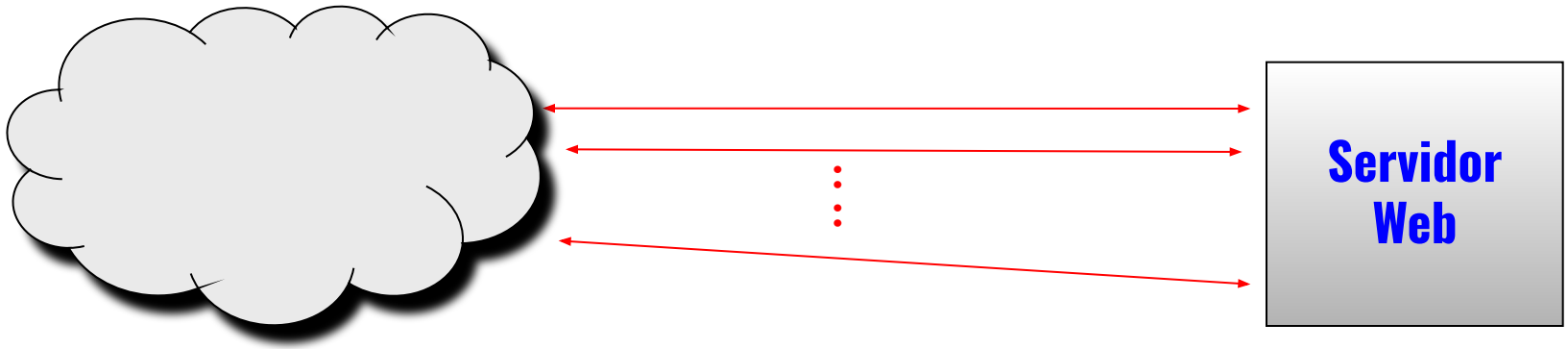
Soporte de Sesiones HTTP en Servlets

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-Compartir Igual 4.0 Internacional.

Soporte de Sesiones

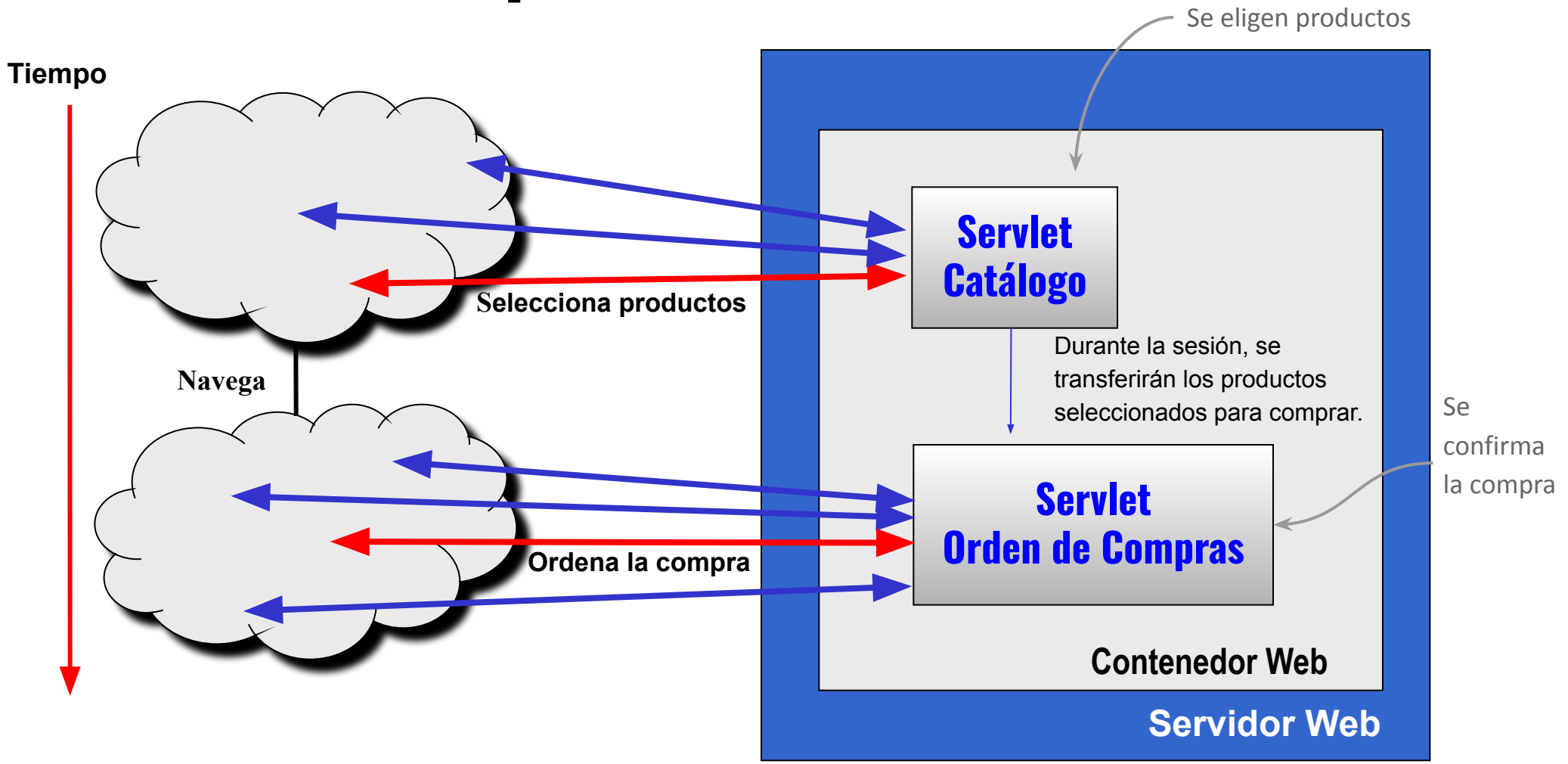
- **HTTP (Hypertext Transfer Protocol) es un protocolo sin estado, orientado a conexión, usado para la comunicación en la web.** Las peticiones y respuestas HTTP son independientes unas de otras. El servidor web desconoce si una serie de peticiones provienen del mismo o de diferentes clientes y si además están relacionadas entre sí.
- Una **sesión** entre un **cliente** y un **servidor** consiste en una **serie de interacciones relacionadas** entre un **mismo navegador y servidor web** durante un período de tiempo dado. Una **sesión mantiene información o un estado** asociado a un **cliente** particular.



- Las **aplicaciones web son las responsables de mantener las sesiones** dado que HTTP es un protocolo sin estado. La clave consiste en **identificar** todas las **peticiones provenientes de un mismo cliente** y mantener un estado entre dichas peticiones.

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-Compartir Igual 4.0 Internacional.

Soporte de Sesiones



¿Cómo hacemos desde una aplicación web para identificar las peticiones que provienen de un mismo cliente (navegador)?, ¿Cómo mantenemos el estado entre las diferentes peticiones?

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-Compartir Igual 4.0 Internacional.

Soporte de Sesiones en la API Servlet

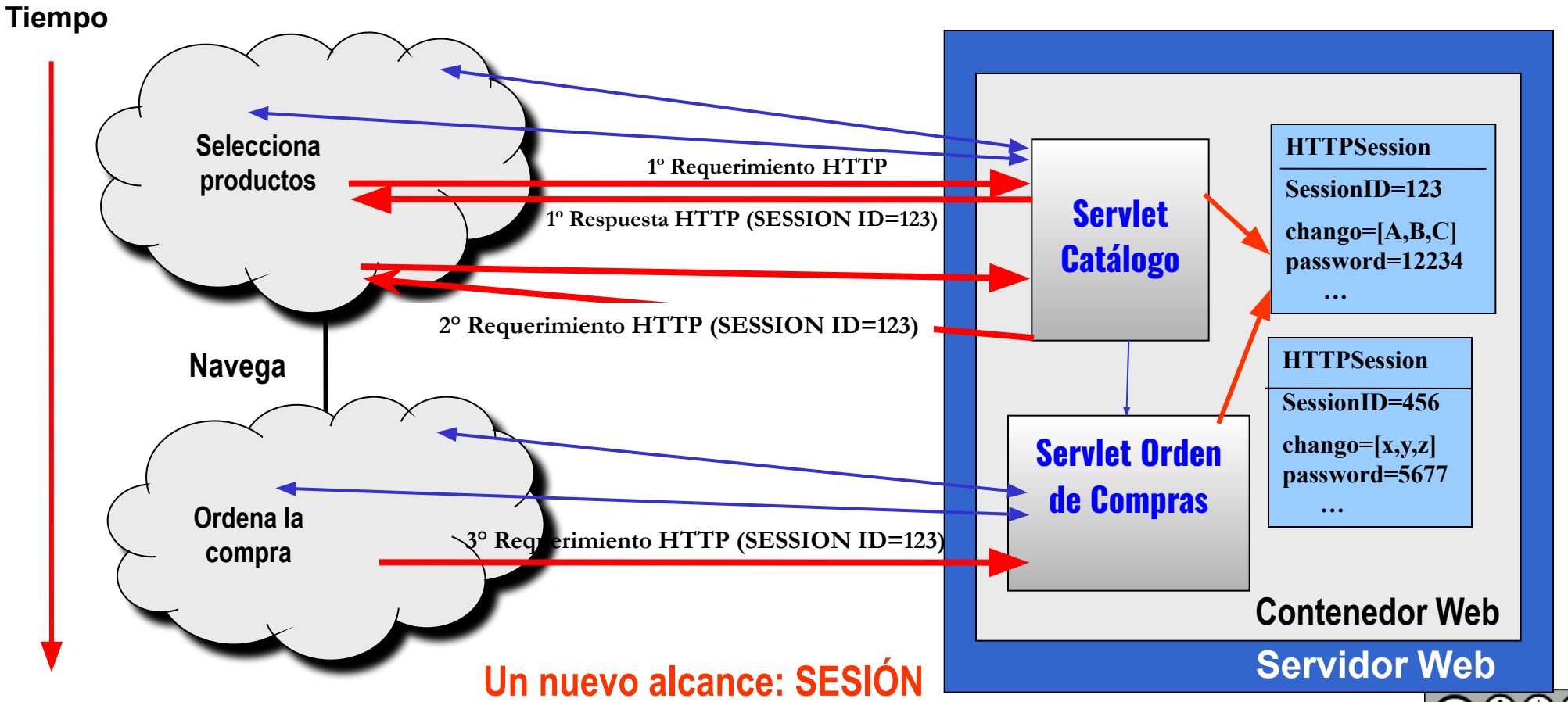
- Una de las funciones más importantes de la **API de Servlet** es el **gerenciamiento de sesiones: creación**, **finalización, invalidación** y **mantenimiento del estado de la sesión** del usuario, a través de la interface **jakarta.servlet.http.HttpSession**.
- Las **sesiones** están representadas por objetos **HttpSession**. Son **objetos server-side** que **mantienen el estado del cliente** y **almacenan atributos** que son propios del cliente que está interactuando con la aplicación.
- **HttpSession** es una interface JAVA que utiliza **Cookies** o **reescritura de URL** para **intercambiar la identificación del ID de usuario entre el servidor y el cliente**. En forma predeterminada, los contenedores usan cookies si el navegador lo soporta y automáticamente revierten a reescritura de URL cuando las cookies están deshabilitadas.
- El programador de servlets tiene garantizado un **espacio en la memoria del contenedor web** dónde se **guardan los datos asociados con cada sesión** de usuario. No necesita preocuparse por los detalles de la sesión, no tiene que manipular explícitamente cookies ni agregar información a la URL.
- **La especificación de servlets obliga a los contenedores web JAVA a implementar la interface HttpSession**. Los contenedores usan los objetos **HttpSession** para **crear una sesión HTTP** entre un cliente HTTP y un servidor HTTP.

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-Compartir Igual 4.0 Internacional.

Soporte de Sesiones

El objeto HTTPSession - Flujo de interacción

- El objeto **HttpSession** tiene asociado una identificación única, **SESSION ID**, que se entrega al cliente mediante cookies o reescritura de URLs. El **contenedor web JAVA crea** el objeto **HttpSession**
- Cada vez que el cliente interactúa con una componente de la aplicación web, le provee a la componente (servlet) el **SESSION ID** previamente asignado.

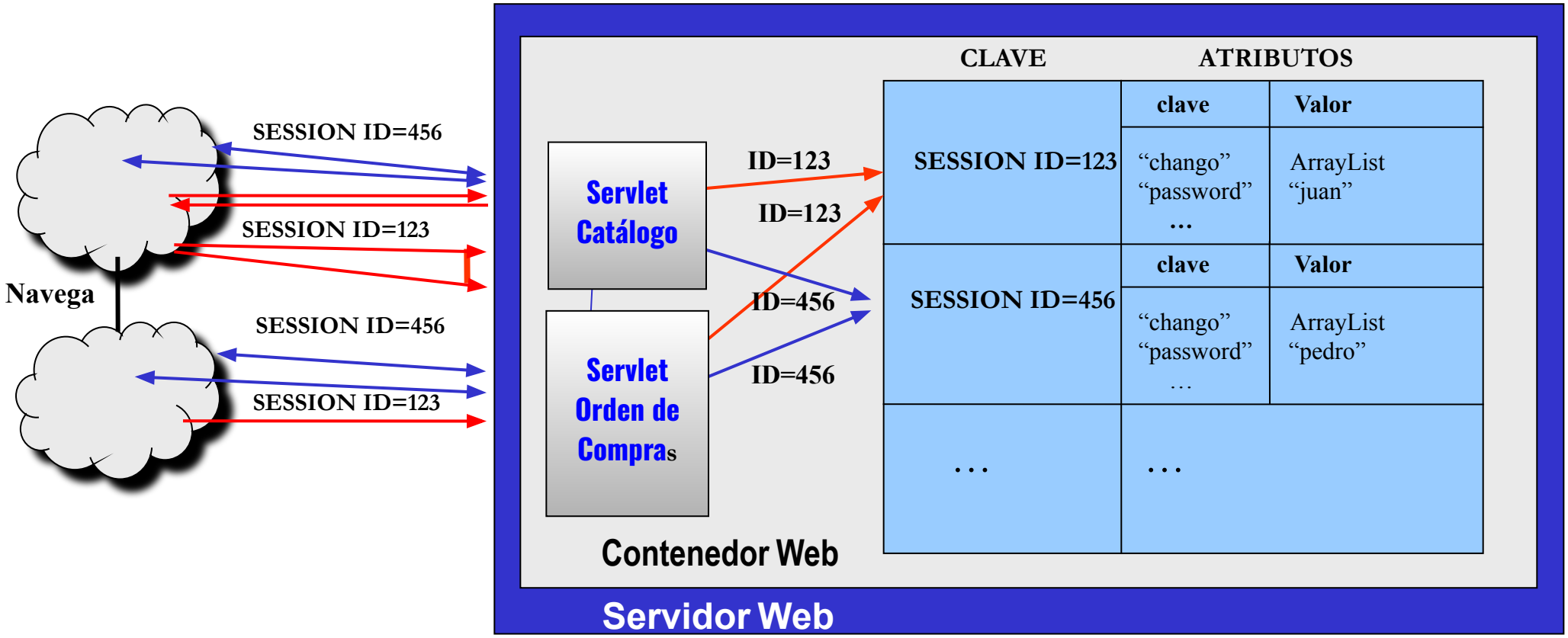


Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-Compartir Igual 4.0 Internacional.

Soporte de Sesiones

El objeto HTTPSession - Flujo de interacción

Un objeto **HttpSession** es un **contenedor de datos que reside en el contenedor web**. Los datos que mantiene son **privados de cada cliente particular** y permanecen **disponibles en la memoria del servidor** mientras la **sesión esté activa**. El objeto **HttpSession** es usado por las componentes web para **guardar** y **recuperar objetos JAVA**. Podría pensarse como una gran tabla de hash con **clave=SESSION ID** y cuyos valores son tablas de hash con los datos particulares cada cliente.



Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-Compartir Igual 4.0 Internacional.

Soporte de Sesiones

¿Cómo crear y recuperar una sesión?

Las **sesiones están asociadas a las peticiones HTTP** es decir al objeto **HttpServletRequest**.

La interface **jakarta.servlet.http.HttpServletRequest** provee **métodos** para **crear sesiones** nuevas y **obtener sesiones existentes**:

Métodos de HttpServletRequest

HttpSession getSession()	Devuelve la sesión asociada al requerimiento actual o crea una nueva si el requerimiento no tiene sesión asociada.
HttpSession getSession(boolean crear)	Devuelve la sesión asociada al requerimiento actual, o crea una nueva si *crear* es true.
boolean **isRequestedSessionIdFromCookie()**	Chequea si el SESSION ID enviado con el requerimiento proviene de una Cookie. En ese caso devuelve true.
boolean **isRequestedSessionIdFromURL()**	Chequea si el SESSION ID enviado con el requerimiento proviene de reescritura de url. En ese caso devuelve true.
boolean **isRequestedSessionIdValid()**	Chequea si el requerimiento tiene asociada una sesión válida. En ese caso devuelve true.

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-Compartir Igual 4.0 Internacional.

Soporte de Sesiones

Métodos de HTTPSession

La interface **HttpSession** tiene cuatro métodos que permiten **guardar** y **recuperar atributos** en la sesión. Los **atributos** son **objetos de alcance sesión** identificados por un nombre.

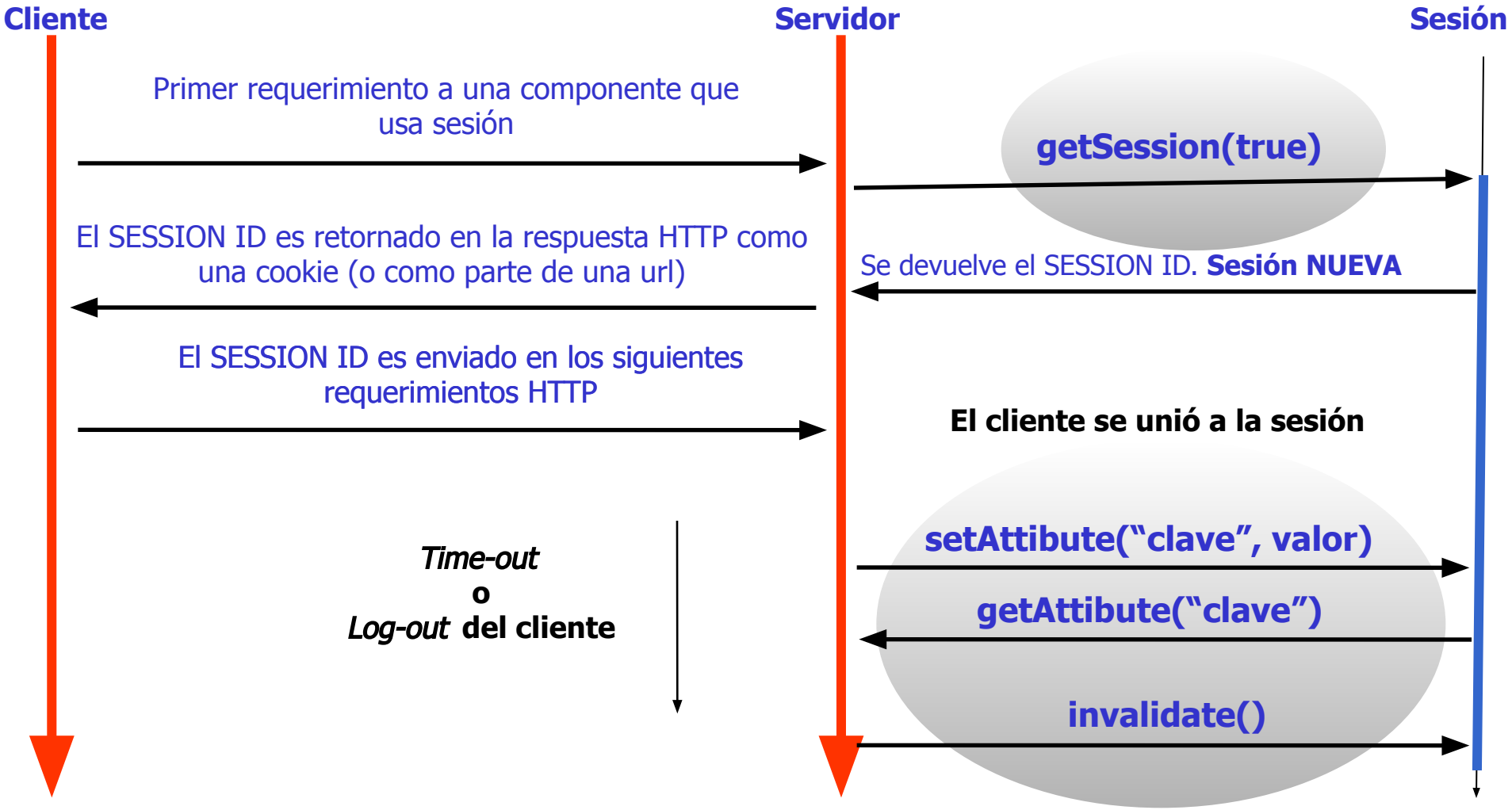
Object getAttribute(String nombre)	**Devuelve el objeto ligado a la sesión** cuyo nombre coincide con el pasado como parámetro. Usualmente el objeto retornado es una instancia de una subclase de Object y necesita ser casteado antes de ser usado.
Enumeration getAttributeNames()	**Devuelve un Enumeration** (de Strings) con todos los nombres de los objetos ligados a la sesión.
Object removeAttribute(String nombre)	**Remueve y devuelve el objeto de alcance sesión** cuyo nombre coincide con el parámetro. Devuelve null si no existe ningún objeto ligado con ese nombre.
void setAttribute(String nombre, Object valor)	**Guarda el objeto valor** en la sesión con el nombre pasado como parámetro. Cualquier objeto ligado previamente con el mismo nombre será reemplazado.

- Estos **métodos son usados** por las componentes web para **recuperar y almacenar datos** de cada cliente particular. Los datos almacenados en el **alcance sesión** son mantenidos como **pares nombre-valor**. Los *nombres* son *Strings* y los *valores* objetos Java.
- El **alcance sesión** permite acceder a los mismos objetos durante múltiples requerimientos sucesivos que comparten una sesión. Ejemplo: en una aplicación de compras por Internet los productos comprados por un usuario, necesitan mantenerse en memoria durante las múltiples peticiones del cliente.

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-Compartir Igual 4.0 Internacional.

Soporte de Sesiones

Ciclo de vida de una sesión



El cliente http no tiene manera de indicar que no se necesita más la sesión por eso cada sesión tiene un time-out asociado

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-Compartir Igual 4.0 Internacional.

Soporte de Sesiones

Leer y escribir datos en un objeto HTTPSession

- En el objeto **HttpSession** se puede **guardar cualquier tipo de objeto JAVA,** no se admiten tipos de datos primitivos.
- Para **agregar un objeto a la sesión** es necesario asignarle un **nombre único** (un String) que lo identifica.

Escribir datos en un objeto HttpSession

```
HttpSession sesion = request.getSession(true);
CarritoCompras compras = new CarritoCompras();
sesion.setAttribute("canasta", compras);
```

El objeto compras tiene alcance Sesión

Si ya existía en la sesión un objeto con nombre "canasta", lo reemplaza.

- Para **recuperar un objeto de la sesión** es necesario hacer un ***casting*** explícito a un tipo de dato JAVA.

Recuperar datos desde un objeto HttpSession

Devuelve null si no se asignó previamente un valor para el ítem.

```
HttpSession sesion = request.getSession(true);
CarritoCompras compras=(CarritoCompras)sesion.getAttribute("canasta");
```

Retorna ***null*** si no encuentra en la sesión un objeto con clave "carrito"

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-Compartir Igual 4.0 Internacional.

Soporte de Sesiones
Timeout

- Los objetos **HttpSession** tienen un **tiempo de vida finito**. Una sesión puede expirar por **2 causas**:
  - Permaneció **inactiva un determinado período de tiempo**. Se puede establecer el período de tiempo que una sesión puede estar ociosa antes de que expire:
    - Configurando el **web.xml:**

```
<session-config>
   <session-timeout>30</session-timeout>
</session-config>
```

Los contenedores establecen un tiempo de vida por defecto.

Un valor negativo o cero indica que la sesión no expira nunca.

    - Programáticamente invocando al método **setMaxInactiveInterval(30)** del objeto **HttpSession**.
  - Fue **invalidada por la aplicación.** Una aplicación puede invalidar una sesión en cualquier momento invocando al método **invalidate()** del objeto **HttpSession**. Si una sesión no es explícitamente invalidada, será invalidada automáticamente después de un cierto período de tiempo de inactividad.
- Métodos que permiten manipular el *time-out* y *accesos* a una sesión:

```
long getCreationTime()
long getLastAccessedTime()
void invalidate()
boolean isNew()
int setMaxInactiveInterval(int intervalo)
int getMaxInactiveInterval()
```

invalidate(): invalida la sesión, elimina el objeto HttpSession (remueve todos los objetos ligados).

isNew(): devuelve un valor ***boolean*** que indica si la sesión es nueva, esto es, no se ha recibido del cliente un requerimiento con ID todavía (el cliente no se unió).

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-Compartir Igual 4.0 Internacional.

Soporte de Sesiones

Ejemplo

```
package misServlets;

import java.io.IOException;
import java.io.PrintWriter;
import jakarta.servlet.*;
import jakarta.servlet.http.*;

public class ComprarProductos extends HttpServlet {

public void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
  String[] itemsSeleccionados;
  String nombre=null;
  Integer cantItems;
  HttpSession sesion = req.getSession(true);
  if  (sesion.getAttribute("cant_items")==null) {
     cantItems=new Integer(0);
  } else cantItems = (Integer) sesion.getAttribute("cant_items");

itemsSeleccionados = req.getParameterValues("item");

  if (itemsSeleccionados != null){
    for (int i=0; i<itemsSeleccionados.length; i++) {
       nombre = itemsSeleccionados[i];     //Producto elegido para comprar
       cantItems ++;

       sesion.setAttribute("Item"+ cantItems , nombre);
    }
    sesion.setAttribute("cant_items", cantItems);
  }
   resp.sendRedirect("mostrarCompra");
 }

}
```

Se obtiene la cantidad de productos elegidos para comprar

Se recuperan en el **arreglo** los productos seleccionados para comprar (desde del formulario html)

Actualiza la cantidad de productos a comprar

Guarda en el objeto sesión cada producto seleccionado bajo la clave **Item#** (Item1, Item2, etc) y la **cantidad total de ítems comprados.**

Transfiere el control a la componente web **mostrarCompra** para que genere la respuesta.

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-Compartir Igual 4.0 Internacional.

Soporte de Sesiones

Ejemplo

```
package misServlets;

import java.io.IOException;
import java.io.PrintWriter;
import jakarta.servlet.*;
import jakarta.servlet.http.*;

public class MostrarCompra extends HttpServlet {

public void doGet(HttpServletRequest req, HttpServletResponse resp)  throws ServletException, IOException {

   PrintWriter out = resp.getWriter();
   resp.setContentType("text/html");
   HttpSession sesion = req.getSession(true);
   if (sesion.isNew())
      resp.sendRedirect("compras.html");
   Integer cant_items  = (Integer) sesion.getAttribute("cant_items");
   out.print("<HTML><BODY>");
    for (int i=1; i<=cant_items.intValue(); i++) {
       String item = (String) sesion.getAttribute("Item"+i);
       out.print("<P> item“+i+” : ” + item+"</P>");
    }
     out.print("<a href=\"/JavaYAplicaciones/compras.html\">Comprar Más</a>");
     out.print("<a href=\"Salir\">Salir</a>");
     out.print("</BODY></HTML>");
     out.close();
   }
}
```

Se obtiene de la sesión la cantidad de productos seleccionados para comprar

Se recupera de la sesión cada uno de los productos comprados y se imprimen

Se construye la página HTML de respuesta

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-Compartir Igual 4.0 Internacional.

Soporte de Sesiones

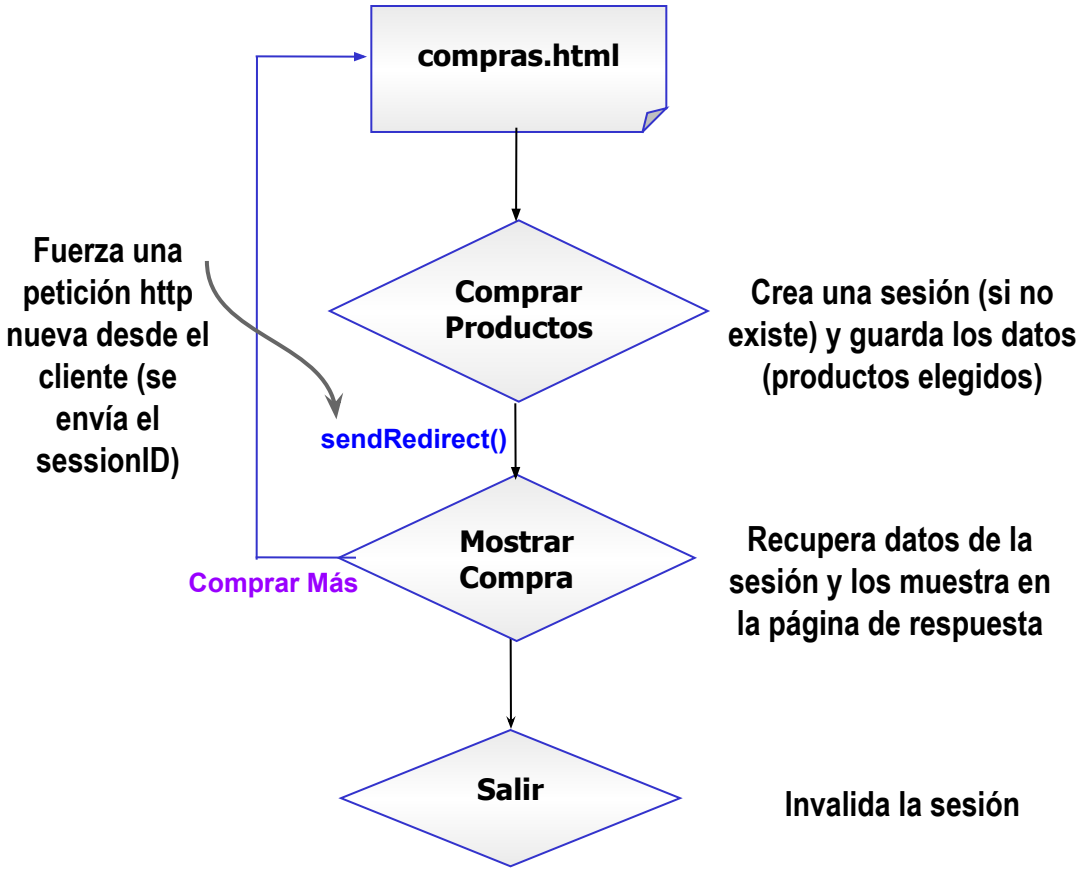
Un ejemplo

```
package misServlets;

import java.io.IOException;
import java.io.PrintWriter;
import jakarta.servlet.*;
import jakarta.servlet.http.*;

public class Salir extends HttpServlet {

public void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

   PrintWriter out=resp.getWriter();
   resp.setContentType("text/html");
   out.print("<HTML>");
   out.print("<HEAD><BODY>");
   HttpSession ses = req.getSession(false);
   if (ses!=null){
      out.print("<H1>Gracias por su compra!!</H1>");
      ses.invalidate();
   }
   out.print("</HEAD></BODY>");
   out.print("<HTML>");
   out.close();
 }

}
```


Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-Compartir Igual 4.0 Internacional.

Soporte de Sesiones

Sintetizando



Objetos HttpSession en el Servidor

	clave	Valor
JSESSIONID= 50BA23213NH213	"item01" "item02" "item03" "item04"	Cabutia Remolacha Queso Gouda Dulce de Leche
. . .	. . .	
. . .	. . .	

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-Compartir Igual 4.0 Internacional.

¿Cómo garantizar el funcionamiento de sesiones?

- El Contenedor Web puede usar diferentes mecanismos para asociar una sesión con un cliente; todos ellos involucran intercambiar un **identificador** entre el cliente y el servidor.
- El **identificador** puede ser mantenido en el cliente como una **cookie**; o la componente web puede **incluirlo en cada URL** que le envía al cliente, **reescritura de URL**.
- El **mecanismo predeterminado** usado por los servidores web JAVA para **enviar el identificador** es **cookies** y automáticamente revierten a **reescritura de URL** cuando las cookies no son soportadas o están deshabilitadas.
- La API de Servlet facilita la reescritura de URL a través de 2 métodos de la **interface HttpServletResponse**, los cuales incluyen **automáticamente el SESSION_ID en la URL.**

  encodeURL(java.lang.String url): toma como parámetro una url y retorna un String representando la url junto con el id de la sesión.

  encodeRedirectURL(java.lang.String url): trabaja de la misma manera pero se utiliza para redirección.

Ejemplos de uso: **/applic/MiServlet;jsessionid=90Y643232hTsy67KK98isefds9923432**

```
out.println("<a href=\"/applic/MiServlet\">Seguir</a>);
out.print("<a href="+ response.encodeURL("/applic/MiServlet") +">Seguir </a>");

response.sendRedirect("/applic/MiServlet")
response.sendRedirect(response.encodeRedirectURL("/applic/MiServlet"))
```


Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-Compartir Igual 4.0 Internacional.

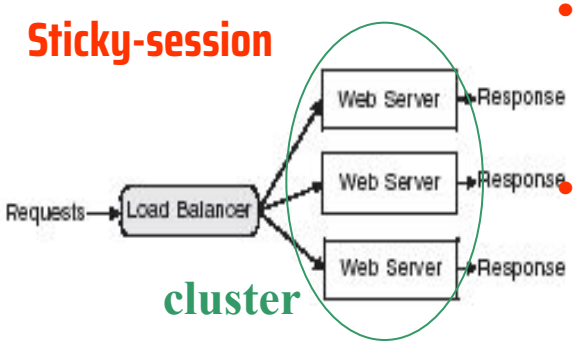
Sesiones en ambientes multi-servidor

Persistencia de la sesión

Las **sesiones HttpSession mantienen en el servidor el estado** de las interacciones con los clientes y el **modelo** que hemos presentado **guarda el estado en la memoria de un único servidor web.**

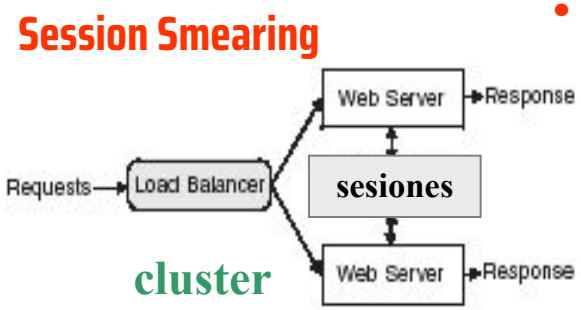
¿Cómo manejar el estado en un ambiente multi-servidor ?

Todos los requerimientos del mismo usuario van al mismo servidor físico:



- **mal balanceo de carga:** si toda la información de la sesión reside en un único servidor y se fuerza a que todos los requerimientos vayan al mismo servidor, se arruina el balanceo.
- **no hay respaldo del estado**: ¿Qué pasa si el servidor se cae? Si el estado de la sesión está guardado en la memoria de un sólo servidor, la información se pierde.

Todos los servidores del cluster comparten la misma información:



- **solución preferida!!**: en este modelo la información de la sesión del usuario está disponible para cualquier servidor del cluster. El estado puede estar en memoria o en archivos.

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-Compartir Igual 4.0 Internacional.

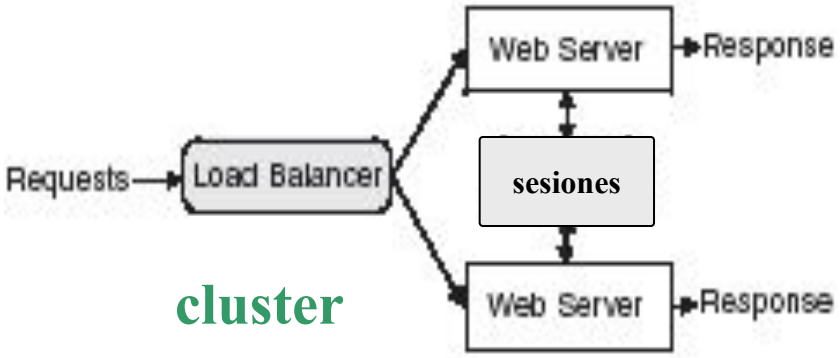
Sesiones en ambientes multi-servidor

Persistencia de la sesión

Todos los servidores del cluster comparten la misma información: Session Smearing

Los **servidores del cluster pueden replicar el estado de la sesión en memoria**: en general, el servidor que recibe el primer requerimiento crea la sesión (objeto HTTPSession) y la replica al resto de los servidores. El proceso de copiar el estado de una sesión desde un servidor a otro es llamado **réplica en memoria**. Todos los servidores mantienen actualizados los estados de las sesiones.

Los **servidores también pueden mantener el estado de las sesiones usando persistencia** basada en **archivos o bases de datos (JDBC)**.



Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-Compartir Igual 4.0 Internacional.

Sesiones en ambientes multi-servidor

Persistencia de la sesión

Session Smearing: todos los servidores comparten la misma información

(A) Tener la sesión replicada en memoria en todos los servidores del cluster

La especificación de Servlet define el elemento **<distributable>** el cual se incluye en el **web.xml** para indicar que la aplicación web puede ser distribuida a través de múltiples servidores.

Para que una sesión pueda compartirse entre los servidores del cluster se debe cumplir con el siguiente requisito: los **objetos de alcance sesión** deben implementar la interface **java.io.Serializable**.

Los contenedores que soportan ambientes multi-server, invocarán los métodos **writeObject() y readObject()** para **guardar** y **recuperar** información de la sesión. Los objetos de alcance sesión pueden sobreescribir estos métodos para proveer un comportamiento customizado cuando los objetos son movidos entre servidores (JVMs). Si no se sobreescriben se tiene un comportamiento por defecto.

Implementar la interface **java.io.Serializable** es generalmente lo único que **tiene que hacer un programador para que todos los objetos de alcance sesión puedan distribuirse** en un ambiente multi-servidor.

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-Compartir Igual 4.0 Internacional.

Sesiones en ambientes multi-servidor

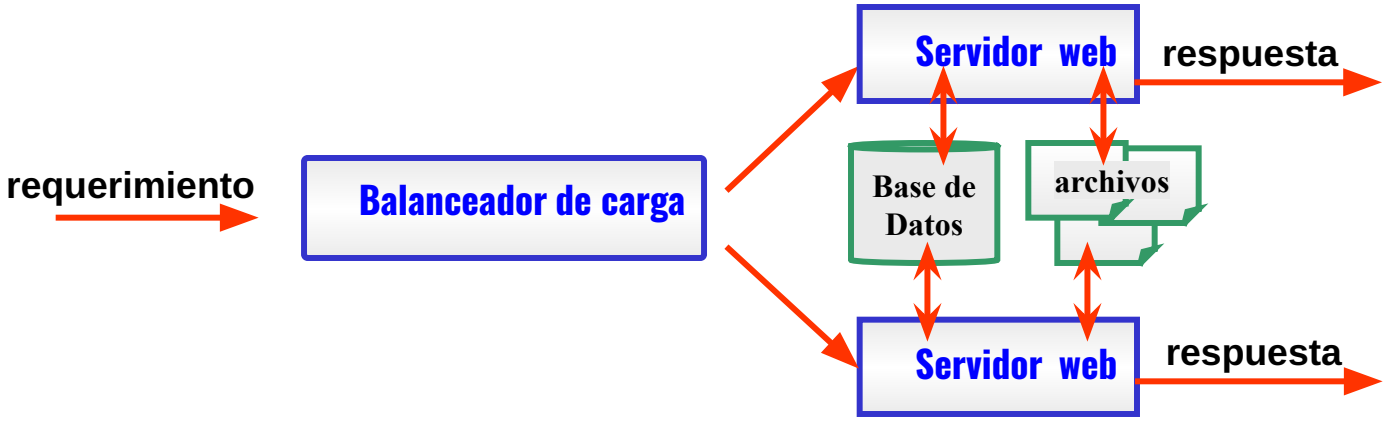
Persistencia de la sesión

Session Smearing: todos los servidores comparten la misma información

(B) Compartir el estado de las sesiones usando una base de datos o archivos.

Consiste en usar un servidor centralizado para manejar la información de las sesiones. Esta es una solución atractiva porque le permite a los programadores tener un soporte completo sobre la persistencia del estado. También podrían guardarse en archivos.

La API de Servlets provee clases que ayudan a construir la base de datos o los archivos: **HttpSessionListener**. Estas clases son útiles porque permiten cargar datos basándose en la creación y destrucción de una sesión.



En cuanto a esta solución hay que tener en cuenta que los accesos a base de datos son casi siempre los módulos que bajan la *performance* de una aplicación.

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-Compartir Igual 4.0 Internacional.

Sesiones en ambientes multi-servidor

Persistencia de la sesión

Conclusiones

- Cuando existe un cluster de servidores las sesiones pueden manejarse bajo 2 modelos: **sticky session (sesiones pegajosas)** y **session smearing** (**sesiones replicadas)**.
- Las **sticky session** son aquellas que se guardan en el servidor que recibió el primer requerimiento. Los otros servidores miembros del cluster no tienen conocimiento del estado de la sesión. Si este servidor se cae, la información de la sesión se pierde.
- En las **session smearing** el estado de la sesión es copiado a todos los servidores del cluster. Los datos también son actualizados cuando el estado de la sesión cambia. La replicación es gestionada automáticamente nivel de servidor de aplicaciones.
- Las **sticky session** son simples y fáciles de manejar, ya que no se necesita replicar en otros servidores. Esto resulta en menos *overhead* y mejor *performance*. Pero si el servidor se cae, como todos los datos de la sesión están en su memoria y no fue replicado, se pierde todo. Esto puede causar problemas en determinados tipos de aplicaciones. Por ejemplo si esto ocurre en medio de una transacción bancaria.

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-Compartir Igual 4.0 Internacional.