# Presentación de DatabaseMetaData

Una de las funcionalidades más habituales consiste en representar en pantalla un ResultSet obtenido mediante la llamada a un método de la clase DatabaseMetaDataImpl. Un gran número de funciones de esta clase devuelve una estructura de datos de este tipo. En este capítulo comentamos los detalles de la aplicación en la llamada y presentación de este tipo de datos.

## inicial.jsp

Una de las páginas típicas que realiza llamadas de este tipo es la pantalla Inicial.jsp:

<ol>

<s:iterator value="menus">

<li>

<s:url action="%{action}" var="urlTag">

<s:param name="metodo"><s:property value="metodo"/></s:param>

<s:param name="parametros"><s:property value="parametros"/></s:param>

</s:url>

<s:a href="%{urlTag}"><s:property value="%{getText(metodo)}"/></s:a>

</li>

</s:iterator>

</ol>

La lista de opciones de menú representada mediante una lista html consiste en una colección de links cuya acción es *resultset o List<List>* y se acompaña de un parámetro en la llamada con el nombre del método que se ha de ejecutar y a resultas de lo que se recibirá un resultset o un List<List>

## struts.xml

Se mapea la acción *resultset* hacia el correspondiente método de la clase *DatabaseMetaDataAction*

<action name="resultset" class="es.ubu.alu.mydatabasejc.actions.DatabaseMetaDataAction" method="resultset">

<result>/DatabaseMetaData.jsp</result>

</action>

Una vez finalizada la acción se presentará la pantalla resultset.jsp

Igualmente, se mapea la acción “info”:

<action name="info" class="es.ubu.alu.mydatabasejc.actions.DatabaseMetaDataAction" method="info">

<result>/DatabaseMetaData.jsp</result>

</action>

## DatabaseMetaData.jsp

Carpeta /

La página *DatabaseMetaData.jsp* es básicamente idéntica a la página *inicial.jsp* con la diferencia de que la parte principal de la página debe visualizar la información obtenida en el *List<List>*.

Comenzamos construyendo una tabla html con iterador para cada fila:

<table class="tabla">

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

<tr>

...

</tr>

</s:iterator>

</table>

Para cada fila, se define una variable *cuenta* que permitirá dar un formato especial a la primera columna:

<s:set var="cuenta" value="1"/>

A continuación, se construye un iterador para cada columna dentro de cada fila:

<s:iterator value="#record">

...

</s:iterator>

Dentro del iterador, se crea una variable *propiedad* que contendrá el valor de cada dato:

<s:set name="propiedad"><s:property /></s:set>

Además, si se trata de la primera fila, se utiliza el tag **th** en vez del tag **td** y se presenta el dato

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

<th><s:property /></th>

</s:if>

<s:else>

...

</s:else>

Si no es la primera fila, se pasará a presentar el dato bajo el tag **td**, pero si se trata de la primera columna, el valor es internacionalizado usando el tag de struts **s:text** y la variable propiedad establecida anteriormente con el valor del campo:

<td>

<s:if test="#cuenta == 1">

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

se establece la variable cuenta a 2 (distinto de 1) para que solo la primera columna sea internacionalizada evitando de esta manera sobrecarga innecesaria.

<s:set var="cuenta" value="2"/>

En el resto de columnas, se presenta el dato sin internacionalizar:

</s:if>

<s:else>

<s:property />

</s:else>

</td>

## DatabaseMetaDataAction.java

Paquete es.ubu.alu.mydatabasejc.actions

El método *resultset* es el encargado de tratar la petición del cliente, recibir el método de la clase *DatabaseMetaDataImpl* que se debe ejecutar en la propiedad *metodo* y que devuelve el objeto *resultset* que, una vez tratado, será enviado a la página jsp correspondiente que lo presentará en forma de tabla html.

Métodos:

* **resultset()**: Es el método que, en función del método recibido en la propiedad *metodo*, llama al correspondiente método de la clase *DatabaseMetaDataImpl* obteniendo el resultset correspondiente y lo transforma y asigna a la lista *listResultSet* que es la que finalmente se visualizará. La transformación del resultset a la lista se realiza con la siguiente llamada

listInfo = getListInfo(rs);

Para realizar la llamada al método se utiliza java Reflection y el método invoke acompañándole los parámetros necesarios para su ejecución. Puesto que es proceso complejo, se aísla en una método privado al que se llama con la siguiente línea:

ResultSet rs = getInvoke(connectionImpl.getConnection().getMetaData(), metodo, parametro);

* **info()**: Este método invoca al método correspondiente que devolverá ya el objeto List<List> capaz de ser representado por la página web
* **private getListInfo(ResultSet resultSet)**: Este método privado recibe un *ResultSet* y lo transforma en una lista de listas de objetos. Cada lista, a primer nivel, representa cada registro del resultset. Cada elemento de la lista interna contiene cada campo del registro. El primer elemento de la lista contiene los nombres de los campos del resultset.

int i = 0;

List cabecera = new ArrayList();

for (i = 1; i <= rsMetadata.getColumnCount(); i++) {

cabecera.add(rsMetadata.getColumnName(i));

}

lista.add(cabecera);

En el bucle que recorre el recordset, si se llega al máximo de registros a devolver indicado en la clase ValoresPorDefecto, se manda un error:

int n = 1;

while (resultSet.next()) {

// si se supera el máximo de registros, se manda error

if (n++>ValoresPorDefecto.numMaxRecords)

throw new ResultSetException("Demasiados.registros.Filtrar", resultSet);

List record = new ArrayList();

for (int j = 1; j < i; j++)

record.add(resultSet.getObject(j));

lista.add(record);

}

* **private getInvoke(DatabaseMetaData, metodo, parametros)**: invoca al método de la clase *DatabaseMetaData* indicada. Retorna el *ResultSet* obtenido por dicha invocación.

En primer lugar decodifica los tipos de parametros que debe recibir el método y que están codificados en Base64 y recibidos en el parámetro *parametro*:

Class[] parameterTypes = getParameterTypes(parametros);

Una vez que se tienen los tipos de parámetros en forma de array y el nombre del método a invocar, se obtiene de *DatabaseMetaDataImpl*:

Method method = DatabaseMetaDataImpl.class.getMethod(metodo, parameterTypes);

A continuación se obtiene el array de parámetros del método, no sus tipos, sino sus nombres:

Parameter[] arrayParametros = method.getParameters();

y se crea una lista de objetos que serán finalmente los parámetros a pasar al método invocado

List<Object> listaParametros = new ArrayList<>();

se busca cada parámetro en la sesión del usuario y lo asigna a la lista de objetos a pasar

for (Parameter parametro : arrayParametros) {

listaParametros.add(sesion.get(parametro.getName()));

}

Se pasa la lista a la forma array de objetos

Object[] args = listaParametros.toArray();

y finalmente se invoca al método pasándole los argumentos necesarios

Object o = method.invoke(dbMetadata, args);

Ya solo queda retornar el objeto previa comprobación de que es del tipo ResultSet:

if (o instanceof ResultSet)

return (ResultSet)o;

o del tipo List:

if (o instanceof List)

return (List<List>)o;

* **Class[] getParameterTypes(String parametros)**: Este método privado devuelve el array de clases que un método requiere como argumentos a partir del argumento que contiene la información en formato String Base64:

ByteArrayInputStream bais = new ByteArrayInputStream(Base64.decodeBase64(parametros));

ObjectInputStream ois = new ObjectInputStream(bais);

// se obtiene el array de parámetros a partir de los datos obtenidos

return (Class[])ois.readObject();

## DatabaseMetaDataImpl.java

Como se comentó en el capítulo XXXX, existen una serie de métodos en esta tabla que presentan datos informativos del tipo *Propiedad - valor*, en formato *List<List>*, para que su tratamiento en la presentación de datos sea el mismo que el que se hace en los métodos de *DatabaseMetaData* que devuelven un *ResultSet* convertido finalmente a un *List<List>* como se ha tratado en este capítulo.

A continuación, se explica cómo se convierte la colección de datos *Propiedad Valor* en un List<List> en estos métodos.

* **getInfo(Methods[] metodos)**: Va creando una lista de objetos:

List<List> lista = new ArrayList();

y le añade una cabecera genérica:

List cabecera = new ArrayList();

cabecera.add("Propiedad");

cabecera.add("Valor");

lista.add(cabecera);

Por cada método de la clase referenciada, se invoca y se guarda el nombre del método y el valor obtenido en la lista. Los métodos invocados no deben admitir parámetros:

Object o = null;

try {

o = method.invoke(this, new Object[]{});

} catch (Exception e) {

o = e.getLocalizedMessage();

}

List registro = new ArrayList();

registro.add(method.getName());

registro.add(o);

lista.add(registro);

Finalmente retorna la lista de listas:

return lista;

Cada método personalizado de DatabaseMetaDataImpl envía a esta función un conjunto de métodos de ejecución simple (sin parámetros y que retornan objetos de tipo simple). Para conseguir ese conjunto de métodos se utilizan las anotaciones que marcan los métodos correspondientes y la categoría a la que pertenecen. Para ello, se ha creado una anotación personalizada que permite categorizar los métodos. Es la clase MetaDataIntoCategorias que se explica más adelante.

* **getMetodos(String categoria)**: Crea una lista de almacenamiento temporal

List lista = new ArrayList();

Obtiene todos los métodos

Method[] methods = this.getClass().getMethods();

Para cada método obtiene la anotación del tipo adecuado de cada método

MetaDataInfoCategorias anotacion = methods[i].getAnnotation(MetaDataInfoCategorias.class);

si no existe la anotación, pasa al siguiente método

if (anotacion==null) continue;

si existe y es de la categoría buscada, se añade a la lista temporal de métodos

if (anotacion.categoria().equals(categoria))

lista.add(methods[i]);

}

convierte la lista temporal a array y lo retorna

return (Method[])lista.toArray();

* Métodos informativos: Obtienen la lista de métodos marcados con la anotación en función de su categoría y retorna la información de estos métodos en formato resultset:

Method[] methods = getMetodos("Básica");

return getInfo(methods, metadataBasicInfo);

La declaración de uno de estos métodos sería, a modo de ejemplo:

public ResultSet getBasicInfo() {

* Métodos sobreescritos de tipo informativos: Como se comentó, algunos métodos deben ser marcados con una anotación para permitirnos agruparlos bajo un método propio que devuelva sus métodos y ejecuciones en forma de resultset. Estas anotaciones, a modo de ejemplo, serían de la siguiente forma:

@MetaDataInfoCategorias(categoria = "Usuario")

## MetaDataInfoCategorias.java

Paquete es.ubu.alu.mydatabasejc.annotations

Esta clase define el comportamiento de la anotación que usamos para categorizar los métodos de DatabaseMetaData de tipo informativo simple (sin parámetros y devolviendo objetos de tipo simple).

@Target(ElementType.METHOD)

@Retention(RetentionPolicy.RUNTIME)

@Documented

public @interface MetaDataInfoCategorias {

String categoria() default "Básica";

}