Filtro y ordenación de resultset de DatabaseMetaData

El número máximo de registros de un resultset representados es el que corresponde con el atributo *ValoresPorDefecto.numMaxRecords*. Si se procesan más de esta cantidad de registros se genera la excepción *ResultSetException* y no se presenta ninguno, ni siquiera los primeros 1000 registros y en su lugar se presenta el texto correspondiente del error.

Como pueden existir fácilmente una gran cantidad de funciones o consultas que devuelvan una cantidad superior de registros, se hace necesario un filtrado de datos que, mediante la aplicación de uno o varios criterios, reduzca el número de registros del resultset.

En los resultset proporcionados por las funciones de *DatabaseMetaData*, el filtrado se realiza mediante los argumentos de llamada de la función. Existen funciones que no precisan argumentos y otras que sí los precisan. Entre estas, cuando a un argumento se le asigna el valor null, dicho concepto no entra en el filtrado del resultset resultante. Por ejemplo, en la función *getTables()* uno de los argumentos es *schemaPattern*. Si en la llamada se le asigna el valor null a este parámetro, la función no utilizará el filtro del nombre del esquema en el resultado de la misma.

Como se ha comentado antes, el número y nombre de los argumentos varía de una función a otra, por lo que se tiene que diseñar un sistema flexible que pueda adaptarse a estos requisitos. Además hay que tener en cuenta que el tipo de datos de los argumentos también cambia de unos argumentos a otros y existen argumentos de tipo *String*, de tipo *int*, de tipo *boolean* y de tipo *String[]*.

En cuanto a la ordenación, se pretende que el usuario pueda ordenar la tabla html obtenida por cualquiera de las columnas que la conforman tanto en orden ascendente como descendente. Es una operación que se desarrolla integramente en el cliente mediante tecnología javascript para lo que se utiliza jquery y propiedades css. Al ser un código de ejecución en el cliente, el punto de entrada será siempre el jsp.

# DatabaseMetaDataAction.java

Paquete es.ubu.alu.mydatabasejc.actions

Los tipos de datos de una función de DatabaseMetaData son pasado en la request bajo el parámetro *parametros* codificado en *Base64* como vimos en capítulos anteriores. Estos datos son recogidos en la clase action y convertidos al tipo Class[] mediante la ejecución de la siguiente instrucción como ya se explicó en el capítulo XXX:

Class[] parameterTypes = getParameterTypes(parametros);

A continuación se debe llamar al método que toma los valores de los argumentos recibidos en la llamada y los guarda en la sesión:

setParametrosSesion(parameterTypes);

porque, finalmente, la función que ejecuta el método correspondiente tomará de la sesión del usuario el valor del argumento previamente guardado. De esta manera, conseguimos que los parámetros iguales sean mantenidos en llamadas a funciones diferentes siempre que utilicen el mismo nombre.

Los siguientes métodos trabajan en este sentido:

* void setParametrosSesion(Class[] parameterTypes): Es un método privado que recibe los tipos de los argumentos que debe analizar y, si existen, los guarda en la sesión del usuario. Junto con el tipo de los parámetros, la función debe obtener tanto los nombres como los valores de los argumentos de dos atributos de tipo String[] que son pasado, o no, en la request, junto con el nombre y los tipos de los argumentos a invocar. Si no se mandan estos datos, la función no realiza ningún trabajo:

if (filtroArgumentos == null || filtroValores == null) return;

Para cada tipo de dato requerido de la función se localiza el nombre del argumento correspondiente:

for (int i = 0; i < parameterTypes.length; i++) {

String atributo = filtroArgumentos[i];

Si el valor del argumento es distinto de la cadena vacía:

if (!"".equals(filtroValores[i]))

Se asigna a una variable de tipo Object el valor correspondiente casteado convenientemente. Para poder castear adecuadamente el valor que llega en formato String al tipo de dato requerido se utiliza una estructura switch:

switch (parameterTypes[i].getName()) {

Cuando el tipo de datos es un array de Strings se crea un array de Strings utilizando la coma como separador de elementos y deshechando los elementos vacíos:

case "[Ljava.lang.String;":

String[] valores = filtroValores[i].split(",",0);

List<String> lvalores = new ArrayList();

for (int j = 0; j< valores.length; j++)

if (!"".equals(valores[j])) lvalores.add(valores[j].trim());

valor = lvalores.toArray(valores);

break;

Para otros tipos de datos simples, se utiliza la clase correspondiente para hacer el casteado desde String. Por ejemplo, para tipo de dato boolean:

case "boolean": valor = Boolean.valueOf(filtroValores[i]); break;

En caso contrario, cuando el tipo sea String o desconocido, se asigna directamente el valor correspondiente:

default: valor = filtroValores[i];

Finalmente, se añade el valor a la sesión del usuario:

sesion.put(atributo, valor);

* **List getArrayParametros()**: Esta clase es utilizada en la página jsp para generar el array de parámetros que se deben incluir en la invocación del método. Una vez que se ha determinado el método, se puede conocer el array de parámetros que se deben incluir en su invocación mediante el siguiente código:

method.getParameters();

que se almacena en la variable global:

Parameter[] arrayParametros

El método, transforma el array en formato List para su mejor tratamiento en un iterador struts:

List camposFiltro = new ArrayList();

for (Parameter p : arrayParametros) {

camposFiltro.add(p.getName());

}

return camposFiltro;

* **Object getParameter(int i)**: Este método devuelve desde la sesión del usuario el valor almacenado para el parámetro de posición *i*:

return sesion.get(arrayParametros[i].getName());

# Filtro.jsp

Carpeta /WEB-INF/jspf

Esta página visualiza los campos que conforman el filtro para la ejecución del correspondiente método. Dentro de un bloque form, puesto que estos datos serán enviados en una request:

<s:form name="filtro" method="POST" theme="simple">

Lo primero es incluir tanto el método, como los tipos de los argumentos que requiere. Se oculta puesto que información irrelevante para el usuario:

<s:hidden name="metodo"/>

<s:hidden name="parametros"/>

y mediante la utilización de una lista, se recorre el iterador *arrayParametros*. Se define una variable counter que nos permitirá llevar un contador del argumento en el que nos encontramos:

<ul>

<s:set var="counter" value="0"/>

<s:iterator value="arrayParametros">

…

</s:iterator>

</ul>

Dentro del iterador, cada elemento de la lista contiene la etiqueta del nombre del argumento correspondientemente idiomatizada, un campo oculto (*filtroArgumentos*) con el nombre del argumento (sin idiomatizar) y un campo visible (*filtroValores*) con el valor del argumento obtenido mediante la función *getParameter*:

<li>

<s:set var="filtro"><s:property /></s:set>

<s:text name="%{filtro}" />:

<s:hidden name="filtroArgumentos" value="%{filtro}"/>

<s:textfield name="filtroValores" value="%{getParameter(#counter)}"/>

</li>

Se completa el iterador con el incremento del contador

<s:set var="counter" value="%{#counter+1}"/>

y se completa el formulario con un botón de submit

<s:submit name="filtrar"/>

# DatabaseMetaData.jsp

carpeta /

El código que a continuación se detalla ha sido adaptado del indicado por Wanderson López en la web http://wanderlp.com/tabla-con-ordenamiento-utilizando-css-y-jquery/

El fichero DatabaseMetaData.jsp es modificado para incluir referencias que permitan realizar la ordenación de la tabla html que en él se representa.

Para que la ordenación pueda llevarse a cabo, la tabla html debe estar correctamente construida lo que lleva a diferenciar la zona de cabecera de la zona de cuerpo mediante los tags *<thead>* y *<tbody>* y los tags *<th>* y *<td>*

Para ello debemos modificar la página añadiendo el siguiente código:

<s:iterator value="listInfo" var="record" status="status">

<s:if test="#status.first == true"><thead></s:if>

…

<s:if test="#status.first == true"></thead></s:if>

</s:iterator>

que marca la zona de cabecera solo en el primer registro del iterador

Se debe incluir la referencia a la biblioteca jquery utilizada para realizar la ordenación mediante el siguiente código en el bloque <head>:

<script src="${pageContext.request.contextPath}/resources/jquery-3.3.1.min.js"></script>

Con la utilización de *pageContext.request.contextPath* conseguimos que se localice el javascript en la carpeta correcta independientemente de la ubicación del fichero jsp.

El método de ordenación de la tabla consiste en tres fases:

1. Enviar el contenido de la tabla a un arreglo
2. Ordenar el arreglo
3. Mostrar en el encabezado de la tabla cual es el orden aplicado

## Ordenar tabla

Este paso debe realizarse en respuesta a una acción clic en algún elemento de la cabecera de la tabla. Para ello, programamos una función que responda a un evento clic sobre el objeto th:

$('th').click(function() {

...

});

Carga la variable table con el objeto html padre table y la tabla rows con las filas mayores que 0 y ordenadas

var table = $(this).parents('table').eq(0)

var rows = table.find('tr:gt(0)').toArray().sort(comparer($(this).index()))

La función comparer permite comparar los valores de la tabla entre sí:

function comparer(index) {

return function(a, b) {

var valA = getCellValue(a, index),

valB = getCellValue(b, index)

return $.isNumeric(valA) && $.isNumeric(valB) ? valA - valB : valA.localeCompare(valB)

}

}

para lo que utiliza la función getCellValue que obtiene los valores de la celda:

function getCellValue(row, index) {

return $(row).children('td').eq(index).html()

}

En la función original del evento click, una variable mantiene el tipo de ordenación establecido:

this.asc = !this.asc

if (!this.asc) {

rows = rows.reverse()

}

Finalmente añade a la tabla las filas en el orden correcto

for (var i = 0; i < rows.length; i++) {

table.append(rows[i])

}

y con la siguiente llamada visualiza el icono que le indicará al usuario el campo y el tipo de ordenación establecido:

setIcon($(this), this.asc);

Esta función muestra gráficamente qué ordenamiento se está aplicando asignando o desasignando clases css a los distintos componentes th de la cabecera de la tabla.

function setIcon(element, asc) {

$("th").each(function(index) {

$(this).removeClass("sorting");

$(this).removeClass("asc");

$(this).removeClass("desc");

});

element.addClass("sorting");

if (asc) element.addClass("asc");

else element.addClass("desc");

}

## Mostrar cuál es el orden aplicado

Para conseguir este efecto se recurre a la utilización de CSS.

En primer lugar el cursor debe ser un puntero cuando apunte a la cabecera de la tabla. Si no, aparecerá un cursor:

table tr th {

cursor: pointer;

-webkit-user-select: none;

-moz-user-select: none;

-ms-user-select: none;

user-select: none;

}

La cabecera de la columna utilizada para la ordenación se diferencia del resto mediante un color de fondo diferente:

.sorting {

background-color: #D4D4D4;

}

y se incluirá una flecha hacia arriba o hacia abajo para indicar el tipo de ordenación ascendente o descendente:

.asc:after {

content: ' ↑';

}

.desc:after {

content: " ↓";

}

Todo esto se añade en la sección <head> mediante tags <style>