Herramientas de trabajo (5)

Miguel Angel Piña Avelino

Ingeniería de Software,
Facultad de Ciencias, UNAM

30 de septiembre de 2018

Índice

Alcance de los beans manejados JSF

Ámbito de un Bean Manejado

En los beans se define un ámbito (*bean scope*). Este ámbito se refiere principalmente a la información que contiene y que es mantenida durante la vida del bean

Ámbito de un Bean Manejado

En los beans se define un ámbito (*bean scope*). Este ámbito se refiere principalmente a la información que contiene y que es mantenida durante la vida del bean

El ámbito de los beans determina su ciclo de vida: cuándo se crean y destruyen instancias del bean y cuándo se asocian a las páginas *JSF*. Es muy importante definir correctamente el ámbito de un bean, porque en el momento de su creación se realiza su inicialización. *JSF* llama al método constructor de la clase, donde habremos colocado el código para inicializar sus valores, creando una instancia de la clase que pasará a ser gestionada por el framework.

Ámbito de un Bean Manejado

Ámbitos

En JSF se definen los siguientes ámbitos para los beans:

- RequestScope
- ViewScope
- SessionScope
- ApplicationScope

Ámbito de un Bean Manejado

RequestScoped

Se define con la anotación `@RequestScoped` en la clase. El bean se asocia a una petición HTTP. Cada nueva petición (cuando desde el navegador se abre una página por primera vez una página o se recarga) crea un nuevo bean y lo asocia con la página. Dada su corta vida, se recomienda usar este ámbito para el paso de mensajes (bien sea de error o de estatus), o para cualquier otro tipo de dato que no sea necesario propagar a lo largo de la aplicación.

Ámbito de un Bean Manejado

ViewScoped

Se define con la anotación **@ViewScoped** en la clase. Un bean en este ámbito persistirá mientras se repinte la misma página (vista = página JSF), al navegar a otra página, el bean sale del ámbito. Es bastante útil para aplicaciones que usen Ajax en parte de sus páginas.

Ámbito de un Bean Manejado

SessionScoped

Se define con la anotación **@SessionScoped** en la clase. Las sesiones se definen internamente con el API de *Servlets*. Una sesión está asociada con una visita desde una navegador. Cuando se visita la página por primera vez se inicia la sesión. Cualquier página que se abra dentro del mismo navegador comparte la sesión. La sesión mantiene el estado de los elementos de nuestra aplicación a lo largo de las distintas peticiones. Se implementa utilizando cookies o reescritura de URLs, con el parámetro jsessionid. Una sesión no finaliza hasta que se invoca el método invalidate en el objeto *HttpSession*, o hasta que se produce un timeout.

Ámbito de un Bean Manejado

ApplicationScoped

Se define con la anotación **@ApplicationScoped**. Los beans con este ámbito viven asociados a la aplicación. Definen singletons que se crean e inicializa sólo una vez, al comienzo de la aplicación. Se suelen utilizar para guardar características comunes compartidas y utilizadas por el resto de beans de la aplicación.

Sesiones

Sesiones

HTTP Stateless

HTTP es un protocolo sin estado, eso quiere decir que nos encontraremos con el problema de compartir estado entre las páginas de nuestra aplicación web, entonces, ¿cómo lo hacemos?

Sesiones

HTTP Stateless

HTTP es un protocolo sin estado, eso quiere decir que nos encontraremos con el problema de compartir estado entre las páginas de nuestra aplicación web, entonces, ¿cómo lo hacemos?

Java HTTPSession

Para resolver el problema anterior, Java EE proporciona una clase llamada `HTTPSession`, la cuál tiene una estructura de diccionario (`HashMap`).

Sesiones

La interfaz `HTTPSession (javax.servlet.http.HTTPSession)` provee una forma de identificar a un usuario a través de más de una petición de página o visita a un sitio web, de modo que se pueda almacenar información sobre el usuario.

Sesiones

La interfaz `HTTPSession (javax.servlet.http.HTTPSession)` provee una forma de identificar a un usuario a través de más de una petición de página o visita a un sitio web, de modo que se pueda almacenar información sobre el usuario.

El contenedor de servlets usa esta interfaz para crear una sesión entre el cliente HTTP y el servidor HTTP. La sesión se persiste por un periodo de tiempo especificado, a través de más de una conexión o petición de página desde el usuario.

Sesiones

Una sesión usualmente corresponde a un usuario, quien podría visitar el sitio varias veces. El servidor puede mantener una sesión de varias formas, ya sea usando cookies o reescribiendo URL's.

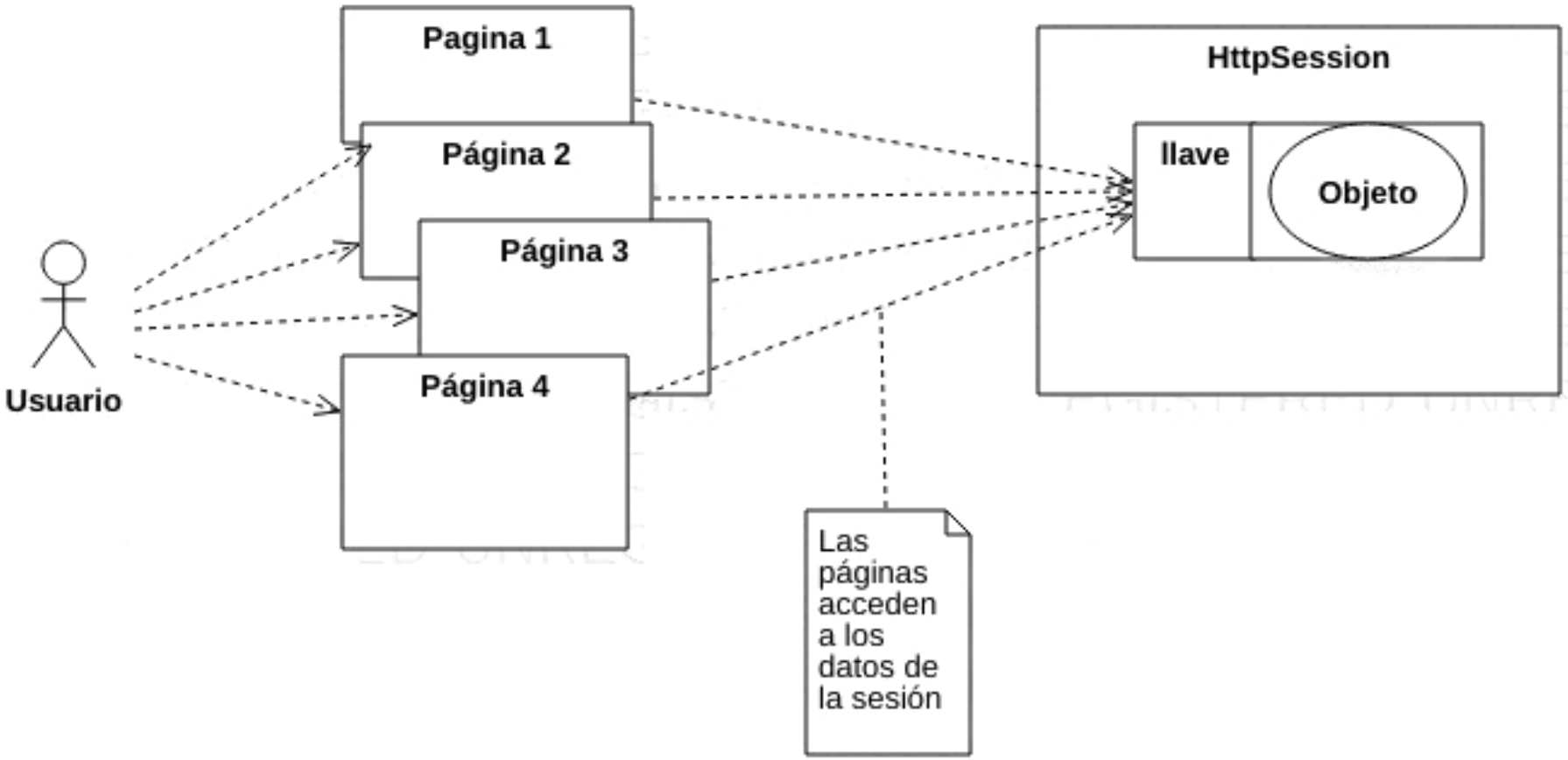
Sesiones

Una sesión usualmente corresponde a un usuario, quien podría visitar el sitio varias veces. El servidor puede mantener una sesión de varias formas, ya sea usando cookies o reescribiendo URL's.

La interfaz HTTPSession permite:

- Ver y manipular información sobre la sesión, tal como un identificador de sesión, fecha de creación y última fecha de acceso.
- Asociar objetos a sesiones, permitiendo que la información de usuario persista a través de múltiples conexiones de usuario.

Sesiones


Pagina 1
Página 2
Página 3
Página 4
Usuario
HttpSession
llave
Objeto
Las páginas acceden a los datos de la sesión

¿Cómo obtener una sesión en JSF?

En JSF no se puede invocar directamente a la creación de una sesión a partir de la clase HTTPRequest, si no hay que invocarla a partir de la clase FacesContext, la cuál contiene toda la información del estado por petición asociada al procesamiento de una petición de JavaServerFaces, y el renderizado de la correspondiente respuesta. La forma de hacerlo es la siguiente:

```
HTTPSession session = (HTTPSession)
          FacesContext.getCurrentInstance()
                      .getExternalContext()
                      .getSession(false);
```

¿Cómo obtener una sesión en JSF?

Adicionalmente, podemos obtener el diccionario de atributos de la sesión para guardar o quitar información del usuario, de modo que sea utilizada a través de la navegación entre páginas. La forma de obtenerla es la siguiente:

```
Map<String, Object> sessionMap = (Map<String, Object>)
                    FacesContext.getCurrentInstance()
                                .getExternalContext()
                                .getSessionMap();
```

Y cómo es un mapa, podemos usar los métodos put y get, los cuáles serían equivalentes a setAttribute y getAttribute de HTTPSession.

Ejemplo

```
@ManagedBean
@RequestScoped
public class LoginController {

  private final EntityManagerFactory emf;
  private final LoginJpaController jpaController;
  private final UsuarioJpaController usuarioJpaController;
  private UsuarioBean usuario;

  public LoginController() {
    FacesContext.getCurrentInstance().getViewRoot()
                .setLocale(new Locale("es-Mx"));
    emf = EntityProvider.provider();
    jpaController = new LoginJpaController(emf);
    usuarioJpaController = new UsuarioJpaController(emf);
    usuario = new UsuarioBean();
  }
  public UsuarioBean getusuario() {
    return usuario;
  }
  public void setUsuario(UsuarioBean usuario) {
    this.usuario = usuario;
  }
```

Ejemplo

```
  public String canLogin() {
      Login l = jpaController
      .findLogin(usuario.getUsuario(), usuario.getContraseña());
      boolean logged = l != null;
      if (logged) {
        Usuario u = usuarioJpaController.findUsuarioByLoginId(l.getId());
        FacesContext context = getCurrentInstance();
        context.getExternalContext().getSessionMap().put("usuario", l);
        context.getExternalContext().getSessionMap().put("datos", u);
        return "secured/inicio?faces-redirect=true";
      }
      return "registro?faces-redirect=true";
    }

    public String logout() {
      FacesContext context = getCurrentInstance();
      context.getExternalContext().invalidateSession();
      return "index?faces-redirect=true";
    }
}
```

Filtros (Filters)

Filtros

Dentro de la especificación de Java Servlets, existe un componente llamado filtros. Estos nos ayudan a analizar o transformar las peticiones hechas a las páginas Web, ya sean JSFs o servlets. Estos filtros trabajan de forma encadenada, por lo que se puede crear una serie de filtros que van pasando el control de uno al otro.

Filtros

Dentro de la especificación de Java Servlets, existe un componente llamado filtros. Estos nos ayudan a analizar o transformar las peticiones hechas a las páginas Web, ya sean JSFs o servlets. Estos filtros trabajan de forma encadenada, por lo que se puede crear una serie de filtros que van pasando el control de uno al otro.

Los Filtros son muy sencillos de implementar, y son componentes altamente reutilizables que nos dan mucho poder sobre las aplicaciones web. Nos permite por ejemplo, controlar el acceso a la aplicación, manejo de compresión de datos, transformaciones, auditoria, registro, cifrado, manejo de sesiones, en fin, un sinnúmero de utilidades que se adaptan a nuestras necesidades.

Filtros

Para hacer uso de los filtros, tenemos que crear una clase que implemente la interfaz **javax.servlet.Filter**. Esta interfaz contiene 3 métodos:

- **doFilter** Este es el mas importante ya que es el que realiza toda la actividad del filtro. Además de disparar el resto de los filtros.
- **init** Este método es llamado cuando el filtro es inicializado en el contenedor.
- **destroy** Este método es llamado cuando el filtro es destruido por el contenedor.

Para mostrar como es un filter, vamos a mostrar la implementación de uno en la siguiente diapositiva.

Ejemplo de filter

```
public class SessionFilter implements Filter {
    public SessionFilter() {}
    /**
     * @param request The servlet request we are processing
     * @param response The servlet response we are creating
     * @param chain The filter chain we are processing
     * @exception IOException if an input/output error occurs
     * @exception ServletException if a servlet error occurs
     */
    public void doFilter(ServletRequest request, ServletResponse response,
                         FilterChain chain)
            throws IOException, ServletException {

        HttpServletRequest req = (HttpServletRequest) request;
        HttpServletResponse res = (HttpServletResponse) response;
        HttpSession session = req.getSession(true);
        String pageRequest = req.getRequestURL().toString();
        if (session.getAttribute("usuario") == null) {
            res.sendRedirect(req.getContextPath() + "/index.xhtml");
            return;
        }
        chain.doFilter(request, response);
    }
```

Ejemplo de filter

```
    @Override
    public void init(FilterConfig fc) throws ServletException { }
    @Override
    public void destroy() {}
}
```

Ejemplo de filter

Además agregamos al archivo **web.xml** para que sea ejecutado cuando se levante la instancia del servidor.

```
<!-- aplicamos el filtro sobre las páginas deseadas.-->
 <filter>
   <filter-name>SessionFilter</filter-name>
   <filter-class>com.miguel.proyecto.web.SessionFilter</filter-class>
 </filter>
 <filter-mapping>
   <filter-name>SessionFilter</filter-name>
   <url-pattern>/secured/*</url-pattern>
 </filter-mapping>

 <session-config>
   <session-timeout>60</session-timeout>
```