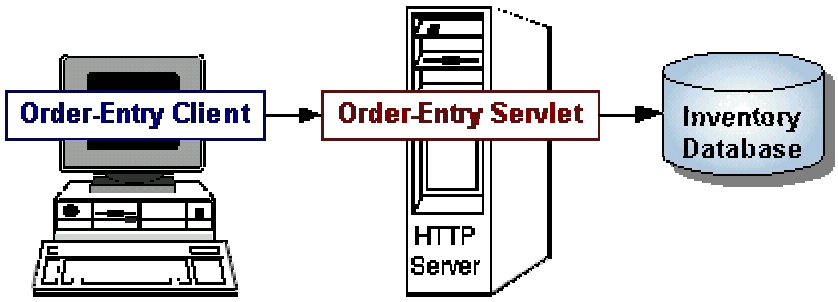
Clase 11: Servlets

11.1 Introducción a los Servlets

Los Servlets son módulos que extienden los servidores orientados a pedido-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 compañía.



Los Servlets son para los servidores lo que los applets son para los navegadores. Sin embargo, al contrario que los applets, los servlets no tienen interface gráfica de usuario.
Los servlets pueden ser incluidos en muchos servidores diferentes porque el API Servlet, el que se utiliza para escribir Servlets, no asume nada sobre el entorno o protocolo del servidor. Los servlets se están utilizando ampliamente dentro de servidores HTTP; muchos servidores Web soportan el API Servlet.

11.1.1 Utilizar Servlets en lugar de Scripts CGI!

Los Servlets son un reemplazo efectivo para los scripts CGI. Proporcionan una forma de generar documentos dinámicos que son fáciles de escribir y rápidos en ejecutarse. Los Servlets también solucionan el problema de hacer la programación del lado del servidor con APIs específicos de la plataforma: están desarrollados con el API Java Servlet, una extensión estándar de Java.
Por eso se utilizan los servlets para manejar peticiones de cliente HTTP. Por ejemplo, tener un servlet procesando datos POSTeados sobre HTTP utilizando un formulario HTML, incluyendo datos del pedido o de la tarjeta de crédito. Un servlet como este podría ser parte de un sistema de procesamiento de pedidos, trabajando con bases de datos de productos e inventarios, y quizás un sistema de pago on-line.

11.1.2 Otros usos de los Servlets

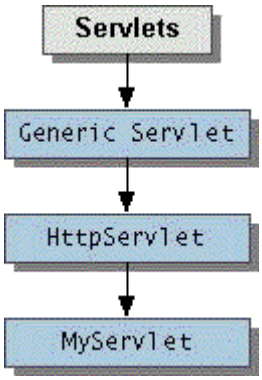
- Permitir la colaboración entre la gente. Un servlet puede manejar múltiples peticiones concurrentes, y puede sincronizarlas. Esto permite a los servlets soportar sistemas como conferencias on-line
- Reenviar peticiones. Los Servlets pueden reenviar peticiones a otros servidores y servlets. Con esto los servlets pueden ser utilizados para cargar balances desde varios servidores que reflejan el mismo contenido, y para particionar un único servicio lógico en varios servidores, de acuerdo con los tipos de tareas o la organización compartida.

11.2 Arquitectura del Paquete Servlet

El paquete **javax.servlet** proporciona clases e interfaces para escribir servlets. La arquitectura de este paquete se describe a continuación.

11.2.1 La Interface Servlet

La abstracción central en el API Servlet es la interface **Servlet**. Todos los servlets implementan esta interface, bien directamente o, más comúnmente, extendiendo una clase que lo implemente como **HttpServlet**



La interface **Servlet** declara, pero no implementa, métodos que manejan el Servlet y su comunicación con los clientes. Los escritores de Servlets proporcionan algunos de esos métodos cuando desarrollan un servlet.

11.2.2 Interacción con el Cliente

Cuando un servlet acepta una llamada de un cliente, recibe dos objetos.

- Un **ServletRequest**, que encapsula la comunicación desde el cliente al servidor.
- Un **ServletResponse**, que encapsula la comunicación de vuelta desde el servlet hacia el cliente.

ServletRequest y **ServletResponse** son interfaces definidos en el paquete **javax.servlet**.

11.2.2.1 La Interface ServletRequest

La Interface **ServletRequest** permite al servlet acceder a :

- Información como los nombres de los parámetros pasados por el cliente, el protocolo (esquema) que está siendo utilizado por el cliente, y los nombres del host remote que ha realizado la petición y la del server que la ha recibido.
- El stream de entrada, **ServletInputStream**. Los Servlets utilizan este stream para obtener los datos desde los clientes que utilizan protocolos como los métodos POST y PUT del HTTP.

Las interfaces que extienden la interface **ServletRequest** permiten al servlet recibir más datos específicos del protocolo. Por ejemplo, la interface **HttpServletRequest** contiene métodos para acceder a información de cabecera específica HTTP.

11.2.2.2 La Interface ServletResponse

La Interface **ServletResponse** le da al servlet los métodos para responder al cliente.

- Permite al servlet seleccionar la longitud del contenido y el tipo MIME de la respuesta.
- Proporciona un stream de salida, **ServletOutputStream**, y un **Writer** a través del cual el servlet puede responder datos.

Las interfaces que extienden la interface **ServletResponse** le dan a los servlets más capacidades específicas del protocolo. Por ejemplo, la interface **HttpServletResponse** contiene métodos que permiten al servlet manipular información de cabecera específica HTTP.

11.2.3 Capacidades Adicionales de los Servlets HTTP

Las clases e interfaces descritos anteriormente construyen un servlet básico. Los servlets HTTP tienen algunos objetos adicionales que proporcionan capacidades de seguimiento de sesión. El escritor se servlets pueden utilizar esos APIs para mantener el estado entre el servlet y el cliente persiste a través de múltiples conexiones durante un periodo de tiempo. Los servlets HTTP también tienen objetos que

proporcionan cookies. El API cookie se utiliza para guardar datos dentro del cliente y recuperar esos datos.

11.3 Un Servlet Sencillo

La siguiente clase define completamente un servlet.

```
public class SimpleServlet extends HttpServlet
{
   /**
    * Maneja el método GET de HTPP para construir una sencilla página Web.
    */
   public void doGet (HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
   {
         PrintWriter out;
         String   title = "Simple Servlet Output";

         // primero selecciona el tipo de contenidos y otros campos de cabecera de la
          // respuesta
         response.setContentType("text/html");

         // Luego escribe los datos de la respuesta
         out = response.getWriter();
         out.println("<HTML><HEAD><TITLE>");
         out.println(title);
         out.println("</TITLE></HEAD><BODY>");
         out.println("<H1>" + title + "</H1>");
         out.println("<P>This is output from SimpleServlet.");
         out.println("</BODY></HTML>");
         out.close();
   }
}
```

Las clases mencionadas en la sección **Arquitectura del Paquete Servlet** se han mostrado en negrita.

- **SimpleServlet** extiende la clase **HttpServlet**, que implementa la interface **Servlet**.
- **SimpleServlet** sobreescribe el método **doGet** de la clase **HttpServlet**. Este método es llamado cuando un cliente hace un petición GET (el método de petición por defecto de HTTP), y resulta en una sencilla página HTML devuelta al cliente.
- Dentro del método **doGet**
    - La petición del usuario está representada por un objeto **HttpServletRequest**.
    - La respuesta al usuario esta representada por un objeto **HttpServletResponse**.
    - Como el texto es devuelto al cliente, el respuesta se envía utilizando el objeto **Writer** obtenido desde el objeto **HttpServletResponse**.

11.4 Ejemplos de Servlets

Las páginas utilizan un ejemplo llamado **Librería de Duke**, una sencilla librería on-line que permite a los clientes realizar varias funciones. Cada función está proporcionada por un Servlet.

Función	Servlet
Navegar por los libros de oferta	**CatalogServlet**
Comprar un libro situándolo en un "tarjeta de venta"	**CatalogServlet**
Obtener más información sobre un libro específico	**BookDetailServlet**
Manejar la base de datos de la librería	**BookDBServlet**
Ver los libros que han sido seleccionados para comprar	**ShowCartServlet**
Eliminar uno o más libros de la tarjeta de compra.	**ShowCartServlet**
Comprar los libros de la tarjeta de compra	**CashierServlet**
Recibir un Agradecimiento por la compra	**ReceiptServlet**

Las páginas utilizan servlets para ilustrar varias tareas. Por ejemplo, el BookDetailServlet se utiliza para mostrar cómo manejar peticiones GET de HTTP, el BookDBServlet se utiliza para mostrar cómo inicializar un servlet, y el CatalogServlet se utiliza para mostrar el seguimiento de sesión.

11.5 Interactuar con los Clientes

Un Servlet HTTP maneja peticiones del cliente a través de su método **service**. Este método soporta peticiones estándar de cliente HTTP despachando cada petición a un método designado para manejar esa petición. Por ejemplo, el método **service** llama al método **doGet** mostrado anteriormente en el ejemplo del servlet sencillo.

11.5.1 Peticiones y Respuestas

Esta página explica la utilización de los objetos que representan peticiones de clientes (un objeto **HttpServletRequest**) y las respuestas del servlet (un objeto **HttpServletResponse**). Estos objetos se proporcionan al método **service** y a los métodos que **service** llama para manejar peticiones HTTP.

11.5.2 Manejar Peticiones GET y POST

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 manejar peticiones POST
- **doPut**, para manejar peticiones PUT
- **doDelete**, para manejar peticiones DELETE

Por defecto, estos métodos devuelven un error **BAD_REQUEST (400)**. Nuestro servlet debería sobrescribir el método o métodos diseñados para manejar las interacciones HTTP que soporta. Esta sección muestra cómo implementar método para manejar las peticiones HTTP más comunes: GET y POST.
El método **service** de **HttpServlet** también llama al método **doOptions** cuando el servlet recibe una petición OPTIONS, y a **doTrace** cuando recibe una petición TRACE. La implementación por defecto de **doOptions** determina automáticamente que opciones HTTP son soportadas y devuelve esa información. La implementación por defecto de **doTrace** realiza una respuesta con un mensaje que contiene todas las cabeceras enviadas en la petición trace. Estos métodos no se sobrescriben normalmente.

11.5.3 Problemas con los Threads

Los Servlets HTTP normalmente pueden servir a múltiples clientes concurrentes. Si los métodos de nuestro Servlet no funcionan con clientes que acceden a recursos compartidos, deberemos.

- Sincronizar el acceso a estos recursos, o
- Crear un servlet que maneje sólo una petición de cliente a la vez.

En esta clase se muestra cómo implementar la segunda opción.

11.5.4 Descripciones de Servlets

Además de manejar peticiones de cliente HTTP, los servlets también son llamados para suministrar descripción de ellos mismos. Esta página muestra como proporcionar una descripción sobrescribiendo el método **getServletInfo**, que suministra una descripción del servlet.

11.6 Peticiones y Respuestas

Los métodos de la clase **HttpServlet** que manejan peticiones de cliente toman dos argumentos.

1. Un objeto **HttpServletRequest**, que encapsula los datos desde el cliente.
2. Un objeto **HttpServletResponse**, que encapsula la respuesta hacia el cliente.

11.6.1 Objetos HttpServletRequest

Un objeto **HttpServletRequest** proporciona acceso a los datos de cabecera HTTP, como cualquier cookie encontrada en la petición, y el método HTTP con el que se ha realizado la petición. El objeto **HttpServletRequest** también permite obtener los argumentos que el cliente envía como parte de la petición.
Para acceder a los datos del cliente

- El método **getParameter** devuelve el valor de un parámetro nombrado. Si nuestro parámetro pudiera tener más de un valor, deberíamos utilizar **getParameterValues** en su lugar. El método **getParameterValues** devuelve un array de valores del parámetro nombrado. (El método **getParameterNames** proporciona los nombres de los parámetros.
- Para peticiones GET de HTTP, el método **getQueryString** devuelve en un **String** una línea de datos desde el cliente. Debemos analizar estos datos nosotros mismos para obtener los parámetros y los valores.
- Para peticiones POST, PUT, y DELETE de HTTP.
    - Si esperamos los datos en formato texto, el método **getReader** devuelve un **BufferedReader** utilizado para leer la línea de datos.
    - Si esperamos datos binarios, el método **getInputStream** devuelve un **ServletInputStream** utilizado para leer la línea de datos.

Nota: Se debe utilizar el método **getParameter[Values]** o uno de los métodos que permitan analizar los datos. No pueden utilizarse juntos en una única petición.

11.6.2 Objetos HttpServletResponse

Un objeto **HttpServletResponse** proporciona dos formas de devolver datos al usuario.

- El método **getWriter** devuelve un **Writer**
- El método **getOutputStream** devuelve un **ServletOutputStream**

Se utiliza el método **getWriter** para devolver datos en formato texto al usuario y el método **getOutputStream** para devolver datos binarios.
Si cerramos el **Writer** o el **ServletOutputStream** después de haber enviado la respuesta, permitimos al servidor saber cuando la respuesta se ha completado.

11.6.3 Cabecera de Datos HTTP

Debemos seleccionar la cabecera de datos HTTP antes de acceder a **Writer** o a **OutputStream**. La clase **HttpServletResponse** proporciona métodos para acceder a los datos de la cabecera. Por ejemplo, el método **setContentType** selecciona el tipo del contenido. (Normalmente esta es la única cabecera que se selecciona manualmente).

11.7 Manejar Peticiones GET y POST

Para manejar peticiones HTTP en un servlet, extendemos la clase **HttpServlet** y sobrescribimos los métodos del servlet que manejan las peticiones HTTP que queremos soportar. Esta página ilustra el manejo de peticiones GET y POST. Los métodos que manejan estas peticiones son **doGet** y **doPost**.

11.7.1 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**.

```
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();
   }
   ...
}
```

El servlet extiende la clase **HttpServlet** y sobrescribe 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.

11.7.2 Manejar Peticiones POST

Manejar peticiones POST implica sobrescribir el método **doPost**. El siguiente ejemplo muestra a ReceiptServlet haciendo esto. De nuevo, los métodos explicados en Peticiones y Respuestas se muestran en **negrita**.

```
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();
   }
   ...
}
```

El servlet extiende la clase **HttpServlet** y sobrescribe 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.

11.8 Problemas con los Threads

Los servlets HTTP normalmente pueden servir a múltiples clientes concurrentemente. Si los métodos de nuestro servlet trabajan con clientes que acceden a recursos compartidos, podemos manejar la concurrencia creando un servlet que maneje sólo una petición de cliente a la vez. (También se puede sincronizar el acceso a los recursos, un punto que se cubre en al sección Threads de Control de esta clase).
Para hacer que el servlet maneje sólo un cliente a la vez, tiene que implementar la interface **SingleThreadModel** además de extender la clase **HttpServlet**.
Implementar la interface **SingleThreadModel** no implica escribir ningún método extra. Sólo se declara que el servlet implementa la interface, y el servidor se asegura de que nuestro servlet sólo ejecute un método **service** cada vez.
Por ejemplo, el ReceiptServlet acepta un nombre de usuario y un número de tarjeta de crédito, y le agradece al usuario su pedido. Si este servlet actualizara realmente una base de datos, por ejemplo, una que siga la pista del inventario, entonces la conexión con la base de datos podría ser un recurso compartido. El servlet podría sincronizar el acceso a ese recurso, o implementar la interface **SingleThreadModel**. Si el servlet implementa este interface, el único cambio en el código es la línea mostrada en **negrita**.

```
   public class ReceiptServlet extends HttpServlet
                        implements SingleThreadModel {

      public void doPost(HttpServletRequest request,
                         HttpServletResponse response)
                  throws ServletException, IOException {
            ...
      }
      ...
   }
```

11.9 Proporcionar Información de un Servlet

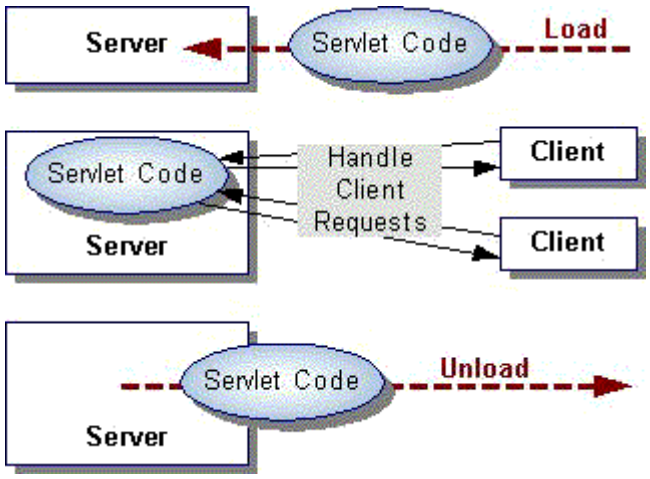
Algunas aplicaciones, obtienen información sobre el servlet y la muestran. La descripción del servlet es un string que puede describir el propósito del servlet, su autor, su número de versión, o aquello que el autor del servlet considere importante.
El método que devuelve esta información es **getServletInfo**, que por defecto devuelve **null**. No es necesario sobrescribir este método, pero las aplicaciones no pueden suministrar descripción de nuestro servlet a menos que nosotros lo hagamos.
El siguiente ejemplo muestra la descripción de BookStoreServlet.

```
public class BookStoreServlet extends HttpServlet {
    ...
    public String getServletInfo() {
        return "The BookStore servlet returns the main web page for Duke's Bookstore.";
    }
}
```

11.10 El Ciclo de Vida de un Servlet

Cada servlet tiene el mismo ciclo de vida.

- Un servidor carga e inicializa el servlet.
- El servlet maneja cero o más peticiones de cliente.
- El servidor elimina el servlet. (Algunos servidores sólo cumplen este paso cuando se desconectan).



11.10.1 Inicializar un Servlet

Cuando un servidor carga un servlet, ejecuta el método **init** del servlet. La inicialización se completa antes de manejar peticiones de clientes y antes de que el servlet sea destruido.
Aunque muchos servlets se ejecutan en servidores multi-thread, los servlets no tienen problemas de concurrencia durante su inicialización. El servidor llama sólo una vez al método **init**, cuando carga el servlet, y no lo llamará de nuevo a menos que vuelva a recargar el servlet. El servidor no puede recargar un servlet sin primero haber destruido el servlet llamando al método **destroy**.

11.10.2 Interactuar con Clientes

Después de la inicialización, el servlet puede manejar peticiones de clientes. Esta parte del ciclo de vida de un servlet se pudo ver en la sección anterior.

11.10.3 Destruir un Servlet

Los servlets se ejecutan hasta que el servidor los destruye, por ejemplo, a petición del administrador del sistema. Cuando un servidor destruye un servlet, ejecuta el método **destroy** del propio servlet. Este método sólo se ejecuta una vez. El servidor no ejecutará de nuevo el servlet, hasta haberlo cargado e inicializado de nuevo.
Mientras se ejecuta el método **destroy**, otro thread podría estar ejecutando una petición de servicio. La página **Manejar Threads de Servicio a la Terminación de un Thread** muestra como proporcionar limpieza cuando threads de larga ejecución podrían estar ejecutando peticiones de servicio.

11.11 Inicializar un Servlet

El método **init** proporcionado por la clase **HttpServlet** inicializa el servlet y graba la inicialización. Para hacer una inicialización específica de nuestro servlet, debemos sobrescribir el método **init** siguiendo estas reglas.

- Si ocurre un error que haga que el servlet no pueda manejar peticiones de cliente, lanzar una **UnavailableException**.

  Un ejemplo de este tipo de error es la imposibilidad de establecer una conexión requerida.

- No llamar al método **System.exit**.
- Guardar el parámetro **ServletConfig** para que el método **getServletConfig** pueda devolver su valor.

  La forma más sencilla de hacer esto es hacer que el nuevo método **init** llame a **super.init**. Si grabamos el objeto nosotros mismos, debemos sobrescribir el método **getServletConfig** para devolver el objeto desde su nueva posición.

Aquí hay un ejemplo del método **init**.

```
public class BookDBServlet ... {

    private BookstoreDB books;

    public void init(ServletConfig config) throws ServletException {

        // Store the ServletConfig object and log the initialization
        super.init(config);

        // Load the database to prepare for requests
        books = new BookstoreDB();
    }
    ...
}
```

El método **init** es bastante sencillo: llama al método **super.init** para manejar el objeto ServletConfig y grabar la inicialización, y seleccionar un campo privado.
Si el BookDBServlet utilizará una base de datos real, en vez de simularla con un objeto, el método **init** sería más complejo. Aquí puedes ver el pseudo-código de como podría ser ese método **init**.

```
public class BookDBServlet ... {

    public void init(ServletConfig config) throws ServletException {

        // Store the ServletConfig object and log the initialization
        super.init(config);

        // Open a database connection to prepare for requests
        try {
            databaseUrl = getInitParameter("databaseUrl");
            ... // get user and password parameters the same way
```

```
            connection = DriverManager.getConnection(databaseUrl, user, password);
        } catch(Exception e) {
            throw new UnavailableException (this, "Could not open a connection");
        }
    }
    ...
}
```

11.11.1 Parámetros de Inicialización

Le segunda versión del método **init** llama al método **getInitParameter**. Este método toma el nombre del parámetro como argumento y devuelve un **String** que representa su valor.
(La especificación de parámetros de inicialización es específica del servidor. Por ejemplo, los parámetros son especificados como una propiedad cuando un servlet se ejecuta con el ServletRunner. La página **La Utilidad servletrunner** contiene una explicación general de las propiedades y cómo crearlas).
Si por alguna razón, necesitamos obtener los nombres de los parámetros, podemos utilizar el método **getParameterNames**.

11.12 Destruir un Servlet

El método **destroy** proporcionado por la clase **HttpServlet** destruye el servlet y graba su destrucción.
Para destruir cualquier recurso específico de nuestro servlet, debemos sobrescribir el método **destroy**. Este método debería deshacer cualquier trabajo de inicialización y cualquier estado de persistencia sincronizado con el estado de memoria actual.
El siguiente ejemplo muestra el método **destroy** que acompaña el método **init** de la página anterior.

```
public class BookDBServlet extends GenericServlet {

    private BookstoreDB books;

    ... // the init method

    public void destroy() {
        // Allow the database to be garbage collected
        books = null;
    }
}
```

Un servidor llama al método **destroy** después de que se hayan completado todas las llamadas de servidor, o en un servidor específico hayan pasado un número de segundos, lo que ocurra primero. Si nuestro servlet manejar operaciones de larga ejecución, los métodos **service** se podrían estar ejecutando cuando el servidor llame al método **destroy**. Somos responsables de asegurarnos de que todos los threads han terminado.
El método **destroy** mostrado arriba espera a que todas las interacciones de cliente se hayan completado cuando se llama al método **destroy**, porque el servlet no tiene operaciones de larga ejecución.

11.13 Manejar Threads de Servicio a la Terminación de un Servlet

Todos los métodos de servicio de un servlet deberían estar terminados cuando se elimina el servlet. El servidor intenta asegurarse llamando al método **destroy** sólo después de que todas las peticiones de servicio hayan retornado, o después del periodo de tiempo de gracia específico del servido, lo que ocurra primero. Si nuestro servlet tiene operaciones que tardan mucho tiempo en ejecutarse (esto es, operaciones que tardan más que el tiempo concedido por el servidor), estas operaciones podrían estar ejecutándose cuando se llame al método **destroy**. Debemos asegurarnos de que cualquier thread que maneje peticiones de cliente se hayan completado; el resto de está página describe una técnica para hacer esto.
Si nuestro servlet tiene peticiones de servicio potencialmente largas, debemos utilizar las técnicas de esta lección para.

- Seguir la pista de cuantos threads están ejecutando el método **service** actualmente.
- Proporcionar una limpieza de desconexión haciendo que el método **destroy** notifique a los threads la desconexión y espere a que ellos se hayan completado.
- Haciendo que todos los métodos de larga duración comprueben periódicamente la desconexión y, si es necesario, paren su trabajo, limpien y retornen.

11.13.1 Peticiones de Seguimiento de Servicio

Para seguir la pista a una petición de servicio, incluimos un campo en nuestra clase servlet que cuente el número de métodos de servicio que se están ejecutando. El campo deberá tener acceso a métodos para incrementar, decrementar y devolver su valor. Por ejemplo:

```
public ShutdownExample extends HttpServlet {
   private int serviceCounter = 0;
   ...
   //Access methods for serviceCounter
   protected synchronized void enteringServiceMethod() {
            serviceCounter++;
   }
   protected synchronized void leavingServiceMethod() {
      serviceCounter--;
   }
   protected synchronized int numServices() {
            return serviceCounter;
   }
}
```

El método **service** debería incrementar el contador de servicios cada vez que se entre en él y decrementarlo cada vez que se salga de él. Esta es una de las pocas veces que al subclasificar la clase **HttpServlet** debamos sobrescribir el método **service**. El nuevo método debería llamar al **super.service** para preservar la funcionalidad del método **HttpServlet.service** original.

```
   protected void service(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException
   {
            enteringServiceMethod();
            try {
                  super.service(req, resp);
            } finally {
                  leavingServiceMethod();
            }
   }
```

11.13.2 Proporcionar Limpieza a la Desconexión

Para proporcionar esta limpieza, nuestro método **destroy** no debería destruir ningún recurso compartido hasta que todas las peticiones de servicio se hayan completado. Una parte de esto es chequear el contador de servicios. Otra parte es notificar a los métodos de larga duración que es la hora de la desconexión. Para esto, se necesita otro campo con sus métodos de acceso normales. Por ejemplo:

```
public ShutdownExample extends HttpServlet {
   private Boolean shuttingDown;
   ...
   //Access methods for shuttingDown
   protected setShuttingDown(Boolean flag) {
            shuttingDown = flag;
   }
   protected Boolean isShuttingDown() {
            return shuttingDown;
   }
}
```

Abajo podemos ver un método **destroy** que utiliza estos campos para proporcionar una limpieza de desconexión.

```
public void destroy() {

    /* Check to see whether there are still service methods running,
         * and if there are, tell them to stop. */
        if (numServices() > 0) {
            setShuttingDown(true);
        }

        /* Wait for the service methods to stop.  */
        while(numServices() > 0) {
            try {
                Thread.sleep(interval);
             } catch (InterruptedException e) {  }
        }
}
```

11.13.3 Crear Métodos de Larga Duración Educados

El paso final para proporcionar una limpieza de desconexión es crear métodos de larga duración que sean educados. Estos métodos deberían comprobar el valor del campo que notifica las desconexiones, e interrumpir su trabajo si es necesario. Por ejemplo.

```
public void doPost(...) {
    ...
        for(i = 0; ((i < lotsOfStuffToDo) && !isShuttingDown()); i++) {
            try {
                partOfLongRunningOperation(i);
            } catch (InterruptedException e) {
        }
    }
}
```

11.14 Guardar el Estado del Cliente

El API Servlet proporciona dos formas de seguir la pista al estado de un cliente.

11.14.1 Seguimiento de Sesión

El seguimiento de sesión es un mecanismo que los servlets utilizan para mantener el estado sobre la serie de peticiones desde un mismo usuario (esto es, peticiones originadas desde el mismo navegador) durante algún periodo de tiempo,.

11.14.2 Cookies

Las Cookies son un mecanismo que el servlet utiliza para mantener en el cliente una pequeña cantidad de información asociada con el usuario. Los servlets pueden utilizar la información del cookie como las entradas del usuario en el site (como una firma de seguridad de bajo nivel, por ejemplo), mientras el usuario navega a través del site (o como expositor de las preferencias del usuario, por ejemplo) o ambas.

11.15 Seguimiento de Sesión

El seguimiento de sesión es un mecanismo que los servlets utilizan para mantener el estado sobre la serie de peticiones desde un mismo usuario (esto es, peticiones originadas desde el mismo navegador) durante un periodo de tiempo.
Las sesiones son compartidas por los servlets a los que accede el cliente. Esto es conveniente para aplicaciones compuestas por varios servlets. Por ejemplo, **Duke's Bookstore** utiliza seguimiento de sesión para seguir la pista de los libros pedidos por el usuario. Todos los servlets del ejemplo tienen acceso a la sesión del usuario.
Para utilizar el seguimiento de sesión debemos.

- Obtener una sesión (un objeto **HttpSession**) para un usuario.
- Almacenar u obtener datos desde el objeto **HttpSession**.
- Invalidar la sesión (opcional).

11.15.1 Obtener una Sesión

El método **getSession** del objeto **HttpServletRequest** devuelve una sesión de usuario. Cuando llamamos al método con su argumento **create** como **true**, la implementación creará una sesión si es necesario.
Para mantener la sesión apropiadamente, debemos llamar a **getSession** antes de escribir cualquier respuesta. (Si respondemos utilizando un **Writer**, entonces debemos llamar a **getSession** antes de acceder al **Writer**, no sólo antes de enviar cualquier respuesta).
El ejemplo **Duke's Bookstore** utiliza seguimiento de sesión para seguir la pista de los libros que hay en la hoja de pedido del usuario. Aquí tenemos un ejemplo de **CatalogServlet** obteniendo una sesión de usuario.

```
public class CatalogServlet extends HttpServlet {

    public void doGet (HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException
    {
        // Get the user's session and shopping cart
            HttpSession session = request.getSession(true);
        ...
            out = response.getWriter();
        ...
    }
}
```

11.15.2 Almacenar y Obtener Datos desde la Sesión

La Interface **HttpSession** proporciona métodos que almacenan y recuperan.

- Propiedades de Sesión Estándar, como un identificador de sesión.
- Datos de la aplicación, que son almacenados como parejas nombre-valor, donde el nombre es un **string** y los valores son objetos del lenguaje de programación Java. Como varios servlets pueden acceder a la sesión de usuario, deberemos adoptar una convención de nombrado para organizar los nombres con los datos de la aplicación. Esto evitará que los servlets sobrescriban accidentalmente otros valores de la sesión. Una de esas convenciones es **servletname.name** donde **servletname** es el nombre completo del servlet, incluyendo sus paquetes. Por ejemplo, **com.acme.WidgetServlet.state** es un cookie con el servletname **com.acme.WidgetServlet** y el name **state**.

El ejemplo **Duke's Bookstore** utiliza seguimiento de sesión para seguir la pista de los libros de la hoja de pedido del usuario. Aquí hay un ejemplo de **CatalogServlet** obteniendo un identificador de sesión de usuario, y obteniendo y seleccionando datos de la aplicación asociada con la sesión de usuario.

```
public class CatalogServlet extends HttpServlet {

    public void doGet (HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException
    {
        // Get the user's session and shopping cart
            HttpSession session = request.getSession(true);
```

```
            ShoppingCart cart = (ShoppingCart) session.getValue(session.getId());

        // If the user has no cart, create a new one
        if (cart == null) {
            cart = new ShoppingCart();
            session.putValue(session.getId(), cart);
        }
            ...
        }
    }
```

Como un objeto puede ser asociado con una sesión, el ejemplo **Duke's Bookstore** sigue la pista de los libros que el usuario ha pedido dentro de un objeto. El tipo del objeto es **ShoppingCart** y cada libro que el usuario a seleccionado es almacenado en la hoja de pedidos como un objeto **ShoppingCartItem**. Por ejemplo, el siguiente código procede del método **doGet** de **CatalogServlet**.

```
    public void doGet (HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException
    {
            HttpSession session = request.getSession(true);
            ShoppingCart cart = (ShoppingCart)session.getValue(session.getId());
        ...
        // Check for pending adds to the shopping cart
        String bookId = request.getParameter("Buy");

        //If the user wants to add a book, add it and print the result
        String bookToAdd = request.getParameter("Buy");
        if (bookToAdd != null) {
            BookDetails book = database.getBookDetails(bookToAdd);

            cart.add(bookToAdd, book);
            out.println("<p><h3>" + ...);
        }
    }
```

Finalmente, observa que una sesión puede ser designada como nueva. Una sesión nueva hace que el método **isNew** de la clase **HttpSession** devuelva **true**, indicando que, por ejemplo, el cliente, todavía no sabe nada de la sesión. Una nueva sesión no tiene datos asociados.
Podemos tratar con situaciones que involucran nuevas sesiones. En el ejemplo Duke's Bookstore, si el usuario no tiene hoja de pedido (el único dato asociado con una sesión), el servlet crea una nueva. Alternativamente, si necesitamos información sobre el usuario al iniciar una sesión (como el nombre de usuario), podríamos querer redireccionar al usuario a un "página de entrada" donde recolectamos la información necesaria.

11.15.3 Invalidar la Sesión

Una sesión de usuario puede ser invalidada manual o automáticamente, dependiendo de donde se esté ejecutando el servlet. (Por ejemplo, el Java Web Server, invalida una sesión cuando no hay peticiones de página por un periodo de tiempo, unos 30 minutos por defecto). Invalidar una sesión significa eliminar el objeto **HttpSession** y todos sus valores del sistema.
Para invalidar manualmente una sesión, se utiliza el método **invalidate** de "session". Algunas aplicaciones tienen un punto natural en el que invalidar la sesión. El ejemplo **Duke's Bookstore** invalida una sesión de usuario después de que el usuario haya comprado los libros. Esto sucede en el **ReceiptServlet**.

```
    public class ReceiptServlet extends HttpServlet {

        public void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

            ...
            scart = (ShoppingCart)session.getValue(session.getId());
```

```
            ...
            // Clear out shopping cart by invalidating the session
            session.invalidate();

            // set content type header before accessing the Writer
            response.setContentType("text/html");
            out = response.getWriter();
            ...
        }
    }
```

11.15.4 Manejar todos los Navegadores

Por defecto, el seguimiento de sesión utiliza cookies para asociar un identificador de sesión con un usuario. Para soportar también a los usuarios que acceden al servlet con un navegador que no soporta cookies, o si este está programado para no aceptarlas. debemos utilizar reescritura de URL en su lugar. Cuando se utiliza la reescritura de URL se llama a los métodos que, cuando es necesario, incluyen el ID de sesión en un enlace. Debemos llamar a esos métodos por cada enlace en la respuesta del servlet.
El método que asocia un ID de sesión con una URL es **HttpServletResponse.encodeUrl**. Si redireccionamos al usuario a otra página, el método para asociar el ID de sesión con la URL redireccionada se llama **HttpServletResponse.encodeRedirectUrl**.
Los métodos **encodeUrl** y **encodeRedirectUrl** deciden si las URL necesitan ser reescritas, y devolver la URL cambiada o sin cambiar. (Las reglas para las URLs y las URLs redireccionadas son diferentes, pero en general si el servidor detecta que el navegador soporta cookies, entonces la URL no se reescribirá).
El ejemplo **Duke's Bookstore** utiliza reescritura de URL para todos los enlaces que devuelve a sus usuarios. Por ejemplo, el **CatalogServlet** devuelve un catalogo con dos enlaces para cada libro. Un enlace ofrece detalles sobre el libro y el otro ofrece al usuario añadir el libro a su hoja de pedidos. Ambas URLs son reescritas.

```
   public class CatalogServlet extends HttpServlet {

      public void doGet (HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException
      {
         // Get the user's session and shopping cart, the Writer, etc.
         ...
               // then write the data of the response
         out.println("<html>" + ...);
         ...
         // Get the catalog and send it, nicely formatted
         BookDetails[] books = database.getBooksSortedByTitle();
         ...
         for(int i=0; i < numBooks; i++) {
            ...
            //Print out info on each book in its own two rows
            out.println("<tr>" + ...
                  "<a href=\"" +
                  response.encodeUrl("/servlet/bookdetails?bookId=" +
                              bookId) +
                  "\"> <strong>" + books[i].getTitle() +
                  "  </strong></a></td>" + ...
                  "<a href=\"" +
                  response.encodeUrl("/servlet/catalog?Buy=" + bookId)
                  + "\">   Add to Cart  </a></td></tr>" +

         }
      }
   }
```

Si el usuario pulsa sobre un enlace con una URL re-escrita, el servlet reconoce y extrae el ID de sesión. Luego el método **getSession** utiliza el ID de sesión para obtener el objeto **HttpSession** del usuario.
Por otro lado, si el navegador del usuario no soporta cookies y el usuario pulsa sobre una URL no re-escrita. Se pierde la sesión de usuario. El servlet contactado a través de ese enlace crea una nueva

sesión, pero la nueva sesión no tiene datos asociados con la sesión anterior. Una vez que un servlet pierde los datos de una sesión, los datos se pierden para todos los servlets que comparten la sesión. Debemos utilizar la re-escritura de URLs consistentemente para que nuestro servlet soporte clientes que no soportan o aceptan cookies.

11.16 Utilizar Cookies

Las Cookies son una forma para que un servidor (o un servlet, como parte de un servidor) envíe información al cliente para almacenarla, y para que el servidor pueda posteriormente recuperar esos datos desde el cliente. Los servlet envían cookies al cliente añadiendo campos a las cabeceras de respuesta HTTP. Los clientes devuelven las cookies automáticamente añadiendo campos a las cabeceras de peticiones HTTP.
Cada cabecera de petición o respuesta HTTP es nombrada como un sólo valor. Por ejemplo, una cookie podría tener un nombre de cabecera **BookToBuy** con un valor **304qty1**, indicando a la aplicación llamante que el usuario quiere comprar una copia del libro con el número 304 en el inventario. (Las cookies y sus valores son específicos de la aplicación).
Varias cookies pueden tener el mismo nombre. Por ejemplo, un servlet podría enviar dos cabeceras llamadas **BookToBuy**; una podría tener el valor anterior, **304qty1**, mientras que la otra podría tener el valor **301qty3**. Estas cookies podrían indicar que el usuario quiere comprar una copia del libro con el número 304 en el inventario y tres copias del libro con el número 301del inventario.
Además de un nombre y un valor, también se pueden proporcionar atributos opcionales como comentarios. Los navegadores actuales no siempre tratan correctamente a los atributos opcionales, por eso ten cuidado con ellos.
Un servidor puede proporcionar una o más cookies a un cliente. El software del cliente, como un navegador, se espera que pueda soportar veinte cookies por host de al menos 4 kb cada una.
Cuando se envía una cookie al cliente, el estándar HTTP/1.0 captura la página que no está en la caché. Actualmente, el **javax.servlet.http.Cookie** no soporta los controles de caché del HTTP/1.1.
Las cookies que un cliente almacena para un servidor sólo pueden ser devueltas a ese mismo servidor. Un servidor puede contener múltiples servlets; el ejemplo **Duke's Bookstore** está compuesto por varios servlets ejecutándose en un sólo servidor. Como las cookies son devueltas al servidor, los servlets que se ejecutan dentro de un servidor comparten las cookies. Los ejemplos de esta página ilustran esto mostrando como los servlets **CatalogServlet** y **ShowCart** trabajan con los mismos cookies.

Nota: Esta página tiene código que no forma parte del ejemplo **Duke's Bookstore**. **Duke's Bookstore** utilizaría código como el de esta página si utilizará cookies en vez de seguimiento de sesión para los pedidos de los clientes. Cómo las cookies no forman parte de **Duke's Bookstore**, piensa en los ejemplos de esta página como pseudo-código.

Para enviar una cookie:

1. Ejemplariza un objeto Cookie
2. Selecciona cualquier atributo.
3. Envía el cookie

Para obtener información de un cookie:

1. Recupera todos los cookies de la petición del usuario.
2. Busca el cookie o cookies con el nombre que te interesa, utiliza las técnicas de programación estándar.
3. Obtén los valores de las cookies que hayas encontrado.

11.16.1 Crear un Cookie

El constructor de la clase **javax.servlet.http.Cookie** crea un cookie con un nombre inicial y un valor. Se puede cambiar el valor posteriormente utilizando el método **setValue**.
El nombre del cookie debe ser un token HTTP/1.1. Los tokens son strings que contienen uno de los caracteres especiales listados en **RFC 2068**. (Strings alfanuméricos cualificados como tokens.) Además, los nombres que empiezan con el carácter dollar ("**$**") están reservados por **RFC 2109**.
El valor del cookie puede ser cualquier string, aunque no está garantizado que los valores null funcionen en todos los navegadores. Además, si enviamos una cookie que cumpla con las especificaciones

originales de las cookies de Netscape, no se deben utilizar caracteres blancos ni ninguno de estos caracteres.

[ ] ( ) = , "" / ? @ : ;

Si nuestro servlet devuelve una respuesta al usuario con un **Writer**, debemos crear la cookie antes de acceder a **Writer**. (Porque las cookies se envían al cliente como una cabecera, y las cabeceras deben escribirse antes de acceder al **Writer**.)
Si el **CatalogServlet** utilizará cookies para seguir la pista de una hoja de pedido, el servlet podría crear las cookies de esta forma.

```
   public void doGet (HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException
   {
      BookDBServlet database = (BookDBServlet)
getServletConfig().getServletContext().getServlet("bookdb");

      // Check for pending adds to the shopping cart
      String bookId = request.getParameter("Buy");

      //If the user wants to add a book, remember it by adding a cookie
      if (bookId != null) {
         Cookie getBook = new Cookie("Buy", bookId);
         ...
      }

      // set content-type header before accessing the Writer
      response.setContentType("text/html");

      // now get the writer and write the data of the response
            PrintWriter out = response.getWriter();
            out.println("<html> <head><title> Book Catalog </title></head>" + ...);
      ...
   }
```

11.16.2 Seleccionar los Atributos de un Cookie

La clase **Cookie** proporciona varios métodos para seleccionar los valores del cookie y sus atributos. La utilización correcta de estos métodos, están explicados en el javadoc para la clase **Cookie**.
El siguiente ejemplo selecciona el campo **comment** del cookie **CatalogServlet**. Este campo describe el propósito del cookie.

```
   public void doGet (HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException
   {
      ...
      //If the user wants to add a book, remember it by adding a cookie
      if (values != null) {
         bookId = values[0];
         Cookie getBook = new Cookie("Buy", bookId);
         getBook.setComment("User wants to buy this book from the bookstore.");
      }
      ...
   }
```

También se puede seleccionar la caducidad del cookie. Este atributo es útil, por ejemplo, para borrar un cookie. De nuevo, si **Duke's Bookstore** utilizará cookies para su hoja de pedidos, el ejemplo podría utilizar este atributo para borrar un libro de la hoja de pedido. El usuario borra un libro de la hoja de pedidos en el **ShowCartServlet**; su código se podría parecer a esto.

```
   public void doGet (HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException
```

```
{
        ...
    /* Handle any pending deletes from the shopping cart */
    String bookId = request.getParameter("Remove");
    ...
    if (bookId != null) {
       // Find the cookie that pertains to the book to remove
       ...
            // Delete the cookie by setting its maximum age to zero
            thisCookie.setMaxAge(0);
       ...
    }

        // also set content type header before accessing the Writer
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    //Print out the response
    out.println("<html> <head> <title>Your Shopping Cart</title>" + ...);
```

11.16.3 Enviar Cookies

Las cookies se envían como cabeceras en la respuesta al cliente, se añaden con el método **addCookie** de la clase **HttpServletResponse**. Si estamos utilizando un **Writer** para devolver texto, debemos llamar a **addCookie** antes de llamar al método **getWriter** de **HttpServletResponse**.
Continuando con el ejemplo de **CatalogServlet**, aquí está el código para enviar la cookie.

```
   public void doGet (HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException
   {
      ...
      //If the user wants to add a book, remember it by adding a cookie
      if (values != null) {
         bookId = values[0];
         Cookie getBook = new Cookie("Buy", bookId);
         getBook.setComment("User has indicated a desire to buy this book from the
bookstore.");
         response.addCookie(getBook);
      }
      ...
   }
```

11.16.4 Recuperar Cookies

Los clientes devuelven las cookies como campos añadidos a las cabeceras de petición HTTP. Para recuperar una cookie, debemos recuperar todas las cookies utilizando el método **getCookies** de la clase **HttpServletRequest**.
El método **getCookies** devuelve un array de objetos **Cookie**, en el que podemos buscar la cookie o cookies que querramos. (Recuerda que distintas cookies pueden tener el mismo nombre, para obtener el nombre de una cookie, utiliza su método **getName**.)
Para continuar con el ejemplo **ShowCartServlet**.

```
   public void doGet (HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException
   {
        ...
      /* Handle any pending deletes from the shopping cart */
      String bookId = request.getParameter("Remove");
      ...
      if (bookId != null) {
```

```
        // Find the cookie that pertains to the book to remove
        Cookie[] cookies = request.getCookies();
        ...
        // Delete the book's cookie by setting its maximum age to zero
        thisCookie.setMaxAge(0);
    }

          // also set content type header before accessing the Writer
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    //Print out the response
    out.println("<html> <head> <title>Your Shopping Cart</title>" + ...);
```

11.16.5 Obtener el valor de una Cookie

Para obtener el valor de una cookie, se utiliza el método **getValue**.

```
   public void doGet (HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException
   {
      ...
      /* Handle any pending deletes from the shopping cart */
      String bookId = request.getParameter("Remove");
      ...
      if (bookId != null) {
         // Find the cookie that pertains to that book
         Cookie[] cookies = request.getCookies();
         for(i=0; i < cookies.length; i++) {
            Cookie thisCookie = cookie[i];
            if (thisCookie.getName().equals("Buy") &&
                                          thisCookie.getValue().equals(bookId)) {

               // Delete the cookie by setting its maximum age to zero
               thisCookie.setMaxAge(0);
            }
         }
      }

          // also set content type header before accessing the Writer
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      //Print out the response
      out.println("<html> <head> <title>Your Shopping Cart</title>" + ...);
```

11.17 La Utilidad ServletRunner

Una vez que hemos escrito el servlet, podemos probarlo en la utilidad **servletrunner**
El **servletrunner** es un pequeño, proceso multi-thread que maneja peticiones de servlets. Como **servletrunner** es multi-thread, puede utilizarse para ejecutar varios servlets simultáneamente, o para probar un servlet que llama a otros servlets para satisfacer las peticiones de clientes.
Al contrario que algunos navegadores, **servletrunner** no recarga automáticamente los servlets actualizados. Sin embargo, podemos parar y reiniciar **servletrunner** con una pequeña sobrecarga de a pila para ejecutar una nueva versión de un servlet.

11.17.1 Seleccionar las Propiedades de un Servlet

Podríamos tener que especificar algunos datos para ejecutar un servlet. Por ejemplo, si un servlet necesita parámetros de inicialización, debemos configurar estos datos antes de arrancar **servletrunner**.

11.17.2 Arrancar servletrunner

Después de configurar el fichero de propiedades, podemos ejecutar la utilidad **servletrunner**. Esta página explica cómo.

11.18 Seleccionar Propiedades de un Servlet

Las propiedades son parejas de clave-valor utilizadas para la configuración, creación e inicialización de un servlet. Por ejemplo , **servlet.catalog.code=CatalogServlet** es una propiedad cuya clave es **servlet.catalog.code** y cuyo valor es **CatalogServlet**.
La utilidad **servletrunner** tiene dos propiedades para los servlets.

- **servlet.nombre.code**
- **servlet.nombre.initargs**

11.18.1 La propiedad code

El valor de la propiedad **servlet.nombre.code** es el nombre completo de la clase del servlet, incluido su paquete. Por ejemplo.

```
servlet.bookdb.code=database.BookDBServlet
```

La propiedad **servlet.nombre.code** llama a nuestro servlet asociando un nombre (en el ejemplo, **bookdb**) con una clase (en el ejemplo, **database.BookDBServlet**).

11.18.2 La propiedad initargs

El valor de la propiedad **servlet.nombre.initArgs** contiene los parámetros de inicialización del servlet. La sintaxis de este parámetro es **parameterName=parameterValue**. La propiedad completa (la pareja completa clave-valor) debe ser una sola línea lógica. Para mejorar la lectura, se puede utilizar la barra invertida para dividir la línea lógica en varias líneas de texto. Por ejemplo, si el servlet database leyera datos desde un fichero, el argumento inicial del servlet podría parecerse a esto.

```
servlet.bookdb.initArgs=\
    dbfile=servlets/DatabaseData
```

Los parámetros de inicialización múltiples se especifican separados por comas. Por ejemplo, si el servlet database se conectará a una base de datos real, sus argumentos iniciales podrían parecerse a esto.

```
servlet.bookdb.initArgs=\
        user=duke,\
        password=dukes_password,\
        url=fill_in_the_database_url
```

11.18.3 El fichero de Propiedades

Las propiedades se almacenan en un fichero de texto con un nombre por defecto de **servlet.properties**. (Se puede especificar otro nombre cuando se arranca **servletrunner**.) El fichero guarda las propiedades para todos los servlets que se ejecuten en el **servletrunner**. Aquí puedes ver el fichero de propiedades para el ejemplo Duke's Bookstore.

```
# This file contains the properties for the Duke's Bookstore servlets.

# Duke's Book Store -- main page
servlet.bookstore.code=BookStoreServlet
```

```
# The servlet that manages the database of books
servlet.bookdb.code=database.BookDBServlet

# View all the books in the bookstore
servlet.catalog.code=CatalogServlet

# Show information about a specific book
servlet.bookdetails.code=BookDetailServlet

# See the books that you've chosen to buy
servlet.showcart.code=ShowCartServlet

# Collects information for buying the chosen books
servlet.cashier.code=CashierServlet

# Provide a receipt to the user who's bought books
servlet.receipt.code=ReceiptServlet
```

11.19 Arrancar ServletRunner

El **servletrunner** está en el directorio **<jsdk>/bin**. Se podrá ejecutar más fácilmente si lo ponemos en el path. Por ejemplo.

```
% setenv PATH /usr/local/jsdk/bin:                    (para UNIX)

C> set PATH=C:\jsdk\bin;%PATH%                        (para Win32)
```

Llamar a **servletrunner** con la opción **-help** muestra una ayuda sin ejecutarlo.

```
% servletrunner -help
Usage: servletrunner [options]
Options.
  -p port     the port number to listen on
  -b backlog  the listen backlog
  -m max      maximum number of connection handlers
  -t timeout  connection timeout in milliseconds
  -d dir      servlet directory
  -r root     document root directory
  -s filename servlet property file name
  -v          verbose output
%
```

Para ver los valores por defecto de estas opciones, podemos llamar a **servletrunner** con la opción **-v**. Esto arranca la utilidad, se debe parar inmediatamente si una vez obtenida la información no estamos listos para ejecutarlo. o si queremos ejecutar algo distinto de los valores por defecto. Por ejemplo, en Unix, utilizando el comando **kill** para parar **servletrunner**.

```
% servletrunner -v
Server settings.
  port = 8080
  backlog = 50
  max handlers = 100
  timeout = 5000
  servlet dir = ./examples
  document dir = ./examples
  servlet propfile = ./examples/servlet.properties
```

Nota: En los valores por defecto mostrados arriba. **servlet dir**, **document dir** y el directorio **servlet propfile** contienen un punto ("**.**"). El punto designa el directorio de trabajo actual.

Normalmente este directorio es desde donde se arranca el ejecutable. Sin embargo, en este caso, el punto se refiere al directorio donde está instalado el "servlet development kit".
Si arrancamos **servletrunner** desde un directorio distinto al de instalación, **servletrunner** primero cambia su directorio de trabajo (y, por lo tanto, lo que podrías pensar como el valor de "**.**").

Una vez que **servletrunner** está en ejecución, podemos utilizarlo para probar nuestros servlets.

11.20 Ejecutar Servlets

Esta lección muestra unas cuantas formas de llamar a los servlets.

11.20.1 Tecleando la URL del servlet en un Navegador Web

Los servlets pueden ser llamados directamente tecleando su URL en un navegador Web. Así es como se accede a la página principal del ejemplo **Duke's Bookstore**. Esta página muestra la forma general de la URL de un servlet.

11.20.2 Llamar a un Servlet desde dentro de una página HTML

Las URLs de los servlets pueden utilizarse en etiquetas HTML, donde se podría encontrar una URL de un script CGI-bin o una URL de fichero. Esta página muestra como utilizar la URL de un servlet como destino de un enlace, como la acción de un formulario, y como la localización a utilizar cuando META tag dice que la página sea refrescada. Esta sección asume conocimientos de HTML.

11.20.3 Desde otro servlet

Los Servlets pueden llamar a otros servlets. Si los dos servlets están en distinto servidor, uno puede hacer peticiones HTTP al otro. Si los dos se ejecutan en el mismo servidor, entonces un servlet puede llamar a los métodos públicos del otro directamente.
Estas páginas asumen que.

- Nuestra máquina, **localhost**, está ejecutando servletrunner o un servidor con soporte para servlets, como Java Web Server en el puerto 8080.
- El ejemplo, **Duke's Bookstore**, está localizado en el nivel superior del directorio de procesos para los servlets. Para **servletrunner**, esto significa que los ficheros class están en el directorio servlet especificado por la **opción -d.**

Si estas dos condiciones se cumplen, podremos ejecutar el servlet de ejemplo tecleando las URLs dadas en el ejemplo.

11.21 Llamar a Servlets desde un Navegador

La URL de un servlet tiene la siguiente forma general, donde nombre-servlet corresponde al nombre que le hemos dado a nuestro servlet.

```
http://nombre-de-máquina:puerto/servlet/nombre-servlet
```

Por ejemplo, el servlet que lanza la página principal de **Duke's Bookstore** tiene la propiedad **servlet.bookstore.code=BookStoreServlet**. Para ver la página principal, teclearemos esta URL en nuestro navegador.

```
http://localhost:8080/servlet/bookstore
```

Las URLs de servlets pueden contener preguntas, como las peticiones GET de HTTP. Por ejemplo, el servlet que sirve los detalles sobre un libro particular toma el número de inventario del libro como

pregunta. El nombre del servlet es **bookdetails**; la URL del servlet para obtener (GET) y mostrar toda la información sobre las características de un libro.

```
http://localhost:8080/servlet/bookdetails?bookId=203
```

11.22 Llamar a Servlets desde una Página HTML

Para invocar un servlet desde dentro de una página HTML se utiliza la URL del servlet en la etiqueta HTML apropiada.
Esta página utiliza los servles **ShowCart**, **Cashier**, y **Receipt** de **Duke's Bookstore**. Afortunadamente este es el orden en que se verán los servlets cuando miremos nuestra hoja y compremos nuestros libros. Para un acceso más directo al servlet ShowCart servlet, pulsa el enlace **Show Cart** que hay en la página principal del **Duke's Bookstore**. Si tenemos **servletrunner** o un servidor web configurados para ejecutar el ejemplo, vayamos a la página principal de la librería mostrada en la página anterior. Sólo por diversión, podríamos añadir un libro a nuestra hoja de pedido antes de acceder al servlet **ShowCart**.

11.22.1 Ejemplos de URLs de Servlets en etiquetas HTML

La página devuelta por ShowCartServlet tiene varios enlaces, cada uno de los cuales tiene un servlet como destino. Aquí podemos ver el código de esos enlaces.

```
public class ShowCartServlet extends HttpServlet {

    public void doGet (HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException
    {
        ...
        out.println(... + "<a href=\"" + response.encodeUrl("/servlet/cashier") +
                "\">Check Out</a>      " + ...);
        ...
    }
    ...
}
```

Este código resulta en una página HTML que tiene el siguiente enlace.

```
<a href="http://localhost:8080/servlet/cashier>Check Out</a">
```

Si llamamos a la página del showcart, podremos ver el enlace como si viéramos el fuente de la página. Luego pulsamos sobre el enlace. El servlet cashier devolverá la página que contiene el siguiente ejemplo. La página mostrada por el server cashier presenta un formulario que pide el nombre del usuario y el número de la tarjeta de crédito. El código que imprime el formulario se parece a esto.

```
public class CashierServlet extends HttpServlet {

    public void doGet (HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException
    {
        ...
        out.println(... +
                "<form action=\"" + response.encodeUrl("/servlet/receipt") + "\"
method=\"post\">" +  ...   "<td><input type=\"text\" name=\"cardname\"" +
"value=\"Gwen Canigetit\" size=\"19\"></td>" +  ...  "<td><input type=\"submit\"" +
"value=\"Submit Information\"></td>" +  ... "</form>" +  ...);
        out.close();
    }
    ...
}
```

Este código resulta en una página HTML que tiene la siguiente etiqueta para iniciar el formulario.

```
<form action="http://localhost:8080/servlet/receipt" method="post">
```

Si cargamos la página del servlet cashier en nuestro navegador podremos ver la etiqueta que inicia el formulario como si viéramos el fuente de la página. Luego enviamos el formulario. El servlet receipt devolverá una página que contiene el siguiente ejemplo. La página del servlet receipt se resetea a sí misma, por eso si queremos verla, tenemos que hacerlo **rápido!**.
La página devuelta por el servlet receipt tiene una "meta tag" que utiliza una URL de servlet como parte del valor del atributo **http-equiv**. Específicamente, la etiqueta redirecciona la página hacia a la página principal del **Duke's Bookstore** después de dar las gracias al usuario por su pedido. Aquí podemos ver el código de esta etiqueta.

```
public class ReceiptServlet extends HttpServlet {

    public void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException
    {
        ...
        out.println("<html>" +
                "<head><title> Receipt </title>" +
                "<meta http-equiv=\"refresh\" content=\"4; url=" +
                "http://" + request.getHeader("Host") +
                "/servlet/bookstore;\">" +
                "</head>" +
        ...
    }
    ...
}
```

Este código resulta en una página HTML que tiene la siguiente etiqueta.

```
<meta http-equiv="refresh"
    content="4; url=http://localhost:8080/servlet/bookstore;">
```

11.23 Llamar a un Servlet desde otro Servlet

Para hacer que nuestro servlet llame a otro servlet, podemos.

- Un servlet puede hacer peticiones HTTP a otro servlet.
- Un servlet puede llamar directamente a los métodos públicos de otros servlet, si los dos se están ejecutando dentro del mismo servidor.

Esta página explica la segunda opción. Para llamar directamente a los métodos públicos de otro servlet, debemos.

- Conocer el nombre del servlet al que queremos llamar.
- Obtener el acceso al objeto **Servlet** del servlet.
- Llamar al método público del servlet.

Para obtener el acceso al objeto**Servlet**, utilizamos el método **getServlet** del objeto **ServletContext**. Obtener el objeto **ServletContext** desde el objeto **ServletConfig** almacenado en el objeto **Servlet**. Un ejemplo aclarará esto. Cuando el servlet **BookDetail** llama al servlet **BookDB**, el servlet **BookDetail** obtiene el objeto **Servlet** del **BookDB Servlet** de esta forma.

```
public class BookDetailServlet extends HttpServlet {

    public void doGet (HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException
    {
```

```
        ...
            BookDBServlet database = (BookDBServlet)
                getServletConfig().getServletContext().getServlet("bookdb");
        ...
    }
}
```

Una vez que tenemos el objeto Servlet, podemos llamar a cualquiera de los métodos públicos del servlet. Por ejemplo, el servlet **BookDetail** llama al método **getBookDetails** del servlet **BookDB**.

```
public class BookDetailServlet extends HttpServlet {

    public void doGet (HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException
    {
        ...
            BookDBServlet database = (BookDBServlet)
                getServletConfig().getServletContext().getServlet("bookdb");
            BookDetails bd = database.getBookDetails(bookId);
        ...
    }
}
```

Debemos tener precaución cuando llamemos a métodos de otro servlet. Si el servlet al que queremos llamar implementa la interface **SingleThreadModel**, nuestra llamada podría violar la naturaleza mono-thread del servlet. (El servidor no tiene forma de intervenir y asegurarse de que nuestra llamada suceda cuando el servlet no está interactuando con otro cliente). En este caso, nuestro servlet debería hacer una petición HTTP al otro servlet en vez de llamar directamente a sus métodos.