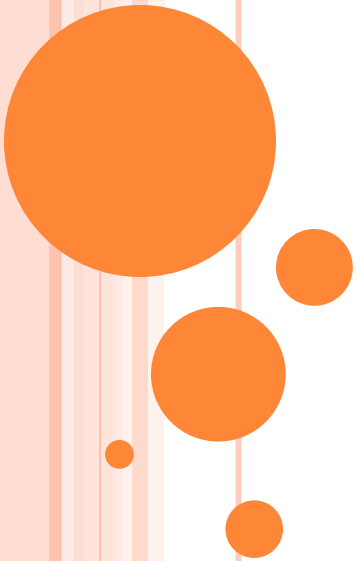


Servlets

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 Contenedor Web
- Actúan como capa intermedia entre:
  - Petición proveniente de un Navegador Web u otro cliente HTTP
  - Bases de Datos o Aplicaciones en el servidor (contenedor EJB, otra aplicación…)

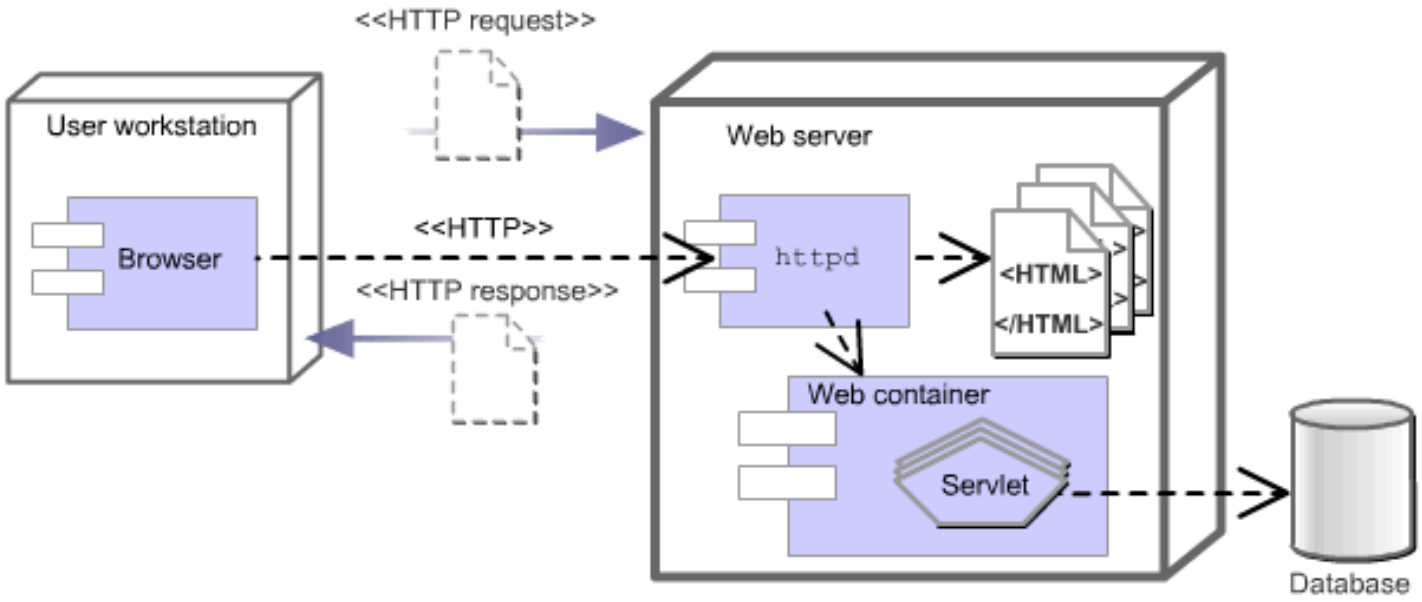
Que puede hacer un servlet

- 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 consultas 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

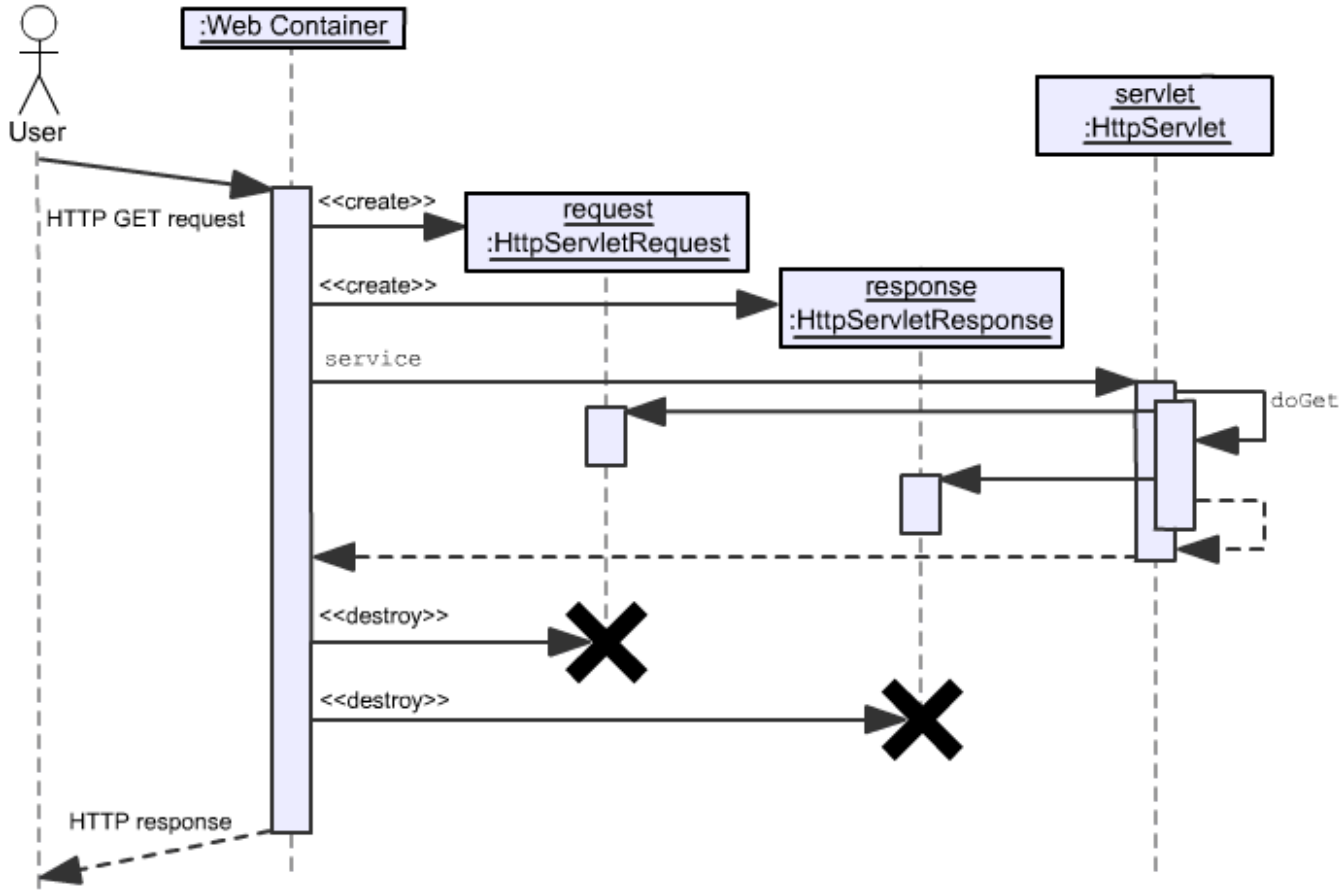
Cuándo y por qué usar Servlets

- 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

Arquitectura De Un servidor con Contenedor Web



Modo de Comunicacion



Modo de Comunicación

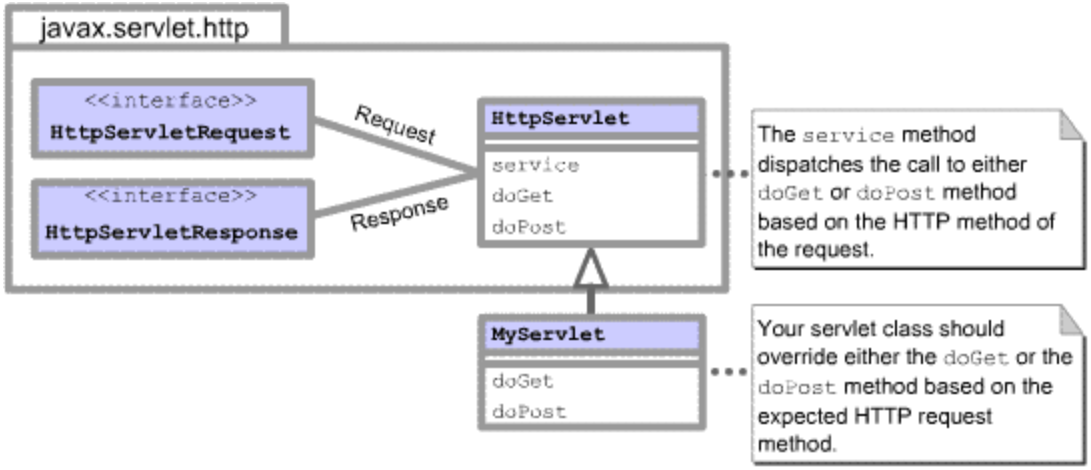
- El primer paso en este proceso es que el cliente envía una petición HTTP al servicio HTTP.
- El segundo paso es que el servicio HTTP transmite a los datos de la petición el Contenedor Web.
- En el tercer paso, el Contenedor Web crea un objeto que encapsule los datos del request stream. El Web Container 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 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

- Al desarrollar servlets genéricos, se debe crear una subclase de GenericServlet y 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/nombre_aplicacion/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/nombre_aplicacion/Nombre_Servlet"
   method="POST">
...
</FORM>
```

    - El servlet se invoca al hacer Submit y lo hace mediante el método definido en el formulario
    - Al servlet se le pasan los valores de los campos

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

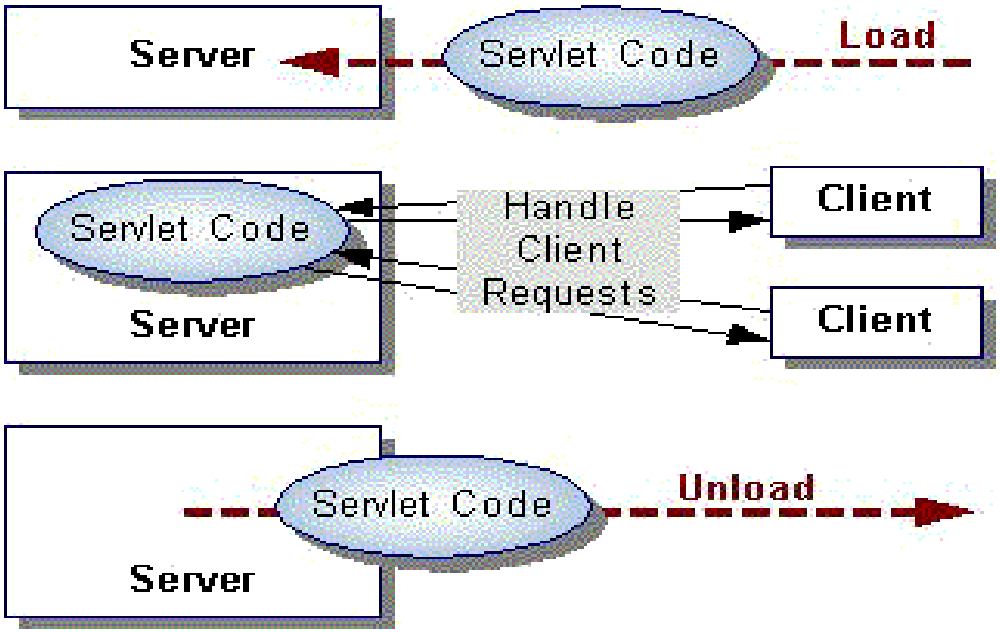
- Manejar peticiones POST implica sobreescribir el método doPost. El siguiente ejemplo muestra a ReceiptServlet haciendo esto. De nuevo, los métodos explicados en Peticiones y Respuestas se muestran en negrita.
- El servlet extiende la clase HttpServlet y sobreescribe el método doPost. Dentro del método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

Seguimiento de session

- La API servlet ofrece un mecanismo para manejar una sesión de un cliente.
- El servlet puede mantener información sobre múltiples páginas y a través de muchas transacciones según navegue el usuario.

- En muchas aplicaciones es importante ofrecer continuidad por medio de una serie de páginas web como seguir compras en un carro de la compra.

Seguimiento de session

- Cada navegador dispone de su propio objeto javax.servlet.http.HttpSession en el servidor
- Es posible enganchar objetos a una sesión y recuperarlos con los métodos:
  - void setAttribute(String, Object)
  - Object getAttribute(String)
- Por motivos de escalabilidad y de que en HTTP no hay nada especial que indique que un navegador ha dejado de usar la aplicación web, cada sesión dispone de un timeout (en minutos)
  - Si transcurre el timeout sin que el navegador acceda a la aplicación, el servidor destruye la sesión

Seguimiento de Session

- Ejemplo de uso:
  - Cada vez que un usuario hace un login, crearemos un a sesión, le engancharemos su nombre de login, y lo redirigiremos a la página principal del portal
  - Cada vez que el usuario accede a la página principal del portal, si ya ha hecho el login, se le saludará por su nombre (se recupera de la sesión)
  - Cada vez que el usuario accede a la página principal del portal, si todavía no ha hecho el login, o su sesión ha caducado, se le redirigirá a la página de login
  - Cuando un usuario hace un logout, le destruiremos la sesión y lo redirigiremos a la página principal

Seguimiento de Session

- ¿ Cómo sabe el servidor de aplicaciones a qué sesión está asociada cada petición HTTP que recibe ? Mediante los siguientes mecanismos:
  - Por medio de cookies: que son pequeños ficheros de texto que guardan información sobre nuestra sesión
  - Por medio reescritura de la URL: se escribe datos de la sesión después de la URL
  - Por medio de campos ocultos en formularios que guardan los datos asociados a nuestra sesión

Manejo de la sesión

- Para manejar la sesión

- Introducir los datos de la sesión en el objeto request:

```
HttpSession session = request.getSession(true);
session.setAttribute("login", login);
```

- Obtener los datos de la sesión del objeto request:

```
HttpSession session = request.getSession(false);
if (session == null) {
return null;
} else{
return (String)
session.getAttribute("login");
}
```

Redirigir la salida desde un servlet

- **sendRedirect**
  - Le decimos al navegador que nos haga una nueva petición a otra URL

    response.sendRedirect("pago")
- **Forward**

  RequesteDispatcher rd=getServletContext().getRequesteDispatcher("/pago.jsp"
  rd.forward(request,response)
  - Nos movemos a otra URL dentro del servidor.
  - Se conserva todo lo que había en la request
  - Útil para tratar errores en formularios
  - Los servlets de procesamiento insertan el atributo errores (un Map) en la request
  - Los servlets que muestran formularios comprueban si la request incluye el atributo errores

Redirigir la salida desde un servlet

- **¿ Cuándo usar forward y cuándo sendRedirect?**
- En principio, un **forward** siempre es más rápido (ocurre en el servidor)
- Un forward es preciso cuando queremos enganchar atributos a la request
  - Ej.: Tratamiento de errores en formularios
- Para el resto de situaciones, es mejor usar un **sendRedirect**, dado que forward no cambia la URL que muestra la caja de diálogo delnavegador

Ejercicio

- Convierte la aplicación java de base de datos de Agenda(versión JDBC) en una aplicación web a través del uso de Servlets:
  - Debe haber una página que muestre todos los contactos ordenadas de alfabeticamente y permita editarlas: añadir/modificar/eliminar datos del contacto.
  - Debe haber una página que muestre todos contactos que cumplan años en un determinado mes que se solicitará en un formulario.
  - El usuario debe poder añadir y eliminar contactos.

Ejercicio Plus

- La Agenda ahora es online, no solamente es usada por una persona como hasta ahora. Amplía el ejercicio para gestionar agendas de varios Usuarios.
- Tendrás que gestionar el registro de usuarios, así como su logueo (cada usuario solo debe ver sus contactos) y logout.