Contexto de Servlets

PDF

Vamos a estudiar los elementos que podemos utilizar para establecer una comunicación entre los distintos servlets de una aplicación web, así como la comunicación entre los servlets y otros elementos de la aplicación web.

Comenzaremos estudiando el contexto de los servlets (*Servlet Context*). Este objeto de contexto es propio de cada aplicación web, es decir, tendremos un objeto `ServletContext` por aplicación web, por lo que nos servirá para comunicar los servlets de dicha aplicación.

```
public void init(ServletConfig config)
```

En la inicialización del servlet (método **init**), se nos proporcionará un objeto **ServletConfig** como parámetro. Mediante este objeto podemos:

- Obtener el nombre del servlet, que figurará en el descriptor de despliegue de la aplicación web (`web.xml` en Tomcat).

```
String nombre = config.getServletName();
```

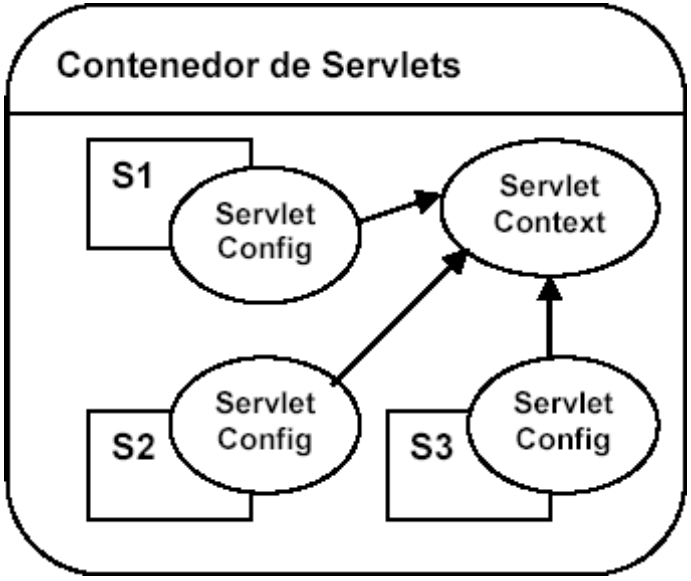
- Obtener los valores de los parámetros de inicialización del servlet, que se hayan establecido en el descriptor de despliegue. Tanto los nombres como los valores de los parámetros son cadenas de texto (`String`).

```
String valor_param = config.getInitParameter(nombre_param);
Enumeration nombres_params = config.getInitParameterNames();
```

- Acceder al objeto de contexto de la aplicación a la que pertenece el servlet.

```
ServletContext context = config.getServletContext();
```

Esta última función es la más importante, ya que nos permite acceder al objeto de contexto global de la aplicación, con el que podremos realizar una serie de operaciones que veremos a continuación.



Tanto el objeto `ServletConfig` como `ServletContext` pueden ser obtenidos directamente desde dentro de nuestro servlet llamando a los métodos `getServletConfig` y `getServletContext` respectivamente, definidos en `GenericServlet`, y por lo tanto disponibles en cualquier servlet.

Atributos de contexto

Dentro del objeto de contexto de nuestra aplicación podremos establecer una serie de atributos, que serán globales dentro de ella. Estos atributos son un conjunto de pares <*nombre, valor*> que podemos establecer y consultar desde los distintos servlets de nuestra aplicación web. El nombre del atributo será una cadena de texto (`String`), mientras que el valor podrá ser cualquier objeto java (`Object`).

Para consultar el valor de un atributo utilizaremos:

```
Object valor = context.getAttribute(nombre);
```

Daremos valor a un atributo con:

```
context.setAttribute(nombre, valor);
```

Podemos también eliminar un atributo:

```
context.removeAttribute(nombre);
```

Lo cual dejará el atributo a `null`, igual que si nunca le hubiesemos asignado un valor. Por último, con

```
Enumeration enum = context.getAttributeNames();
```

Obtenemos la lista de nombres de atributos definidos en el contexto.

Hay que hacer notar en este punto, que el objeto de contexto a parte de ser propio de cada aplicación web, es propio de cada máquina virtual Java. Cuando trabajemos en un contexto distribuido, cada máquina ejecutará una VM distinta, por lo que tendrán también objetos de contexto diferentes. Esto hará que si los servlets de una aplicación se alojan en máquinas distintas, tendrán contextos distintos y este objeto ya no nos servirá para comunicarnos entre ellos. Veremos más adelante formas alternativas de comunicación para estos casos.

Parámetros de inicialización

El objeto `ServletConfig` nos proporcionaba acceso a los parámetros de inicialización del servlet en el que nos encontramos. Con `ServletContext`, tendremos acceso a los parámetros de inicialización globales de nuestra aplicación web. Los métodos para obtener dichos parámetros son análogos a los que usabamos en `ServletConfig`:

```
String valor_param = context.getInitParameter(nombre_param);
Enumeration nombres_params = context.getInitParameterNames();
```

Acceso a recursos estáticos

Este objeto nos permite además acceder a recursos estáticos alojados en nuestro sitio web. Utilizaremos los métodos:

```
URL url = context.getResource(nombre_recurso);
InputStream in = context.getResourceAsStream(nombre_recurso);
```

El nombre del recurso que proporcionamos será una cadena que comience por `"/"`, lo cual indica el directorio raiz dentro del contexto de nuestra aplicación, por lo tanto serán direcciones relativas a la ruta de nuestra

aplicación web.

El primer método nos devuelve la URL de dicho recurso, mientras que el segundo nos devuelve directamente un flujo de entrada para leer dicho recurso.

Hay que señalar que esto nos permitirá leer cualquier recurso de nuestra aplicación como estático. Es decir, si proporcionamos como recurso "`/index.jsp`", lo que hará será leer el código fuente del JSP, no se obtendrá la salida procesada que genera dicho JSP.

Podemos también obtener una lista de recursos de nuestra aplicación web, con:

```
Set recursos = context.getResourcePaths(String ruta);
```

Nos devolverá el conjunto de todos los recursos que haya en la ruta indicada (relativa al contexto de la aplicación), o en cualquier subdirectorio de ella.

Redirecciones

Si lo que queremos es acceder a recursos dinámicos, el método anterior no nos sirve. Para ello utilizaremos estas redirecciones. Utilizaremos el objeto `RequestDispatcher` que nos proporciona `ServletContext`.

Hemos de distinguir estas redirecciones de la que se producen cuando ejecutamos

```
response.sendRedirect();
```

Con `sendRedirect` lo que estamos haciendo es devolver al cliente una respuesta de redirección. Es decir, será el cliente, quien tras recibir esta respuesta solicite la página a la que debe redirigirse.

Con `RequestDispatcher` es el servidor internamente quien solicita el recurso al que nos redirigimos, y devuelve la salida generada por éste al cliente, pero todo ello de forma transparente al cliente. En cliente no sabrá en ningún momento que se ha producido una redirección.

Para obtener un objeto `RequestDispatcher` podemos usar los siguientes métodos de `ServletContext`:

```
RequestDispatcher rd = context.getRequestDispatcher(ruta);
RequestDispatcher rd = context.getNamedDispatcher(ruta);
```

Como ruta proporcionaremos la ruta relativa al contexto de nuestra aplicación, comenzando por el carácter "`/`", del recurso al que nos queramos redirigir. También podemos obtener este objeto proporcionando una ruta relativa respecto al recurso actual, utilizando para ello el método `getRequestDispatcher` del objeto `ServletRequest`, en lugar de `ServletContext`:

```
RequestDispatcher rd = request.getRequestDispatcher(ruta);
```

Podemos utilizar el `RequestDispatcher` de dos formas distintas: llamando a su método `include` o a `forward`.

```
rd.include(request, response);
```

El método `include` incluirá el contenido generado por el recurso al que redireccionamos en la respuesta, permitiendo que se escriba este contenido en el objeto `ServletResponse` a continuación de lo que se haya escrito ya por parte de nuestro servlet. Se podrá llamar a este método en cualquier momento. Lo que no podrá hacer el recurso al que redireccionamos es cambiar las cabeceras de la respuesta, ya que lo único que estamos haciendo es incluir contenido en ella. Cualquier intento de cambiar cabeceras en la llamada a include será ignorado.

Si hemos realizado la redirección utilizando un método `getRequestDispatcher` (no mediante `getNamedDispatcher`), en la petición del servlet al que redireccionamos podremos acceder a los siguientes atributos:

```
javax.servlet.include.request_uri
javax.servlet.include.context_path
javax.servlet.include.servlet_path
javax.servlet.include.path_info
javax.servlet.include.query_string
```

Con los que podrá consultar la ruta desde donde fué invocado.

```
rd.forward(request, response);
```

El método `forward` sólo podrá ser invocado cuando todavía no se ha escrito nada en la respuesta del servlet. Esto es así porque esta llamada devolverá únicamente la salida del objeto al que nos redirigimos. Si esto no fuese así, se produciría una excepción `IllegalStateException`. Una vez el método `forward` haya devuelto el control, la salida ya habrá sido escrita completamente en la respuesta.

Si el recurso al que redireccionamos utiliza direcciones relativas, estás direcciones se considerarán relativas al servlet que ha hecho la redirección, por lo que si se encuentran en rutas distintas se producirá un error. Tenemos que hacer que las direcciones sean relativas a la raiz del servidor para que funcione correctamente (direcciones que comiencen por `"/"`).

Otros métodos

La clase `ServletContext` nos proporciona otros métodos de utilidad, que podremos consultar accediendo a su documentación JavaDoc.

Un método de interés es `log`, que nos permite escribir texto en el fichero de log del servlet:

```
context.log(mensaje);
```

Esto será util para tener un registro de eventos que ocurren en nuestra web, o bien para depurar errores.

Listeners de contexto

Existen objetos que permanecen a la escucha de los distintos eventos que pueden ocurrir en el objeto de contexto de servlets, `ServletContext`.

Un primer listener, es el `ServletContextListener`, que nos permitirá dar respuesta a los eventos de creación y destrucción del contexto del servlet. El código para este listener será como sigue a continuación:

```
import javax.servlet.*;

public class MiContextListener implements ServletContextListener {

    public void contextDestroyed(ServletContextEvent sce) {
        // Destruccion del contexto
    }

    public void contextInitialized(ServletContextEvent sce) {
        // Inicialización del contexto
    }
}
```

Esto nos será de gran utilidad si necesitamos inicializar ciertas estructuras de datos que van a utilizar varios servlets. De esta forma el contexto se habrá inicializado antes de que los servlets puedan ejecutarse.

Si lo que queremos es saber cuando se ha añadido, eliminado, o modificado alguno de los atributos del contexto global, podemos utilizar un listener `ServletContextAttributeListener`. Los métodos que deberemos definir en este caso son los siguientes:

```
import javax.servlet.*;

public class MiContextAttributeListener
                    implements ServletContextAttributeListener {

    public void attributeAdded(ServletContextAttributeEvent scae) {
        // Se ha añadido un nuevo atributo
    }

    public void attributeRemoved(ServletContextAttributeEvent scae) {
        // Se ha eliminado un atributo
    }

    public void attributeReplaced(ServletContextAttributeEvent scae) {
        // Un atributo ha cambiado de valor
    }
}
```

Para hacer que estos objetos se registren como listeners y permanezcan a la escucha, deberemos registrarlos como tales en el descriptor de despliegue de la aplicación. Deberemos añadir un elemento <*listener*> para cada objeto listener que queramos registrar:

```
<listener>
    <listener-class>MiContextListener</listener-class>
</listener>

<listener>
    <listener-class>MiContextAttributeListener</listener-class>
</listener>
```

Otros métodos de comunicación entre servlets

Como hemos dicho anteriormente, el contexto de los servlets tiene el inconveniente de que se creará uno por cada VM. Por lo tanto, si el contenedor de los servlets se encuentra distribuido en varias máquinas, y los tenemos alojados en distintas VM, accederán a contextos distintos, aunque pertenezcan a una misma aplicación web.

Por lo tanto, en estos casos el contexto no nos servirá para comunicar todos los servlets de la aplicación web. Si queremos hacer una comunicación totalmente global, tendremos que utilizar otros métodos, como almacenar los datos en una base de datos, en sesiones, o bien en Enterprise Java Beans (EJB).

 Last Published: 07/30/2010 07:34:22