Capítulo

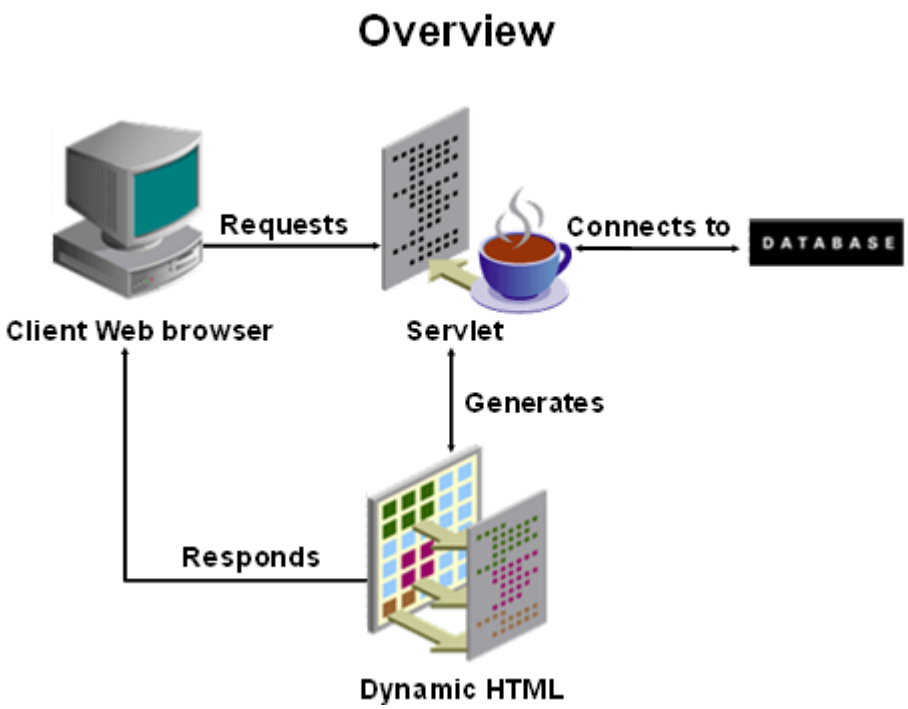
6

Servlets

En esta sección se va a tratar el primer componente que corre del lado del servidor, y es uno de los puntos fundamentales para la comunicación entre el cliente (browser) y el servidor.

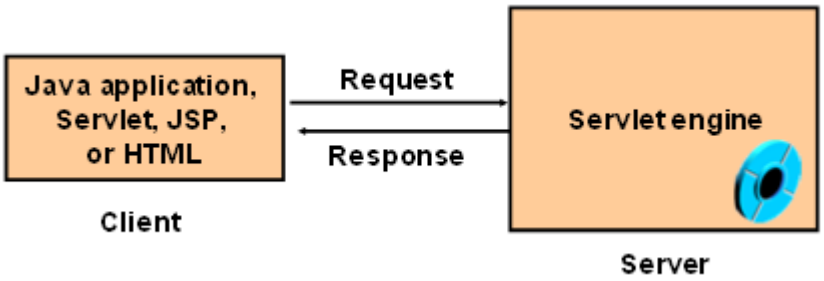
Generalidades

Un Web browser invoca una url, la cual llama a un servlet. El servlet entonces genera contenido html dinámico, Extensible Markup Lenguaje(XML) u otro contenido. El texto dinámico puede haber sido generado por una conexión con la base de datos.



Un servlet es una clase java que implementa la Servlet interface. Un servlet corre dentro de un contexto denominado Servlet engine.

Los servlet pueden ser invocados simultáneamente por múltiples clientes.



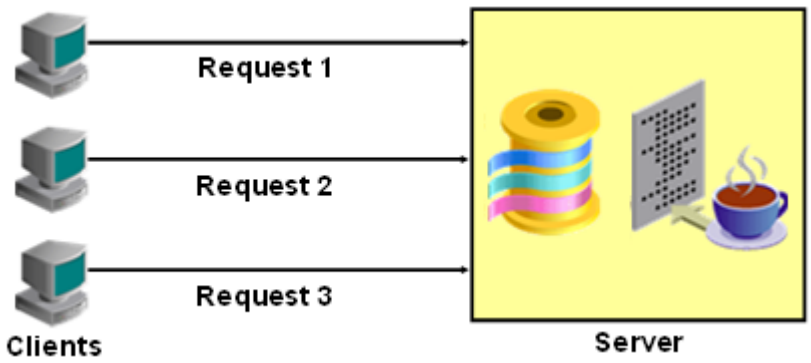
Un servlet es un objeto java que implementa la servlet API. Estas clases están en el javax.servlet package.

Cuando ud. usa el servlet API, tiene en cuenta algunos de las tareas comunes que son llevadas a cabo como respuesta a un requerimiento desde el cliente. La API soporta preinstanciado de objetos en la JVM si múltiples clientes deben acceder simultáneamente a una funcionalidad particular que ud. Provee con una clase Java. Esta característica se conoce como ciclo de vida del servlet.

Así como los applets corren en el contexto de un web browser, un servlet corre en el contexto de una servlet engine. El Web browser y el servlet engine contienen una JVM que es funcional todo el tiempo. A diferencia de los applets un servlet no usa ninguna característica del lenguaje en cuanto a una user interface, como el paquete java swing. Como se muestra en al figura anterior, muchos clientes puede invocar un servlet y pueden usar distintos protocolos. Por ej. Es posible escribir un servlet que implemente el FTP Server protocol

Caracteristicas de los Servlets

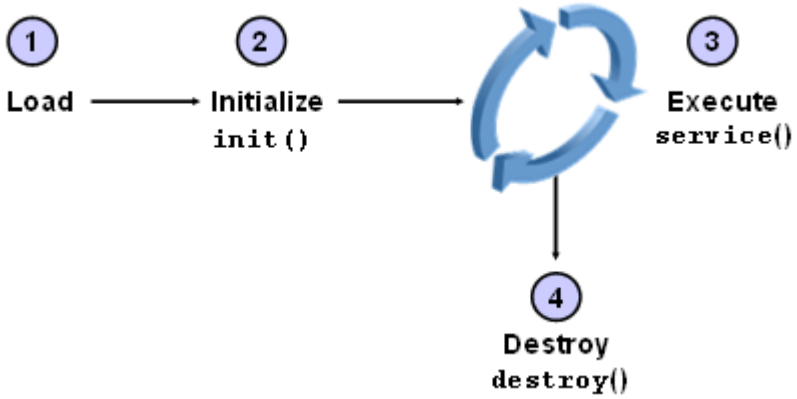
- Múltiples requerimientos son posibles generalmente
- Los métodos de los servlet corren en Threads
- Las instancias de los servlets son compartidas por múltiples clientes



Como muestra la figura anterior, 2 o más requerimientos para el mismo servlet corren en múltiples threads o hilas de ejecución.

Cuando no hay más requerimientos para un servlet, la instancia no es extinguida queda a la espera de futuros requerimientos del usuario. La memoria u otros recursos son reusados cuando es necesario.

Ciclo de vida de un servlet



El ciclo de vida de un servlet tiene las siguientes características:

Muchos servlet engines se ejecutan dentro de una JVM

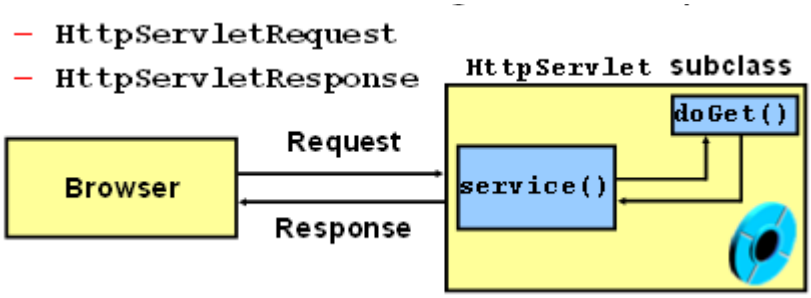
- Los servlets persisten entre requerimiento como instancias de objetos. Si un objeto instancia hace una conexión a la base de datos, esta persiste y no hay necesidad de realizar otra conexión para un segundo requerimiento.

- Se puede sobrescribir el método init(), el cual es invocad por el servlet engine antes de atender los requerimientos del los clientes y el método destroy() el cual es invocado cuando el servlet engine remueve un servlet

- Un servlet puede también ser explícitamente escrito como single-threaded model. En este caso si 2 requerimientos son recibidos al mismo tiempo, el servlet engine usa otra instancia de la clase. Es decir 2 objetos atienden dichos requerimientos.

HTTP Servlets

Los HTTP servlets extienden de la clase javax.servlet.http.HttpServlet, la cual extinde de la clase GenericServlet. La clase GenericServlet implementa la Servlet interface. Un cliente hace un requerimiento http, el tipo de método puede ser de tipo GET o POST. Esto determina el tipo de acción que el servlet llevara a cabo.



Cuando un servlet es invocado, el servlet engine llamar al método service() del servlet. Este método ha sido declarado por la Servlet interface. Si el browser ha invocado el método GET en el http protocol, entonces el método service() invoca al método doGet() del objeto. De la misma manera si el browser invoca el método POST en el http protocol (a través de un html form), el método service() invoca el método doPost(). A veces la funcionalidad es la misma. Del método doPost() se invoca al método Donet() y viceversa. Otros método están disponibles en los servlets como doPut() y doDelete() para operaciones FTP. Ambos métodos toman como parámetros de entrada 2 argumentos:

HttpServletRequest y HttpServletResponse

Ejemplo de un servlet

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class SimplestServlet extends
HttpServlet
{
  public void doGet(HttpServletRequest
request, HttpServletResponse response) throws
ServletException, IOException
  {
    PrintWriter out = response.getWriter();
    out.println("Hello World");
  }
}
```

Métodos doGet() y doPost

EL METODO DOGET()

El método doGet() recibe 2 parámetros de entrada. La figura anterior muestra un ejemplo de este método. Los parámetros del requerimiento son pasados al método doGet() a través de la URL de la siguiente forma:

http://www.acme.com/servlet?param1=value1.

Múltiples parámetros pueden ser enviados de la siguiente manera http://www.oracle.com/servlet?param1=value1&param2=value2.

Además la clase java.net.URLEncoder codifica los parámetros que son pasados en al URL. El método encode(String s, String enc) convierte un String usando el caracter de codificación enc y retorna el String convertido. Es de ayuda para pasar parámetros con caracteres especiales.

El requerimiento GET es generado por el browser en las siguientes instancias:

- URL en la address line del browser.
- HREF link
- Submit de un form cuyo metodo es GET: <FORM METHOD="GET">.

EL METODO DOPOST()

El método doPost() es usado en conjunción con un HTML form. Cuando el usuario Hace clips en el botón de “submit” del form, cualquier parámetro incluido dentro del form es pasado al servlet que es invocado en el tag action. Igual que el doGet() el metod doPost() recibe 2 parámetros como entrada: HttpServletRequest y HttpServletResponse. Un par nombre-valor son pasados al web Server como un requerimiento adiciones en el header y no agregándolos a la URL.

Las ventajas del métodos Post son:

- Parámetros (como password) no son visibles en el URL del browser.
- No se puede hacer un bookmark de la URL conteniendo los valores de los parámetros.
- Los web Server usualmente limitan la cantidad de caracteres pasados en la URL, no hay limite teórico para los parámetros tipo POST.

Objetos httpServletRequest y httpServletresponse

EL OBJETO HTTPSERVLET REQUEST

Los métodos doGet() y doPost() toman el objeto HttpServletRequest como parámetro. Este objeto encapsula la siguiente información acerca del cliente:

- Nombre y valor de los parámetros del Servlet
- El nombre del host remoto que ha realizado el requerimiento
- El nombre del Server que recibe el requerimiento
- Imput stream data

La siguiente tabla describe varios métodos en este objeto y su propósito, basándose en de ejemplo en la siguiente url:

http://bighost:80/finance/servlet/Ledger/June?region=east

Request methods	Return values
`getServerName()`	`bighost`
`getServerPort()`	`80`
`getPathInfo()`	`/finance/servlet/Ledger/June.class`
`getServletPath()`	`/servlet/Ledger.June`
`getContextPath()`	`/finance`
`getRequestURI()`	`/finance/servlet/Ledger.June`

EL OBJETO HTTPSERVLET RESPONSE

El Segundo parámetro de los métodos doGet() y doPost() es el objeto HttpServletResponse. Encapsula información que el servlet ha generado:

- La longitud del contenido de la respuesta
- El MIME Type de la respuesta
- El outputStream

El primer punto consiste del status line (si en servlet fue invocado satisfactoriamente), uno mas headers y el output o salida del servlet.

La siguiente tabla describe varios métodos que están disponibles en el objeto HttpServletResponse y sus propósitos:

`setContentType()`	Sets the `Content-Type` header for the document. Must be used for servlets that return document content. Common settings include `text/html`, `application/pdf`, and `image/gif`.
`setContentLength()`	An optional method that sets the `Content-Length` header, useful for persistent HTTP connections
`sendRedirect()`	Sets the `Location` header and sets the status code to 302. Status codes are discussed in detail later in the course.

Métodos para invocar un Servlet

Ud. Puede directamente tipear la URL en un browser.

Cualquier pagina puede contener una servlet URL usando los tag

- **<a href=> o <FORM action=>**
- Una JSP puede invocar un servlet.
- Un servlet puede ser invocado por otro servlet a través de cadenas de servlets (Chain)

Un Primer Servlet

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloWorld extends HttpServlet {
  public void doGet(
    HttpServletRequest  request,
    HttpServletResponse response)
  throws ServletException, IOException{
    response.setContentType ("text/html");
    PrintWriter out = response.getWriter();
    out.println ("<html>");
    out.println ("<body>");
    out.println ("Hello World!");
    out.println ("</body></html>");
  }
}
```

El ejemplo en la figura anterior muestra un típico método doGet(). El código realiza las siguientes funciones:

- Setea el MIME type de la salida a text/html.
- Obtiene la referencia del objeto PrintWriter
- Escribe código HTML para el objeto.

El resultado muestra en una pagina HTML “Hello World!”.

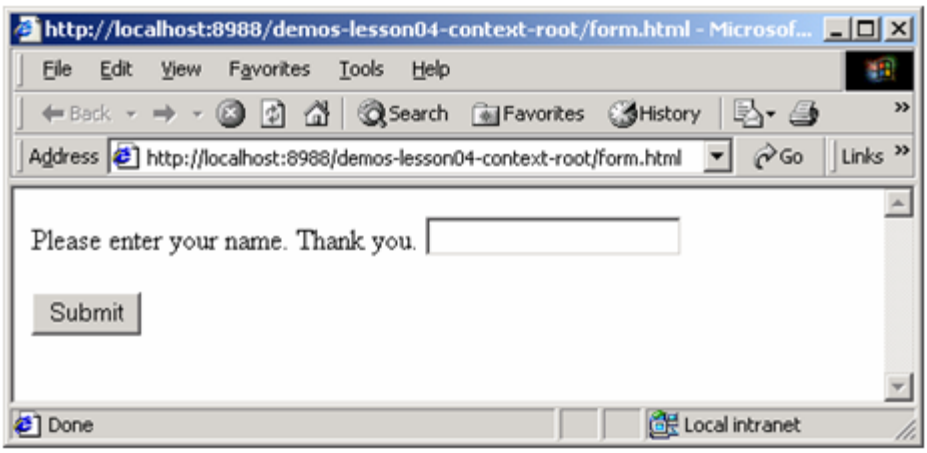
La figura también muestra los objetos HttpServletRequest y HttpServletResponse que son importados desde el paquete javax.servlet.http.

HTML Form

Se puede utilizar un formulario HTML para invocar el método POST de un servlet. El cual es la entrada de datos a nuestro servlet. Ud. Debe procesar el formulario.

```
<html><body>
<form method="post" action="newhelloworld">
Please enter your name. Thank you.
<input type="text" name="firstName"> <P>
<input type="submit" value="Submit">
</form>
</body>
</html>
```

El formato de salida del código anterior es similar a este:



Administrando la entrada de datos desde el servlet

```
public class NewHelloWorld extends HttpServlet {
  public void doPost(
  HttpServletRequest  req, HttpServletResponse res)
  throws ServletException, IOException{
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    out.println ("<html><body>");
    String name = req.getParameter("firstName");
    if ((name != null) && (name.length() > 0))
      out.println ("Hello: " + name +
      "  How are you?");
    else
      out.println ("Hello Anonymous!");
    out.println ("</body></html>");
  }}
```

La figura anterior muestra como se puede procesar una entrada de datos. El contentType setea a “text/html”. El método getParameter() es usado para recibir los datos de manera case-sensitive, tal es el caso del parámetro firstName que fue enviado como un String desde el HTML Form. Si el valor no es encontrado un String vacío es retornado y un null si no hay parámetros con el nombre especificado. Ademas getParameterValues puede ser usado para retornar una array de Strings si el parámetro puede tener mas de un valor.

Inicialización y destrucción de Servlets

Además del método service() los servlets poseen el método init() y destroy().

El método init():

- Puede ser usado para obtener parámetros de inicialización.
- Recibe un objeto ServletConfig.
- Es invocado cuando se crea instancia del servlet.
- Puede ser usado para obtener conexiones a la base de datos.

El método destroy()

- No toma argumentos
- Es invocado cuando el servlet es descargado
- Es útil para liberar recursos

El objeto ServletConfig del método init() puede ser usado para encontrar parámetros de inicialización que son asociados con el servlet. Se puede utilizar el método getInitParameter() para extraer los nombres de los parámetros de inicializacion. Nota ud. Debe llamar a super.init() en el metodo init() si ud. Usa el ServletConfig object,porque el el servletConfig object puede estar en cualquier lugar en el servlet.

public void init(ServletConfig config) throws ServletException

{ super.init(config);

... // your initialization here }

En una aplicación Web J2EE , la inicializacion de parámetros es definida en el archivo web.xml:

```
<init-param>
  <param-name>message</param-name>
  <param-value>Hello From Init Parameter</param-value>
</init-param>
```

Manejo de Errores

Una SerlvetException es lanzada cuando:

- Es generada para indicar un problema genérico en el servlet
- Tiene una subclase llamada UnavaibleException para indicar el servlet no esta disponible temporariamente o definitivamente.
- Es administrado por el Servlet Engine de diferentes maneras.

Una IOException es generada si hay un error de entreda/salida de datos en el procesamiento del request.

El metodo doXxxxx() lanza ambas excepciones ServletException y IOException. El metodo init() lanza la ServletEception. Estas excepciones tienen el mismo constructor que la clase Exception. Generalmente, una IOException es retronada solo cuando alguna clase de una operación stream sobre el servlet no puede ser llevada a cabo. ServletException es retornada si el servlet no puede ser ejecutado correctamente. Para excepciones adicionales, una página de error puede ser seteada para código de errores en el archivo web.xml:

<error-page>

<exception-type> java.lang.NumberFormatException </exception-type>

<location>MyExceptionServlet</location>

</error-page>

SingleThreadModel

```
public class HelloWorld extends HttpServlet
implements SingleThreadModel{
  public void doGet...
}
```

Se utiliza para prevenir que múltiples Threads accedan a datos simultáneamente. Cada concurrencia de un request tiene su propia instancia u objeto servlet dedicada, la cual es aleatoriamente asignada.

Normalmente, el servlet engine crea un nuevo thread para cada requerimiento. Si el servlet implemente la interface SingleThreadModel, entonces siempre un requerimiento será atendido por una simple instancia de un servlet. En este caso el desarrollador no necesita sincronizar el acceso a datos compartidos, como las variables de instancia. El Single la interfaz SingleTheradModel es una solución para bajos volúmenes de información que administra datos no sincronizados y no puede ser accedido simultáneamente por distintos requerimientos, como por ej. crear un userID.

La siguiente tabla muestra las variables que pueden ser thread safe:

Variable/Attribute	Thread Safe?
Local variables	Thread safe
Instance variables	Not thread safe, because a single servlet instance may be servicing multiple requests at any given time
Class variables	Not thread safe, because multiple servlets and requests may try to access a class variable concurrently
Request attributes	Thread safe, because the request object is a local variable
Session attributes	Not thread safe, because sessions are scoped at web app level
Context attributes	Not thread safe, because the same context object can be accessed concurrently

Servlet Maping

El cliente accede al servlet de la siguiente manera:

Tipeando directamente la URL

http://host:port/<context-root>/servlet/<package>.<servlet>

o usando una url mapeada

http://host:port/<context-root>/servlet/<mappedservletname>

Donde <context-root> es el nombre mapeado para el modulo web. Es creado por Jdevelper como <Workspacename>-<ProjectName>-context-root y puede ser cambiando en el archive de configuracion http-web-site.xml configuration file.

Host es en nombre o el dirección IP del Server.

En JDeveloper provee una forma estándar de mapeo mediante el archvio web.xml

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web
Application 2.2//EN" "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
   <servlet>
      <servlet-name>MyFirstServlet</servlet-name>
      <servlet-class>package1.HelloWorld</servlet-class>
   </servlet>
   <servlet-mapping>
      <servlet-name>MyFirstServlet</servlet-name>
      <url-pattern>/helloworld</url-pattern>
   </servlet-mapping>
   ...
</web-app>
```

El archivo web.xml es usado principalmente para mapear un servlet a un directorio virtual y definirlo. Es un archivo estándar de J2EE es creado automáticamente cuando es un servlet es creado y los apropiados tags XML son incluidos en el archivo. Para modificarlo se puede hacer botón-derecho del mouse en el archivo web.xml y elegir Properties desde el context menú.

En el ejemplo el servlet llamado HelloWorld.class es mapeado a un directorio viertual llamado /helloworld. Allí el servlet puede ser accedido por las siguientes URL:

http://localhost:8989/<context-root>/servlet/package1.HelloWorld

http://localhost:8989/<context-root>/helloworld

El directorio virtual /servlet viene con la configuración de J2EE Web Server. Es incluido automáticamente cuando se corre un servlet que está mapeado en el web.xml.