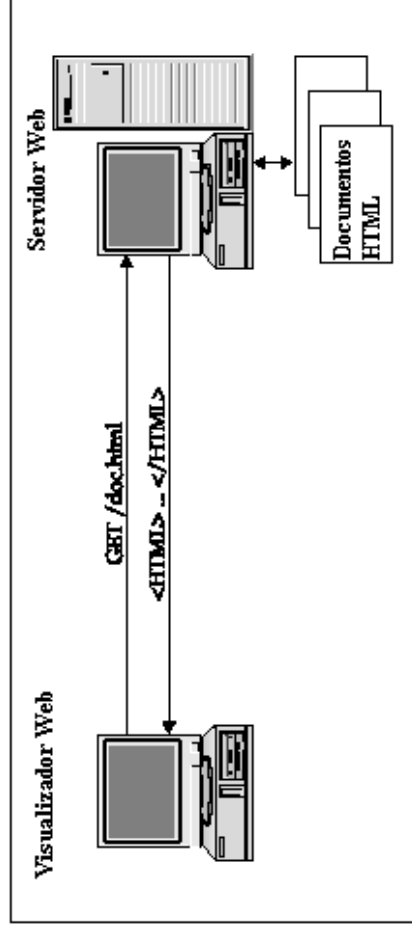
Particularidades de las aplicaciones web

- Las aplicaciones Web son aplicaciones distribuidas muy diferentes a las clásicas, con nuevos problemas y particularidades
- Podemos tener millones en vez de cientos de clientes y por supuesto más concurrencia que en un modelo cliente/servidor clásico. La escalabilidad es un problema constante.
- El protocolo HTTP no está orientado a sesión, hay que buscar formas de simularla.
- Todo el proceso se realiza on-line (transacciones, seguridad) y es crítico si falla perdemos al cliente
- La interfaz de usuario es impersonal, debemos prestar mayor atención al cliente, nuestra competencia es internacional no local.
- El tiempo de desarrollo de las aplicaciones debe ser rápido debido a la competencia

6

Modelo de web estático

- Modelo cliente servidor en el que un servidor web proporciona páginas estáticas a un navegador



7

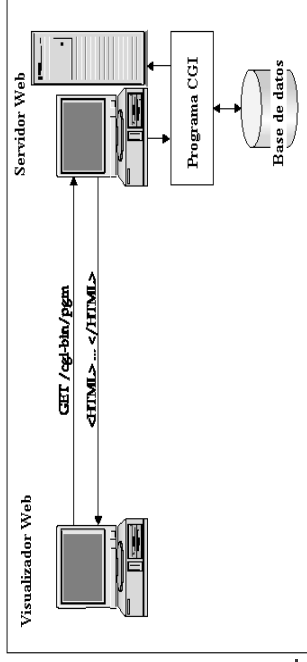
Evolución del servidor web estático

- Se ha evolucionado en los dos extremos
 - Ampliación de las capacidades del cliente
 - Permitiendo la ejecución de código en el cliente que proporcionan tecnologías como:
 - ♦ las applets – programas escritos en Java que se ejecutan dentro del navegador que actúa como cliente
 - ♦ los lenguajes de secuencias de órdenes o scripts (p.e. javascript) que son interpretados por el navegador
 - Ampliación de las capacidades del servidor
 - El servidor como respuesta a una petición ya no sólo es capaz de devolver un documento sino que también es capaz de ejecutar un programa
 - ♦ Interfaz de pasarela común (CGI, Common Gateway Interface)
 - ♦ Programas java que se ejecutan en el servidor (p.e. Servlets, JSP)

8

Common Gateway Interface (CGI)

- Aplicaciones específicas que se ejecutan en el servidor y que conectan el servidor web a las aplicaciones existentes.
- A pesar de ser antiguas es de las más utilizadas hoy en día.
- Presenta varios problemas :
 - ❑ Problemas de escalabilidad, concurrencia y recursos
 - ❑ Mal manejo de sesiones, es responsabilidad del programador
 - ❑ Actualización del código difícil



©Baltasar Fernández Manjón 9

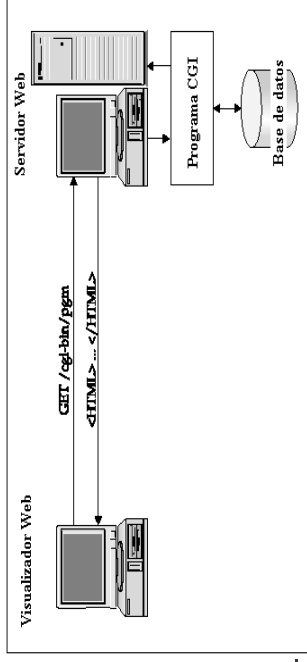
Aplicaciones a medida y soluciones propietarias

- Aplicaciones creadas para cubrir unas necesidades específicas.
- Gran esfuerzo. Pueden convertirse en muy complejas y largas si las empezamos desde cero
- Hay que considerar todos los aspectos generales y que no son los centrales de la aplicación:
 - ❑ Comunicación
 - ❑ Seguridad
 - ❑ Escalabilidad
 - ❑ Etc
- Para simplificar toda esta parte de infraestructura surgen las soluciones propietarias
 - ❑ Se proporcionan API's que simplifican el desarrollo de las aplicaciones
 - ❑ Proporcionan un "middleware"

©Baltasar Fernández Manjón 10

Common Gateway Interface (CGI)

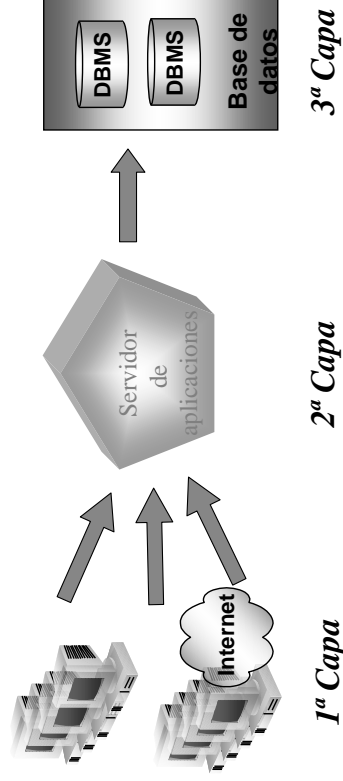
- Aplicaciones específicas que se ejecutan en el servidor y que conectan el servidor web a las aplicaciones existentes.
- A pesar de ser antiguas es de las más utilizadas hoy en día.
- Presenta varios problemas :
 - ❑ Problemas de escalabilidad, concurrencia y recursos
 - ❑ Mal manejo de sesiones, es responsabilidad del programador
 - ❑ Actualización del código difícil



©Baltasar Fernández Manjón 9

Servidores de aplicaciones

- Software de infraestructura o "Middleware" que permite desarrollar y desplegar aplicaciones distribuidas en varias capas.
- Centraliza servicios de aplicación como funcionalidad de servidor web, componentes de negocio y acceso a sistemas empresariales de soporte ("backend").
- Proporciona servicios de seguridad y facilidades de administración.



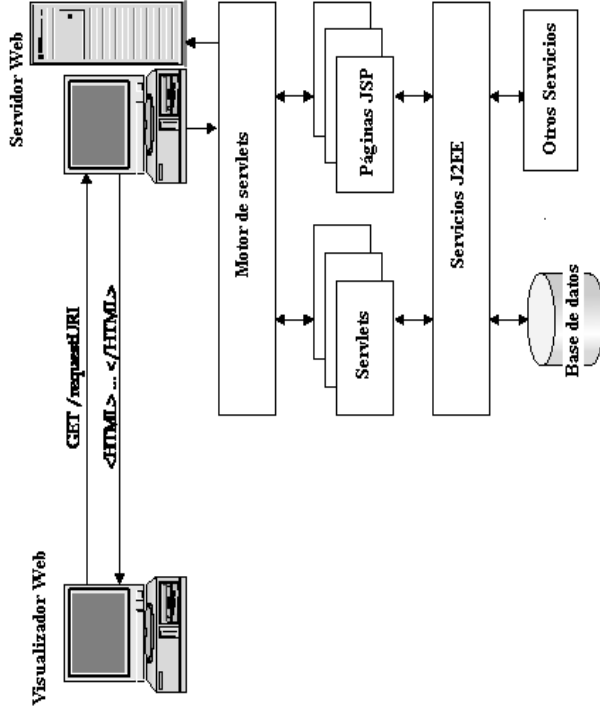
©Baltasar Fernández Manjón 11

Servidores de aplicaciones

- Hay varios tipos de Servidores de aplicaciones como:
 - ❑ Servidores CORBA
 - ❑ Vignette
 - ❑ BroadVision
 - ❑ .NET (Microsoft)
 - ❑ J2EE (Java to Enterprise Edition)
- Algunos servidores de aplicaciones son propietarios (dependen de una única empresa) mientras que otros siguen estándares abiertos
- J2EE
 - ❑ Plataforma estándar para el desarrollo de aplicaciones empresariales multicapa basadas en Java.
 - ❑ Promovido por Sun y otras empresas (IBM, BEA Systems, etc).
 - ❑ Se basa en la utilización de componentes modulares, y maneja parte de la funcionalidad de las aplicaciones de forma declarativa, sin necesidad de programación.

©Baltasar Fernández Manjón 12

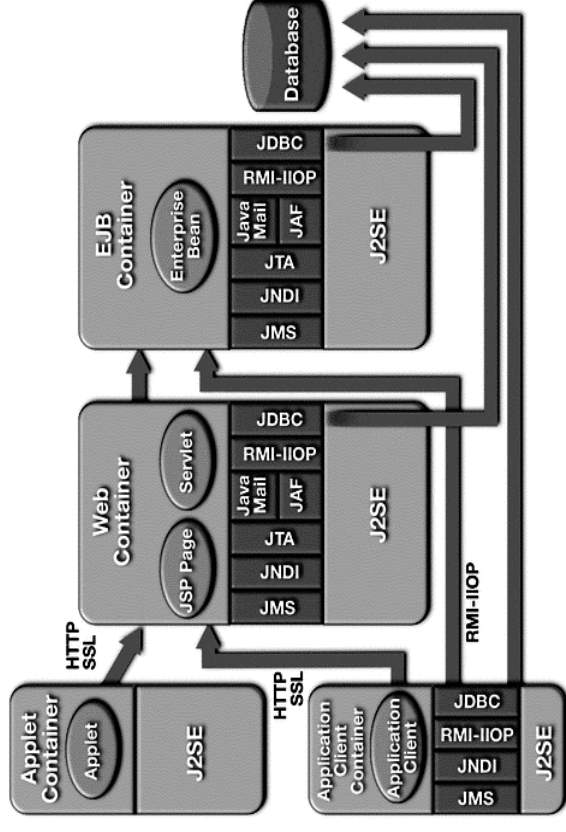
Servidores de aplicaciones J2EE



13

©Baltasar Fernández Manjón

Arquitectura J2EE



14

©Baltasar Fernández Manjón

Aplicación Web

- Una aplicación web es una extensión dinámica a un servidor web (Sun)
- Una aplicación web es un grupo de recursos en el servidor que permite crear una aplicación interactiva (Bea Systems)
- Estos recursos son componentes Web (p.e. servlet, JSP), documentos estáticos (p.e. html, imágenes), clases que se ejecutan en el servidor, bibliotecas de clases, e información de configuración y despliegue.
- Los componentes Web se ejecutan dentro de un entorno denominado contenedor web.
 - Este contenedor proporciona los servicios de infraestructura básicos para la aplicación web (p.e. Seguridad, concurrencia o gestión del ciclo de vida)

15

©Baltasar Fernández Manjón

Servlets

- Objetos Java que se basan en el API servlet y permiten extender la funcionalidad de un servidor HTTP
 - Permiten generar contenido a partir de un programa como respuesta a una petición HTTP
- Se ejecutan mediante una asignación con una URL
- Son gestionados por un contenedor que tiene una arquitectura simple
- Disponibles en la mayoría de los servidores de aplicaciones del mercado (y en todos los J2EE)
- Independientes del servidor y de la plataforma

16

©Baltasar Fernández Manjón

- JSP son documentos de texto que crean dinámicamente páginas web a partir de una petición de un cliente (navegador)
 - Se ejecutan en el servidor
- Un documento JSP contiene
 - Plantillas de texto conteniendo formatos de presentación (HTML, XML)
 - Acciones dinámicas bien sea
 - mediante código o
 - mediante etiquetas JSP que permiten en acceso a directivas o a otros componentes

17

©Baltasar Fernández Manjón

- Normalmente los clientes web utilizan el protocolo HTTP para comunicarse con un servidor de aplicaciones J2EE.
- El protocolo HTTP define
 - las peticiones que puede enviar un cliente al servidor. Cada petición contiene una URL que identifica el componente u objeto estático solicitado
 - Las respuestas que puede enviar el servidor a dichas peticiones
- El servidor J2EE convierte la petición en un objeto de petición HTTP y lo envía al componente web identificado por la URL. El componente web rellena un objeto de respuesta HTTP que el servidor convierte en una respuesta HTTP que se envía al cliente.

18

©Baltasar Fernández Manjón

HTTP peticiones y respuestas

- Una petición HTTP consta de un método de petición, de una URL, de campos de cabecera y de un cuerpo.

○ Métodos de petición de HTTP 1.1

- GET – recupera el recurso identificado por la URL de la petición
- HEAD – devuelve las cabeceras identificadas por la URL de la petición
- POST – envía datos de longitud ilimitada al servidor Web
- PUT – almacena un recurso en la URL de la petición
- DELETE – elimina el recurso identificado por URL de la petición
- OPTIONS – devuelve los métodos HTTP soportados por el servidor
- TRACE – devuelve los campos de cabecera enviados con una petición TRACE

- Una respuesta HTTP contiene un código de resultado, unos campos de cabecera y un cuerpo

- Algunos códigos de resultado habituales son:
 - 404 – indica que el recurso solicitado no está disponible
 - 401 – indica que la petición requiere autenticación HTTP
 - 500 – indica un error en el servidor HTTP que impide dar respuesta a la petición

19

©Baltasar Fernández Manjón

Modelo de petición HTTP

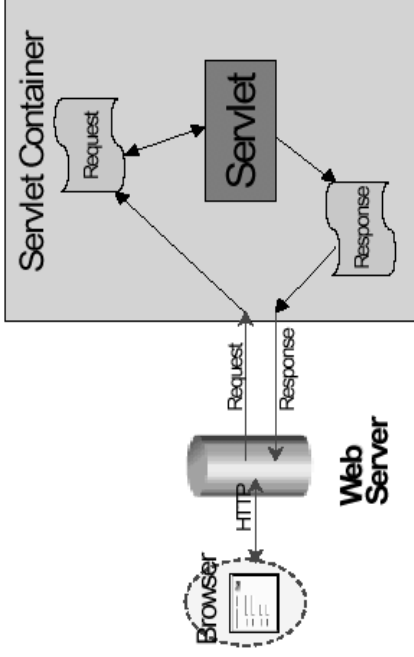
- La especificación describe HTTP como un protocolo de petición / respuesta sin estado cuya operación básica es la siguiente:

1. Una aplicación realizada por un cliente, por ejemplo un visualizador Web, abre un conector al puerto HTTP del servidor Web (el puerto predeterminado es el 80).
2. A través de la conexión el cliente escribe una línea de petición de texto ASCII, seguida de ninguna, una o varias cabeceras HTTP, una línea en blanco, y cualquier dato que acompañe a la petición.
3. El servidor Web analiza la petición y localiza el recurso especificado.
4. El servidor envía una copia del recurso al conector, donde es leído por el cliente.
5. El servidor cierra la conexión.

20

©Baltasar Fernández Manjón

Petición y respuesta con servlets

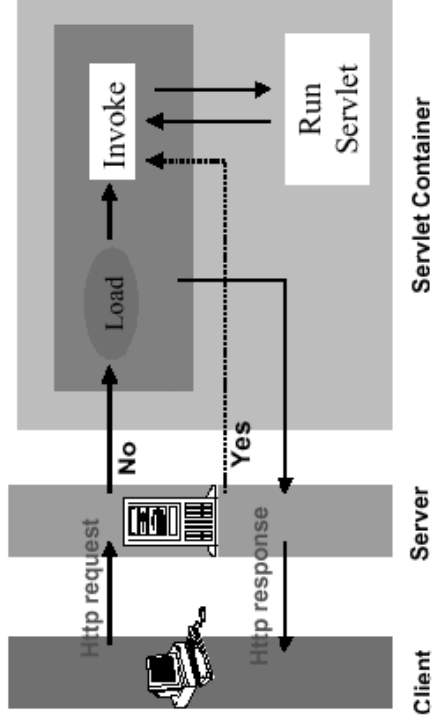


21

©Baltasar Fernández Manjón

Carga y ejecución de un servlet

- ❑ Como respuesta a una petición que implica un servlet, el server lo instancia (lo carga en memoria) y crea un hilo de ejecución. El servlet se carga sólo una vez y permanece en memoria hasta que se desactiva expresamente o se detiene el servidor

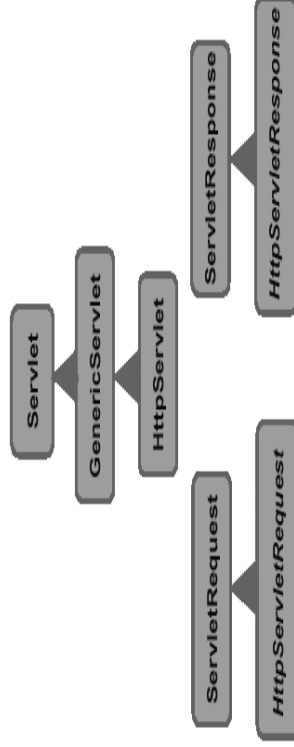


22

©Baltasar Fernández Manjón

Clases relacionadas con los servlets

- La API de servlet es un conjunto de clases e interfaces java que definen una interfaz estándar entre un cliente web y un servlet
 - ❑ Paquetes javax.servlet y javax.servlet.http
- Todos los servlets deben implementar el interface java Servlet que define el ciclo de vida de los servlets
 - ❑ `init()`, `service()`, `destroy()`



23

©Baltasar Fernández Manjón

Métodos definidos en la interface servlet de java.servlet

Method Summary	
<code>void</code>	<code>destroy()</code> Called by the servlet container to indicate to a servlet that the servlet is being taken out of service.
<code>ServletConfig</code>	<code>getServletConfig()</code> Returns a <code>ServletConfig</code> object, which contains initialization and startup parameters for this servlet.
<code>java.lang.String</code>	<code>getServletInfo()</code> Returns information about the servlet, such as author, version, and copyright.
<code>void</code>	<code>init(ServletConfig config)</code> Called by the servlet container to indicate to a servlet that the servlet is being placed into service.
<code>void</code>	<code>service(ServletRequest req, ServletResponse res)</code> Called by the servlet container to allow the servlet to respond to a request.

24

©Baltasar Fernández Manjón

Clases y métodos habituales en los servlets

- Una forma simple de obtener un servlet es mediante extensión de las clases genéricas que se proporcionan en la API

- GenericServlet Clase de soporte genérico de servlets (con independencia del protocolo):
- HttpServlet Clase para servlets que soportan el protocolo HTTP

- La especialización de estas clases debe sobreescibir por lo menos uno de los métodos que proporcionan

- La gestión de las peticiones al servlet se realizan invocando los métodos:

- Servlets genéricos
 - el método `service()`
- Servlets HTTP
 - métodos principales: `doGet()`, `doPost()`,
 - Otros métodos:
 - ♦ `doHead()`, `doDelete()`, `doOptions()`, `doTrace()`

©Baltasar Fernández Manjón 25

Servlets HTTP

- Normalmente se extiende la clase abstracta HttpServlet
- Una petición de un cliente a un servlet se representa y encapsula mediante un objeto HttpServletRequest
- Una respuesta desde un servlet a un cliente se representa y encapsula mediante un objeto HttpServletResponse
- Habitualmente se sobreescibe el método `doGet()` o el método `doPost()`
 - Una petición GET se produce cuando se proporciona una URL o se selecciona un enlace
 - Una petición POST se produce cuando el usuario envía un formulario HTML que especifica el método POST

©Baltasar Fernández Manjón 26

Métodos de HttpServlet

Method Summary	
protected void	<code>doDelete</code> (<u>HttpServletRequest</u> req, <u>HttpServletResponse</u> resp) Called by the server (via the service method) to allow a servlet to handle a DELETE request.
protected void	<code>doGet</code> (<u>HttpServletRequest</u> req, <u>HttpServletResponse</u> resp) Called by the server (via the service method) to allow a servlet to handle a GET request.
protected void	<code>doHead</code> (<u>HttpServletRequest</u> req, <u>HttpServletResponse</u> resp) Receives an HTTP HEAD request from the protected service method and handles the request.
protected void	<code>doOptions</code> (<u>HttpServletRequest</u> req, <u>HttpServletResponse</u> resp) Called by the server (via the service method) to allow a servlet to handle a OPTIONS request.
protected void	<code>doPost</code> (<u>HttpServletRequest</u> req, <u>HttpServletResponse</u> resp) Called by the server (via the service method) to allow a servlet to handle a POST request.
protected void	<code>doPut</code> (<u>HttpServletRequest</u> req, <u>HttpServletResponse</u> resp) Called by the server (via the service method) to allow a servlet to handle a PUT request.
protected void	<code>doTrace</code> (<u>HttpServletRequest</u> req, <u>HttpServletResponse</u> resp) Called by the server (via the service method) to allow a servlet to handle a TRACE request.
protected long	<code>getLastModified</code> (<u>HttpServletRequest</u> req) Returns the time the <u>HttpServletRequest</u> object was last modified, in milliseconds since midnight January 1, 1970 GMT.
protected void	<code>service</code> (<u>HttpServletRequest</u> req, <u>HttpServletResponse</u> resp) Receives standard HTTP requests from the public service method and dispatches them to the <code>doXXX</code> methods defined in this class.
void	<code>service</code> (<u>ServletRequest</u> req, <u>ServletResponse</u> res) Dispatches client requests to the protected service method.

©Baltasar Fernández Manjón 27

Ejemplo de Servlet HTTP

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

Public class HelloServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<title>El primer servlet</title>");
        out.println("<big>Hola Hola</big>");
    }
}
```

©Baltasar Fernández Manjón 28

