Servlets

1. Atributos y Alances
   - request, sesion y aplication
2. Contexto de la aplicación web: ServletsContext
   - Parámetros de inicialización del Contexto
3. Listeners de Contexto
4. Acceso a recursos estáticos de la aplicación
   - Archivos binarios
   - Archivos de Propiedades

Atributos y Alcances

¿qué es un atributo y dónde pueden guardarse?

Un atributo es un objeto java guardado dentro de algunos de los siguientes objetos (conocidos como alcances) de la API de servlets: **`ServletContext`**, **`HttpSession`** o **`HttpServletRequest`**.

alcance aplicación	alcance sesión	alcance request
Los objetos ligados a este contexto pueden ser usados por todos los **`servlets/JSP`** de la aplicación y duran mientras la aplicación web esté ejecutando.	Los objetos ligados a la sesión de un usuario pueden ser usados por todos los **`servlets/JSP`** que accedan a esa sesión y permanecen ahí mientras dure la sesión.	Los objetos ligados al request pueden ser usados por todos los **`servlets/JSP`** que dispongan de ese request y permanecen en él, mientras dure el request. Los atributos NO son parámetros.

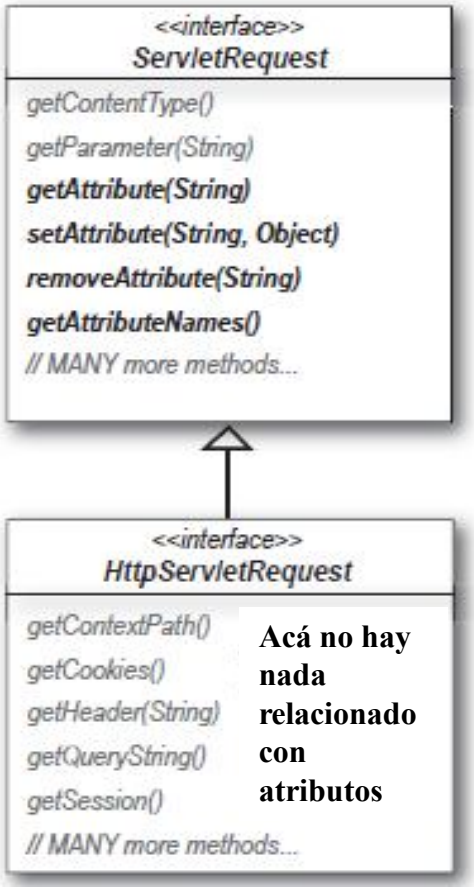
Se pueden usar los siguiente métodos para ligar, recuperar y eliminar atributos de cualquier alcance.

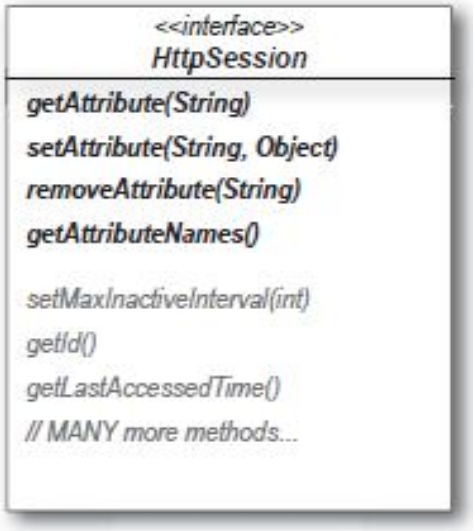
```
void setAttribute(String nombre, Object attr)
Object getAttribute(String nombre)
void removeAttribute(String nombre)
Enumeration getAttributeNames()
```

Notar que el tipo de dato de retorno del método getAttribute es Object y comúnmente se necesita usar casting para recuperar el tipo original.

Atributos y Alcances

Los atributos pueden ser ligados a uno de los siguientes alcances: **requerimiento**, **sesión** o **contexto**. Los métodos de la API para ligar atributos son exactamente los mismos.





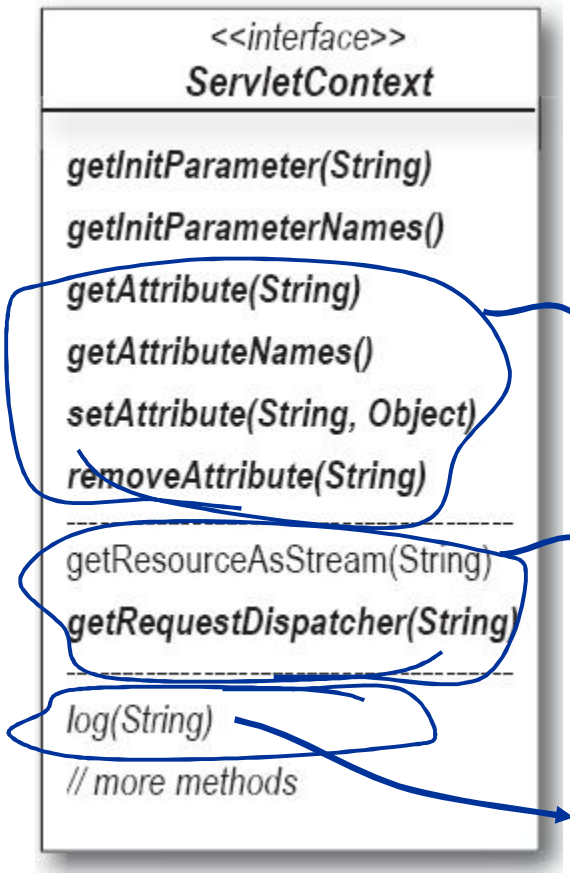
Veremos más adelante

Object getAttribute(String name)
void setAttribute(String name, Object value)
void removeAttribute(String name)
Enumeration getAttributeNames()

ServletContext

¿Qué es?

La interface **`ServletContext`** define una vista de la aplicación web para los servlets. A través del objeto **`ServletContext`** se pueden lograr varias funcionalidades observadas en el diagrama de la clase.



Acceder a parámetros de inicialización de la aplicación web

Ligar y recuperar objetos con alcance aplicación

Devolver el recurso ubicado en el path especificado, como un objeto IInputStream)

almacenar eventos de la aplicación

Cada Contenedor Web provee una implementación específica de la interface **`ServletContext`**.

Existe un único objeto **`ServletContext`** por aplicación web ejecutándose en el Contenedor Web. En el caso de una instalación multi-server, la aplicación web tendrá un objeto **`ServeltContext`** por JVM.

Los servlets disponen del método **`getServletContext`**() que retorna una referencia al objeto ServletContext. El alcance del objeto **`ServletContext`** es toda la aplicación.

Para hacer logging de aplicaciones web se recomienda usar una API que permita desde cualquier clase acceder y usar el logging. Ejemplo: Log4j, http://jakarta.apache.org/log4j o la API estándar para logging del JDK (java.util.logging).

ServletContext

Parámetros de Inicialización de la Aplicación Web

De manera similar a los Servlets, las aplicaciones también pueden tener parámetros de inicialización

web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?> <!DOCTYPE web-app PUBLIC "-//Sun Microsystems,
Inc.//DTD Web Application 2.3//EN" "http://java.sun.com/j2ee/dts/web-app_2_3.dtd">
<web-app>
<context-param>
      <param-name>email</param-name>
       <param-value>admin@info.unlp.edu.ar</param-value>
 </context-param>
. . .
</web-app>
```

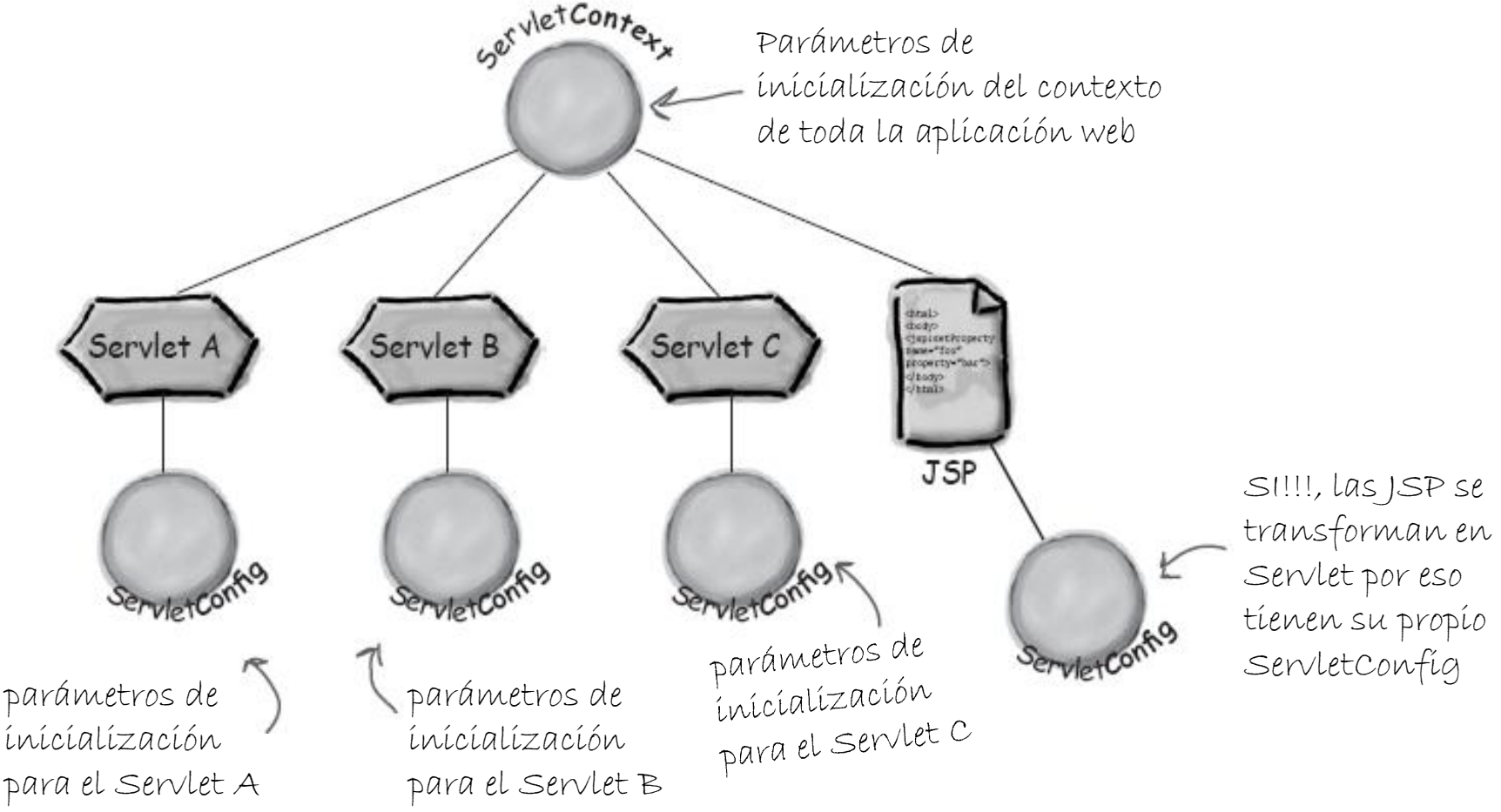
Los parámetros de inicialización de la aplicación web pueden ser accedidos desde todas las componentes web.

```
package com.servlets;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class PaginaError extends HttpServlet {
 public void doGet(HttpServletRequest req,HttpServletResponse resp)..{
     resp.setContentType("text/html");
     PrintWriter out=resp.getWriter();
     out.println("<html><head>");
     out.println("<title> Ocurrió un Error </title>");
     out.println("</head><body>");
     ServletContext sc = this.getServletContext();
     String mail = sc.getInitParameter("email");
     out.println("<h1> Error inesperado </h1>");
     out.println("Por favor, repórtelo a: "+ mail);
     out.println("</body></html>");
   }
}
```

Parámetros de inicialización
de Servlets y del Contexto

Hay un único **`ServletContext`** para toda la aplicación web y todas las componentes lo comparten.
El contenedor crea el objeto **`ServletContext`** cuando se carga la aplicación.
Cada servlet en la aplicación tiene su propio **`ServletConfig`**.



Los atributos nos son parámetros

A manera de repaso pensemos las diferencias entre atributos y parámetros

	Atributos	Parámetros
Tipos	Aplicación/Contexto **`getServletContext().getAtributte("xx")`**	Parámetros de inicialización de la Aplicación
	Requerimiento **`request.getAtributte("xx")`**	Parámetros del Request
	Sesión **`getSession().getAtributte("xx")`**	-
	-	Parámetros de inicialización del Servlet
Métodos para setear	**`setAtribute(String nombre,Object valor)`**	Los parámetros de inicialización se configuran en el **`WEB.XML`** o por anotación y los del Request son seteados automáticamente.
Métodos para recuperar	**`getAttribute(String nombre)`**	**`getParameter(String parametro)`** **`getInitParameter(String nombre)`** **`(*)`**
Tipos de Dato	**`Object`**	**`String`**

`(*) getServletContext().getInitParameter("valor")` ó
`getServletConfig().getInitParmeter("valor")`

Los parámetros de inicialización

WEB.XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web
Application 2.3//EN" "http://java.sun.com/j2ee/dts/web-
app_2_3.dtd">
<web-app>
  <!--Inicializa las salas -->
  <context-param>
    <param-name>salas</param-name>
    <param-value>WEB-INF/salas.txt</param-value>
  </context-param>
  . . .
```

del Contexto/de la Aplicación

```
getServletContext().getInitParameter("salas")
```

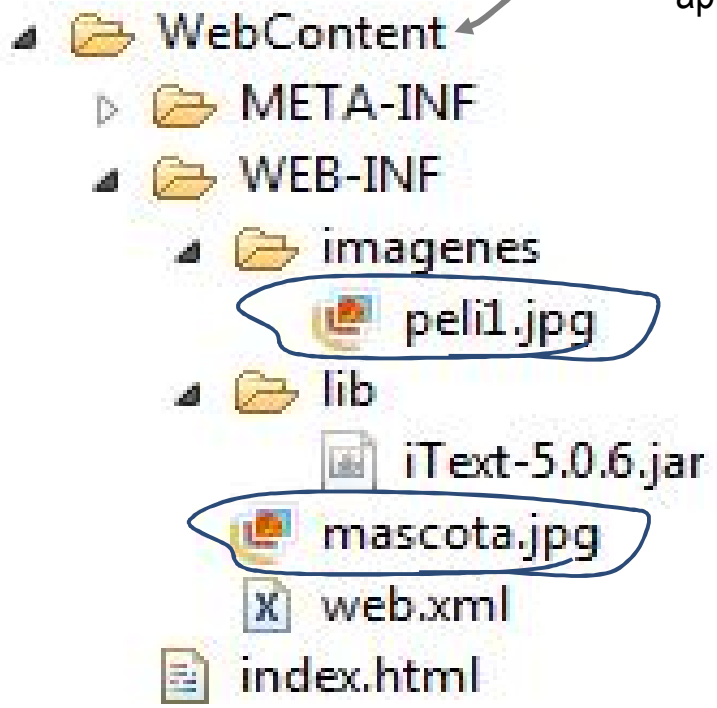
```
<servlet>
 <servlet-name>ServletFecha</servlet-name>
 <servlet-class>misServlets30.ServletFecha</servlet-class>
   <init-param>
     <param-name>dia</param-name>
     <param-value>Hoy es:</param-value>
   </init-param>
</servlet>
<servlet-mapping>
    <servlet-name>ServletFecha</servlet-name>
    <url-pattern>/ServletFecha</url-pattern>
</servlet-mapping>
. . .
</web-app>
```

del Servlet ServletFecha

```
getServletConfig().getInitParameter("dia")
```

ServletContext
Recursos web estáticos

El objeto ServletContext provee acceso a la jerarquía de documentos estáticos que forman parte de la aplicación Web, como por ej. archivos TXT, GIF, JPEG.



Aplicación Web = WebContent de Eclipse que se traduce al nombre de la aplicación en los contenedores Web. **Por ejemplo** `Cupones`

Nombre de la aplicación web

`http://www.cinemacity.com/Cupones`

Paths virtuales de las imágenes:

`/WEB-INF/mascota.jpg`
`/WEB-INF/imagenes/peli1.jpg`

Todos los recursos de una aplicación web son abstraídos en directorios virtuales a partir de la raíz de la aplicación web. Un directorio virtual comienza con "/" y continúa con un path virtual a directorios y recursos.

ServletContext

Recursos web estáticos

El objeto **ServletContext** provee acceso a la jerarquía de documentos estáticos a través de los siguientes métodos:

- **URL getResource(String path)**
  Devuelve la URL al recurso que coincide con el **path** dado como parámetro. El **path** debe comenzar con "/" y es relativo a la raíz de la aplicación web.

```
URL unaUrl = getServletContext().getResource("/WEB-INF/mascota.jpg");
```

- **String getRealPath(String path)**
  Devuelve un String que contiene el path real del path virtual especificado como parámetro.

```
String str = getServletContext().getRealPath("/WEB-INF/mascota.jpg")
```

```
D:\eclipse_workspaces\workspaceTTPS2015\.metadata\.plugins\org.eclipse.wst.server.core\tmp0\wtpwebapps\TicketComedor\WEB-INF\mascota.jpg
```

- **InputStream getResourceAsStream(String path)**
  Devuelve el recurso ubicado en el **path** especificado como parámetro, como un objeto **InputStream**. Los datos en el **InputStream** pueden ser de cualquier tipo y longitud. El path debe empezar con "/" y es relativo a la raíz de la aplicación web.

```
InputStream is =
    this.getServletContext().getResourceAsStream("/WEB-INF/salas.txt");
```

```
file:/D:/eclipse_workspaces/workspaceTTPS2015/.metadata/.plugins/org.eclipse.wst.server.core/tmp0/wtpwebapps/TicketComedor/WEB-INF/mascota.jpg
```

ServletContext

Recursos web estáticos – Archivos de Propiedad

Un archivo de propiedades está formado por lineas de texto de la forma **clave=valor**. La extensión de estos archivos es **properties**. Estos archivos de texto son comúnmente usados para guardar datos de configuración de una aplicación.
A continuación se muestra el contenido de un archivo llamado **datos.properties.**

```
jdbc.driverClassName = com.mysql.jdbc.Driver
jdbc.url = jdbc:mysql://localhost:3306/mvcwebsystique
jdbc.username = root
jdbc.password = root
hibernate.dialect = org.hibernate.dialect.MySQLDialect
hibernate.show_sql = true
```

La clase **ResourceBundle**, del paquete **java.util**, facilita la lectura de este tipo de archivos. La clase tiene un método **static getBundle(String name)** que obtiene un objeto **ResourceBundle** usando el **name**.

Lectura de los datos del archivo:

```
ResourceBundle rb = ResourceBundle.getBundle("recursos.datos");

String usr = rb.getString("jdbc.username");

Enumeration<String> keys = rb.getKeys();
while (keys.hasMoreElements()) {
    String key = keys.nextElement();
    String value = rb.getString(key);
    System.out.println(key + ": " + value);
}
```

recrusos es una carpeta interna a **/src**, donde se guardó el archivo **datos.properties**

Si se conoce la clave se puede leer directamente el valor. En este caso **usr** quedaría con el valor **root**

ServletContext
Listeners de Contexto

¿Cómo haríamos para correr algún código antes de que un Servlet –o JSP– pueda responder a un requerimiento? ¿Podemos saber cuando la aplicación web arranca?

Necesitamos de un objeto java que inicialice la aplicación web. Podemos crear una clase separada (ni Servlet, ni JSP), que pueda escuchar por 2 eventos del ciclo de vida de la aplicación: creación y destrucción de la aplicación web. Para lograr esto, la clase debe implementar la interface **`ServletContextListener`** => Listeners de Contexto

```
package mislisteners;
public class InicializaSalas implements ServletContextListener {
 private ServletContext context = null;
 private HashMap<String, String> salas = new HashMap();

 public void contextInitialized(ServletContextEvent event) {
     //Se leen datos de un salas.txt, se guardan en una Tabla de Hash
     context = event.getServletContext();
     String nomArchSalas = context.getInitParameter("salas");
     InputStream is = context.getResourceAsStream(nomArchSalas);
     actualizaSalas(is);
     // Se liga la tabla al context
     context.setAttribute("salas", salas);
 }
  public void contextDestroyed(ServletContextEvent sce) {..}
}
```

Le pide al evento por el ServletContext

Este método lee el archivo salas.txt y arma con esos datos el objeto **salas**.

Se liga un atributo al contexto de la aplicación. Ahora cualquier componente puede recuperarlo usando el método: `getAttribute("salas")`

ServletContext

Listener de contexto – configuración en el WEB.XML

Los **servlet listeners** deben configurarse en el archivo **`web.xml`** y de esta manera el Contenedor Web se entera de su existencia. Para publicarlos se utiliza el tag **`<listener>`**.

El Contenedor Web crea **una instancia de cada clase listener** declarada y **registra** dicho objeto para ser **notificado ante la ocurrencia de eventos**.

salas.txt

```
sala1,170
sala2,90
sala3,150
sala4,220
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web
Application 2.3//EN" "http://java.sun.com/j2ee/dts/web-app_2_3.dtd">
<web-app>
  <!--Inicializa las salas -->
  <context-param>
    <param-name>salas</param-name>
    <param-value>WEB-INF/salas.txt</param-value>
  </context-param>
  <listener>
    <listener-class>mislisteners.InicializaSalas</listener-class>
  </listener>
</web-app>
```

Es posible declarar múltiples listeners. El orden en que son declarados determina el orden en el que son invocados por el Contenedor Web.

El tag **`listener`** se declara después del tag **`context-param`** y antes de la definición de todos los servlets de la aplicación web.

Este tag **`listener`** puede reemplazarse por al anotación **`@WebListener,`** sin embargo no existe una anotación para los parámetros de inicialización del contexto.

ServletContext

Listeners de Contexto completo


```
package mislisteners;
public class InicializaSalas implements ServletContextListener {
 private ServletContext context = null;
 private HashMao<String, String> salas = new HashMap<String, String>();
 public void contextInitialized(ServletContextEvent event) {
    context = event.getServletContext();
    String nomArchSalas = context.getInitParameter("salas");
    InputStream is = context.getResourceAsStream(nomArchSalas);
    actualizaSalas(is); //carga la tabla de hash salas
    context.setAttribute("salas", salas);
 }
 private void actualizaSalas(InputStream is) {
    try {
      int posComa=0;String nom, disponible=null;
      BufferedReader r = new BufferedReader(new InputStreamReader(is));
      String linea = r.readLine();
      while (linea != null) {
        nom = linea.substring(0, linea.indexOf(","));
        disponibles = linea.substring(linea.indexOf(",") + 1);
        salas.put(nom, disponibles);
        linea = r.readLine();
      }
    } catch (IOException e) {...}
 }
}
```

Se lee del web.xml el nombre del archivo

Devuelve el recurso ubicado en el path especificado, como un objeto InputStream. Los datos en el InputStream pueden ser de cualquier tipo y longitud. El método retorna null si no existe el recurso en el path especificado.

Se liga el objeto al contexto

WEB.XML

```
<web-app>
  <context-param>
    <param-name>salas</param-name>
    <param-value>WEB-INF/salas.txt</param-value>
  </context-param>
  . . .
</web-app>
```