Introducción a los Servlets

Sistemas Uni

Que es un Servlet

- Los Servlets son módulos que extienden los servidores orientados a petición-respuesta, como los servidores web compatibles con Java. Por ejemplo, un servlet podría ser responsable de tomar los datos de un formulario de entrada de pedidos en HTML y aplicarle la lógica de negocios utilizada para actualizar la base de datos de pedidos de la compañia.

Que es un Servlet

- Programas en Java que se ejecutan en un servidor HTTP (servidor Web)
- Actúan como capa intermedia entre:
  - Petición proveniente de un Navegador Web u otro cliente HTTP
  - Bases de Datos o Aplicaciones en el servidor HTTP

Que puede hacer un servlet

- Leer los datos enviados por un usuario
  - Usualmente de formularios en páginas Web
  - Pueden venir de applets de Java o programas cliente HTTP.
- Buscar cualquier otra información sobre la petición que venga incluida en esta
  - Detalles de las capacidades del navegador, cookies, nombre del host del cliente, etc.
- Generar los resultados
  - Puede requerir consults a Base de Datos, invocar a otras aplicaciones, computar directamente la respuesta, etc.
- Dar formato a los resultados en un documento
  - Incluir la información en una página HTML
- Establecer los parámetros de la respuesta HTTP
  - Decirle al navegador el tipo de documento que se va a devolver, establecer las cookies, etc.
- Enviar el documento al cliente

Cuándo y por qué usar Servlets

- Muchas peticiones desde navegador se satisfacen retornando ***documentos HTML estáticos***, es decir, que están en ficheros
- En ciertos casos, es necesario generar las páginas HTML para cada petición:
  - **Página Web basada en datos enviados por el cliente**
    - Motores de búsqueda, confirmación de pedidos
  - **Página Web derivada de datos que cambian con frecuencia**
    - Informe del tiempo o noticias de última hora
  - **Página Web que usa información de bases de datos corporativas u otras fuentes del la parte del servidor**
    - Comercio electrónico: precios y disponibilidades

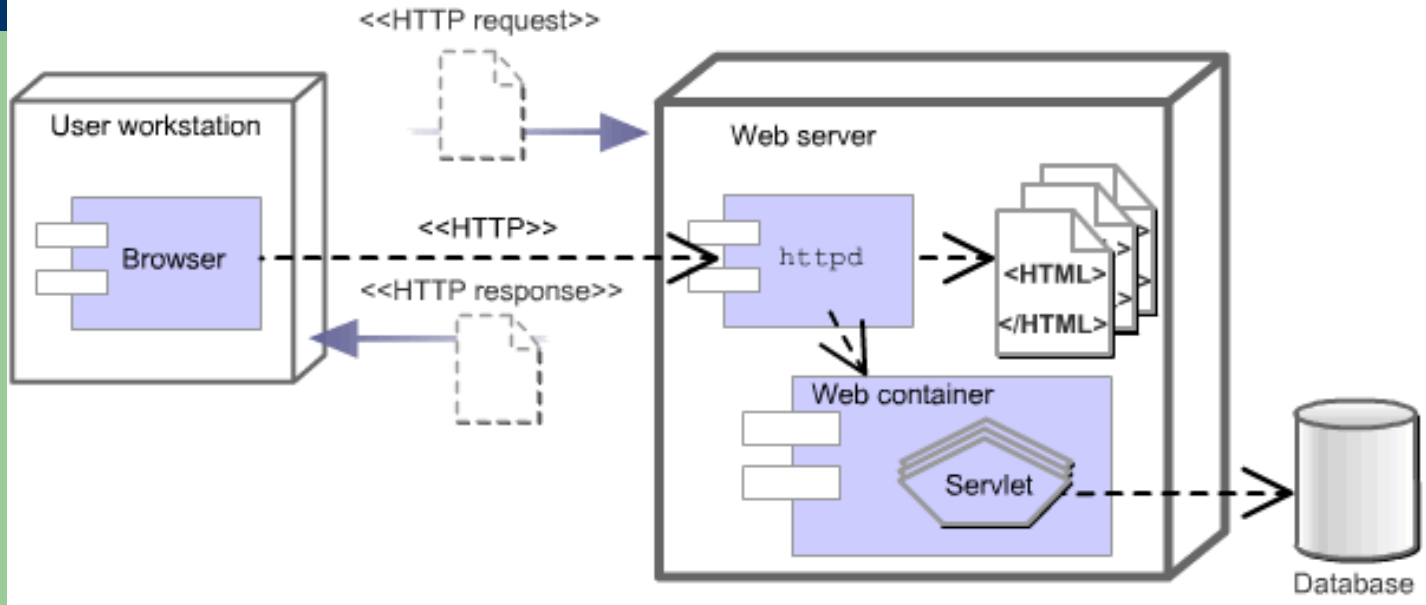
Ventajas sobre CGI

- Eficiencia
  - CGI corto: el proceso de arranque de cada proceso puede dominar el tiempo de ejecución
  - N peticiones simultáneas: el código del CGI se carga en memoria N veces
  - Al terminar el proceso, el CGI se cierra: difícil persistencia de datos (conexiones a BD, caché...)
- Conveniencia
  - Los Servlets tienen una infraestructura muy amplia para la tratar automáticamente datos de formularios HTML, gestionar sesiones y otras utilidades de alto nivel.
- Potencia
  - Los Servlets pueden comunicar directamente con el navegador Web
  - Pueden mantener datos entre peticiones, simplificando el seguimiento de sesiones y operaciones de caché
  - Varios Servlets pueden compartir datos

Ventajas sobre CGI

- Portabilidad
  - Los Servlets están escritos en Java y siguen una API estándar.
  - Pueden funcionar sin ningún cambio en diferentes servidores
- Seguridad
  - CGI adolecen de vulnerabilidades porque:
    - Se ejecutan en el shell del SO
    - Pueden sufrir overflows por el lenguaje (C, C++, ...)
  - Los Servlets no sufren estos problemas
- Economía
  - Añadir soporte para Servlet a un servidor Web ya disponible tiene muy poco coste extra
  - Existen ciertos servidores web y servidores de servlet gratuitos para tráficos pequeños

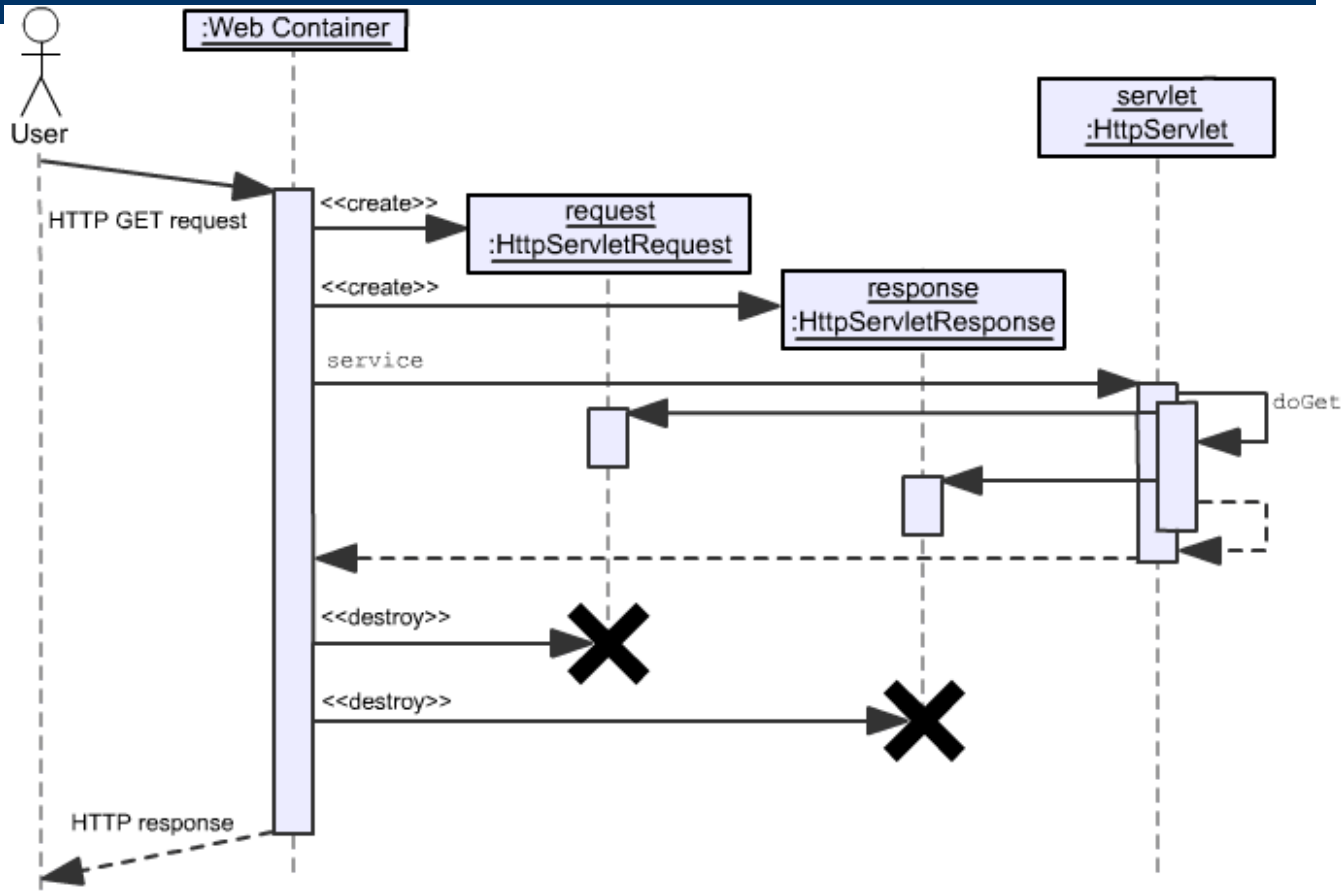
Arquitectura De Contenedor Web



Arquitectura de Contenedor Web

- El Web Container puede ser usado en conjunto con un servicio HTPP o puede ser usado como standalone Web Serve

Modo de Comunicacion



Modo de Comunicación

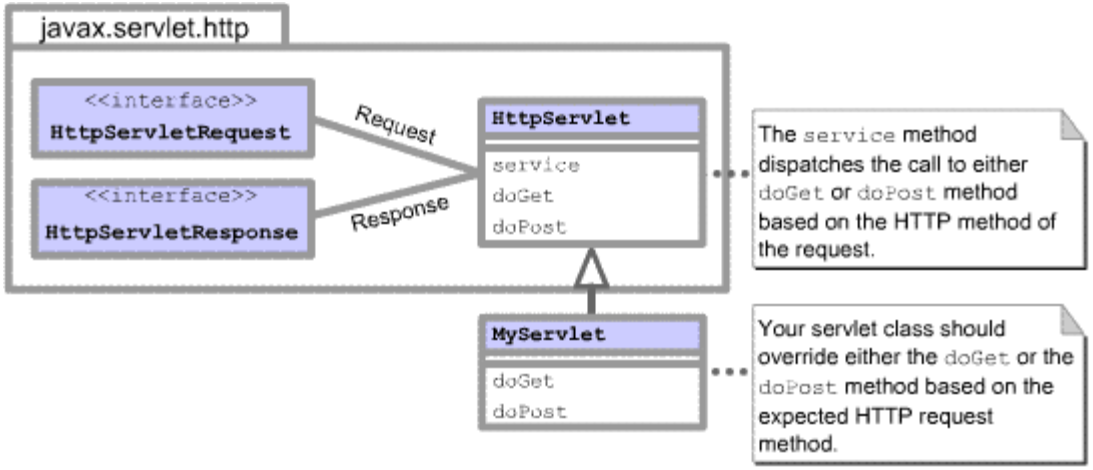
- El primer paso en este proceso es que el cliente envía una petición HTTP al servicio HTTP.
- El segundo paso es que el servicio HTTP transmite a los datos de la petición el Contenedor Web.
- En el tercer paso, el Contenedor Web crea un objeto que encapsule los datos del request stream. El Web Container además crea un objeto que encapsule el Stream Response.
- En el cuarto paso, el WebContainer ejecuta el método de servicio del servlet solicitado. Los objetos de la petición y de la respuesta se pasan como argumentos a este método. La ejecución del método de servicio ocurre en un hilo de separado.
- Finalmente, el texto de la respuesta generada por el servlet se empaqueta en una HTTP response Stream, que se envía al servicio HTTP y se remite al cliente.

Estructura de un HttpServlet

```
import java.io.*;
//Se importan los paquetes con las clases para Servlets y HttpServlets
import javax.servlet.*;
import javax.servlet.http.*;

public class ServletTemplate extends HttpServlet{

  //El método doGet responde a peticiones mediante el método GET
  public void doGet(HttpServletRequest request, HttpServletResponse response)
                                          throws ServletException, IOException {

    // El objeto "request" se usa para leer los "HTTP headers" que llegan
    // (p.e. Cookies) y los datos de formularios HTML enviados por el usuario

    // El objeto "response" se usa para especificar "HTTP status codes" y
    // "HTTP headers" de la respuesta (p.e. El tipo de contenido, cookies, etc.)

    PrintWriter out = response.getWriter();
    // El objeto "out" se usa para enviar contenido al navegador

  }

  //El método doPost responde a peticiones mediante el método POST
  public void doPost(HttpServletRequest request, HttpServletResponse response)
                                          throws ServletException, IOException {
    doGet(request, response);
  }
}
```

Estructura de un HttpServlet

sobreescribir el método service. Sin embargo, para los servicios HTTP, la API servlet proporciona una clase especial llamada HttpServlet a que se puede extender. Esta clase reemplaza la implementación por defecto del método service, así que no se debe sobreescribir este método.



Invocacion de un Servlet

- Invocación de un Servlet
  - **Desde la barra de direcciones del navegador**:

`http://hostname:port/servlet/Nombre_Servlet`

- Ejemplo:

- De esta forma se invoca el servlet mediante el método GET siempre

  - **Desde un formulario**:
    - La dirección del servlet debe ir en el **`action`**

```
<FORM action="http://hostname:port/servlet/Nombre_Servlet" method="POST">
...
</FORM>
```

- El servlet se invoca al hacer Submit y lo hace mediante el método definido en el formulario

Metodos

- Los métodos en los que delega el método service las peticiones HTTP, incluyen

- * doGet, para manejar GET, GET condicional, y peticiones de HEAD
- * doPost, para menajar peticiones POST
- * doPut, para manejar peticiones PUT
- * doDelete, para manejar peticiones DELETE

Manejar peticiones GET

- Manejar peticiones GET implica sobreescribir el método doGet. El siguiente ejemplo muestra a BookDetailServlet haciendo esto. Los métodos explicados en Peticiones y Respuestas se muestran en negrita.
- El servlet extiende la clase HttpServlet y sobreescribe el método doGet. Dentro del método doGet, el método getParameter obtiene los argumentos esperados por el servlet.

- Para responder al cliente, el método doGet utiliza un Writer del objeto HttpServletResponse para devolver datos en formato texto al cliente. Antes de acceder al writer, el ejemplo selecciona la cabecera del tipo del contenido. Al final del método doGet, después de haber enviado la respuesta, el Writer se cierra.

Manejar Peticiones GET

```
public class BookDetailServlet extends HttpServlet {

   public void doGet (HttpServletRequest request,
                      HttpServletResponse response)
      throws ServletException, IOException
   {
      ...
       // selecciona el tipo de contenido en la cabecera antes de acceder a Writer
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

       // Luego escribe la respuesta
      out.println("<html>" +
                  "<head><title>Book Description</title></head>" +
                  ...);

      //Obtiene el identificador del libro a mostrar
      String bookId = request.getParameter("bookId");
      if (bookId != null) {
         // Y la información sobre el libro y la imprime
         ...
      }
      out.println("</body></html>");
      out.close();
   }
   ...
}
```

Manejar Peticiones POST

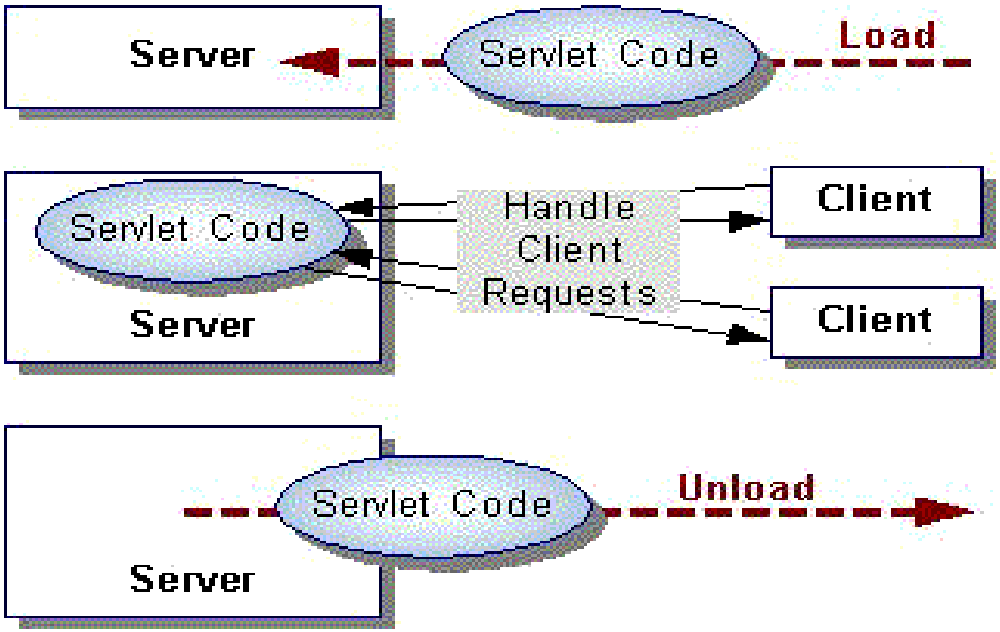
- Manejar peticiones POST implica sobreescribir el método doPost. El siguiente ejemplo muestra a ReceiptServlet haciendo esto. De nuevo, los métodos explicados en Peticiones y Respuestas se muestran en negrita.
- El servlet extiende la clase HttpServlet y sobreescribe el método doPost. Dentro del método doPost, el método getParameter obtiene los argumentos esperados por el servlet.

- Para responder al cliente, el método doPost utiliza un Writer del objeto HttpServletResponse para devolver datos en formato texto al cliente. Antes de acceder al writer, el ejemplo selecciona la cabecera del tipo de contenido. Al final del método doPost, después de haber enviado la respuesta, el Writer se cierra.

Manejar Peticiones POST

```
public class ReceiptServlet extends HttpServlet {

   public void doPost(HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException
   {
      ...
      // selecciona la cabecera de tipo de contenido antes de acceder a Writer
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      // Luego escribe la respuesta
      out.println("<html>" +
                  "<head><title> Receipt </title>" +
                  ...);

      out.println("<h3>Thank you for purchasing your books from us " +
                  request.getParameter("cardname") +
                  ...);
      out.close();
   }
   ...
}
```

Ciclo de Vida de un servlet



Ciclo de Vida de un Servlet

- Viene dado por tres métodos: `init`, `service` y `destroy`
- **INICIALIZACIÓN**: Una única llamada al método "`init`" por parte del servidor. Incluso se pueden recoger unos parametros concretos con "`getInitParameter`" de "`ServletConfig`".
- **SERVICIO**: una llamada a `service()` por cada invocación al servlet
  - ¡Cuidado! El contenedor es multihilo
- **DESTRUCCIÓN**: Cuando todas las llamadas desde el cliente cesen o un temporizador del servidor así lo indique. Se usa el método "`destroy`"
- Revisar documentación de la clase **`javax.servlet.Servlet`**

Estructura de Directorio

- El directorio raíz contiene un directorio especial llamado WEB-INF. Este directorio no es visible para los usuarios de la aplicación, sin embargo, aquí se guardan todas las clases, servlets y archivos JAR de los que conste una aplicación web. Dentro del directorio WEB-INF hay dos subdirectorios y un archivo que son de especial interés:
- * classes - Este directorio contiene servlets y otras clases. Estas clases se hayan automáticamente por el cargador de servlets como si estuvieran en la ruta de clases (CLASSPATH). Puede incluir subdirectorios que correspondan a la estructura de paquetes.
- * lib - Es similar al directorio anterior, pero contiene únicamente archivos JAR.
- * web.xml - Es un documento XML llamado descriptor de despliegue. Tiene una estructura rigurosamente definida que se usa para configurar los servlets y otros recursos que forman parte de una aplicación web.
- (ver aplicacion y ejemplo para mayor claridad)