Comunicación con clientes ricos y AJAX

Índice

1. Servlets y clientes ricos

Vamos a ver a continuación cómo establecer una comunicación entre clientes ricos y *servlets* utilizando el protocolo HTTP. Este protocolo es la interfaz que nos ofrecen los servidores para comunicarnos con los componentes web que contengan, por lo tanto podremos utilizarlo desde un cliente rico Java para comunicarnos con los *servlets* o JSP alojados en el servidor. Además tiene la ventaja de que al ser un protocolo web estándar nos va a permitir realizar las comunicaciones sin problemas, ya que los firewalls intermedios normalmente dejarán pasar las conexiones de este tipo. Aunque los navegadores web normalmente utilizan este protocolo simplemente para obtener contenido HTML, a continuación veremos que podemos utilizarlo para intercambiar cualquier tipo de contenido.

1.1. HTTP Tunneling

Si queremos establecer una comunicación entre un cliente rico y un *servlet*, deberemos establecer desde el cliente una conexión con la URL en la que está publicado el *servlet*. La información se intercambiará a través de la conexión HTTP con dicha URL, y esto es lo que se denomina *HTTP tunneling*. Lo que haremos será construir en el cliente un objeto `URL` con la URL del servlet que queramos utilizar, y abrir una conexión con dicha URL para realizar la petición y recibir el resultado generado por el servlet. Al realizar la conexión en conveniente indicar que no utilice la caché, para que siempre se solicite al servidor que genere nuevamente el contenido:

```
URLConnection con = url.openConnection();
con.setUseCaches(false);
```

Ahora podemos establecer las cabeceras de la petición que queramos. De esta forma podemos proporcionar información sencilla mediante pares *(clave, valor)*:

```
con.setRequestProperty("header", "valor");
```

Entonces ya podemos crear la conexión con la URL:

```
InputStream in = con.getInputStream();
```

Es en este momento cuando se establece la conexión con el servlet y éste genera el resultado. Mediante el flujo de entrada que obtenemos podremos leer desde nuestro applet el contenido generado. Este flujo de entrada podrá ser de cualquier tipo, según el tipo de contenido generado por el servlet. Podemos leer directamente la secuencia de bytes, transformarlo a un flujo de carácteres, o bien a un flujo de procesamiento más complejo (por ejemplo `DataInputStream`).

1.2. Paso de objetos

También podremos hacer que el servlet nos devuelva objetos. Si el servlet serializa un objeto y lo escribe en la respuesta (utilizando un `ObjectOutputStream`), desde el cliente podremos utilizar un `ObjectInputStream` para leer dicho objeto:

```
ObjectInputStream ois = new ObjectInputStream(in);
MiClase obj = (MiClase)ois.readObject();
```

Por último, una vez leido todo el contenido, cerraremos el flujo de entrada:

```
in.close();
```

Para que el servlet devuelva un objeto deberá especificar como tipo del contenido generado el siguiente tipo MIME:

```
application/x-java-serialized-object
```

Y lo único que tendrá que hace entonces será obtener un `ObjectOutputStream` a partir del flujo de salida de la respuesta, y escribir el objeto en él:

```
MiClase result = generaObjetoResultante();
response.setContentType("application/x-java-serialized-object");
ObjectOutputStream oos = new
ObjectOutputStream(response.getOutputStream());
oos.writeObject(result);
oos.flush();
```

Hemos visto que el applet podrá realizar una petición al servlet, y como resultado podrá devolvernos cualquier tipo de datos, pudiendo incluso enviarnos un objeto. A continuación veremos que también es posible enviar datos desde el cliente al servidor utilizando esta petición HTTP.

1.3. Envío de datos en la petición

Para poder enviar información al servidor podremos utilizar el bloque de contenido del mensajes de petición HTTP. Para ello la conexión que abrimos con la URL debe ser de lectura/escritura:

```
URLConnection con = url.openConnection();
con.setUseCaches(false);
con.setDoOutput(true);
```

A continuación crearemos un array de bytes donde escribiremos la información que enviaremos al servidor:

```
ByteArrayOutputStream baos = new ByteArrayOutputStream();
```

Podemos usar distintos tipos de flujos sobre este objeto, según vayamos a escribir carácteres (`PrintWriter`), datos codificados en binario (`DataOutputStream`) u objetos serializados (`ObjectOutputStream`), por ejemplo.

Una vez escrito este contenido, deberemos establecer una cabecera con la longitud de dicho contenido:

```
con.setRequestProperty("Content-Length", String.valueOf(baos.size()));
```

Por cuestión de compatibilidad entre navegadores, será conveniente establecer manualmente el tipo de contenido:

```
connection.setRequestProperty("Content-Type",
                              "application/x-java-serialized-object");
```

Tipos MIME que podemos utilizar comunmente para intercambiar información son:

`application/x-www-form-urlencoded`	Se envían los datos codificados de la misma forma en la que son codificados por un formulario HTML con método POST.
`text/plain`	Se envía como contenido texto ASCII.
`application/octet-stream`	Se envía como contenido datos binarios. Dentro de la secuencia de bytes podremos codificar la información como queramos. Por ejemplo, podemos codificar de forma binaria un objeto serializado, utilizando un `DataOutputStream`.
`application/x-java-serialized-object`	Se envía como contenido un objeto Java serializado.

En este momento ya podremos enviar los datos al flujo de salida de la conexión:

```
baos.writeTo(connection.getOutputStream());
```

Una vez hecho esto ya podemos obtener el flujo de entrada de la conexión, para realizar la petición y obtener contenido que genera el servlet como respuesta, de la misma forma que lo haciamos en el caso anterior.

Nota:

Si no abrimos el flujo de entrada (ni intentamos consultar ninguna otra propiedad de las respuesta que genera el *servlet*), la petición no se llegará a realizar y por lo tanto el *servlet* no recibirá los datos que hayamos escrito para ser enviados)

2. Servlets y Javascript: AJAX

AJAX (*Asynchronous Javascript And Xml*) no es una tecnología, sino una técnica de desarrollo que combina una serie de tecnologías para hacer que un navegador en el lado del cliente pueda obtener información del servidor sin tener que recargar la página que se esté mostrando. Las tecnologías que se utilizan son:

- **HTML** y **CSS** para presentar la información.
- **XML** para intercambiar información con el servidor.
- **DOM** y **Javascript** para analizar esta información.

Esta técnica consistirá en hacer una petición HTTP utilizando el objeto `XMLHttpRequest` de *Javascript*. Con esto obtendremos del servidor un documento XML con la información requerida, que podremos analizar con el DOM (también de *Javascript*) Una vez tengamos los datos obtenidos del servidor, podemos actualizarlos en el documento HTML utilizando nuevamente el DOM.

2.1. Petición HTTP desde Javascript

Lo primero que deberemos hacer es obtener el objeto `XMLHttpRequest` que nos permita hacer una petición HTTP desde *Javascript*. Deberemos distinguir entre IE y Firefox:

```
function verMensajes() {
   //Preparar objeto para lanzar peticion
   if (window.XMLHttpRequest) {         //Firefox,etc
      peticion = new XMLHttpRequest();
   } else if (window.ActiveXObject) {   //Explorer
      peticion = new ActiveXObject("Microsoft.XMLHTTP");
   }
```

Una vez obtenido este objeto, deberemos especificar a qué función se deberá llamar cuando se reciba la respuesta HTTP del servidor. Especificaremos esta función *callback* de la siguiente forma:

```
   //a quien llamar cuando el servidor responda
   peticion.onreadystatechange = atenderPeticion;
```

Por último, efectuaremos la petición propiamente dicha, especificando el tipo de petición, la URL a la que vamos a conectar, y si la llamada es asíncrona. Si no fuese asíncrona, el programa quedaría bloqueado hasta recibir la respuesta. Por lo tanto, deberemos poner este último parámetro siempre a `true` para evitar que la web queda bloqueada mientras se obtiene la información:

```
   //lanzar la peticion propiamente dicha
   peticion.open("GET",
"http://localhost:7001/ChatXml/chat/listaMensajesXml.jsp", true);
   peticion.send(null);
}
```

NOTA: Por motivo de seguridad, sólo se permitirá hacer peticiones mediante el objeto `XMLHttpRequest` al servidor del que se ha descargado esta página.

Una vez se reciba la respuesta, se llamará a la función especificada como *callback*. Deberemos fijarnos en el estado en el que se encuentra la petición:

`0`	No inicializada
`1`	Cargando
`2`	Cargada

3	Interactiva
4	Completada

El significado de cada uno de estos estados es:

- **No inicializada.** Se ha creado el objeto de la petición HTTP pero ésta todavía no se ha efectuado.
- **Cargando.** Se ha realizado la petición HTTP y se está esperando a recibir la respuesta.
- **Cargada.** La respuesta HTTP se ha recibido, pero todavía no se puede acceder al modelo DOM que representa al documento obtenido.
- **Interactiva.** El modelo DOM se ha empezado a construir pero todavía no está completo. Podremos consultar las partes del modelo que ya se hayan construido.
- **Completada.** El modelo DOM del documento obtenido está completo.

Sólo nos interesará el caso en el que la petición haya sido completada. Además también deberemos comprobar que la respuesta del servidor no nos haya devuelto un código de error. Si todo ha ido bien, el estado de la respuesta debería ser `200 OK`:

```
function atenderPeticion() {
   if (peticion.readyState == 4) {
      //analizar respuesta
      if (peticion.status!=200) {
         alert("ha habido un error");
         return;
      }
```

En este punto tendremos el contenido de la respuesta en las propiedades `peticion.responseText` y `peticion.responseXML`. La primera de ellas nos ofrecerá la respuesta como una cadena de texto, mientras que la segunda es un objeto del tipo `XMLDocument` al que podremos acceder mediante el DOM.

2.2. Lectura de la respuesta

Normalmente utilizaremos `responseText` cuando queramos incluir directamente en el HTML el contenido devuelto por el servidor, y `responseXML` cuando estemos intercambiando estructuras de datos.

A continuación se muestra un ejemplo en el que se analiza el documento XML obtenido para incluir la información formateada en el HTML:

```
      //mostrar mensajes
      var areaMensajes = document.getElementById("mensajesChat");
          var textoHTML = "";

          docXml = peticion.responseXML;
          var raiz = docXml.getElementsByTagName('mensajes');
          mensajes = raiz[0].getElementsByTagName('mensaje');
```

```
          for(i=0;i<mensajes.length;i++) {
                    var nick =
mensajes[i].getElementsByTagName('nick').item(0).firstChild.data;
                    var texto =
mensajes[i].getElementsByTagName('texto').item(0).firstChild.data;
                    textoHTML += "<strong>&lt;" + nick + "&gt;</strong> " +
texto + "<br/>";
          }

      areaMensajes.innerHTML = textoHTML;
   }
}
```

En el caso de este ejemplo, para intercambiar la información estaríamos utilizando un documento XML con el siguiente formato:

```
<mensajes>
<mensaje>
  <nick>Ana</nick>
  <texto>Hola</texto>
</mensaje>
<mensaje>
  <nick>Jose</nick>
  <texto>Que tal?</texto>
</mensaje>
</mensajes>
```

Podremos invocar la función que efectua la petición HTTP de diferentes formas. Podemos utilizar un temporizador para hacer que la información se actualice periódicamente, o bien podemos hacer que se invoque como respuesta a algún evento del usuario (por ejemplo cuando pulse un botón).

2.3. Aplicaciones de AJAX

Esta técnica tiene numerosas aplicaciones, como por ejemplo:

- *Validación de formularios*: Podremos validar la información introducida en un formulario en el lado del servidor antes de enviarlo. Por ejemplo, al registrarnos en una web podemos comprobar si un *login* ya está utilizado.
- *Autocompletar campos*: Mientras escribimos en un campo de un formulario, podemos consultar en el servidor las coincidencias con lo que llevamos escrito para que nos muestre una lista con las posibilidades existentes.
- *Mostrar datos actualizados*: Podemos mantener información de última hora en la web sin tener que recargar. Por ejemplo, podemos ver el marcador de un partido de fútbol que se está jugando en este momento, la clasificación de la Fórmula 1, o información de la bolsa.