Sorting and Filtering in a Data Grid

Table of contents

1. Overview

The NetUI data grid JSP tags support maintaining and displaying UI based on an abstract set of sorts and filters applied to a data set. Sorts and filters are Java objects that can be created manually or can be inferred from state that is encoded in a URL. These objects represent an abstract notion of a sort and filter that are not coupled to any specific query language. As such, they can be used to programmatically sort or filter a data set or to parameterize a query string in a specific query language like SQL, XQuery, EJB-QL, and so on.

The data grid is related to sorting and filtering, but the data grid itself does not actually sort or filter a data set. While this feature could be supported in the future, the goal of sorts and filters are to support loose coupling of the process used to sort and filter while allowing the data grid to track the states of sorts and filters. The data grid supports sorts and filters through setting CSS class names on columns that are sorted and filtered and by supporting UI gestures -- such as clicking on a column header -- to sort and filter data. Because the sort and filter information is exposed from the DataGridState object, the JSP 2.0 expression language can be used to configure the user interface based on the sort, filter, and paged state of a data set displayed in a data grid.

This document describes the structure of sort and filter objects and discusses how they can be created, used to sort a data set, and how they relate to the data grid. A concrete example that demonstrates some of the sort and filter features can be found in a Beehive sample available in *samples/netui-samples/web/ui/datagrid/sortandfilter*. This sample demonstrates the data grid's support for sorting data by clicking on a grid's header cell and for filtering data using an HTML form.

For the sake of concrete descriptions, this documentation applies abstract Sort and Filter objects to SQL in order to make the examples interesting. It is possible to build a mapping from the Sort and Filter objects to any other query language.

2. Sorts

A data grid sort is represented by the Sort class which has several properties:

Property Name	Description
sortExpression	The sort expression is a String that describes the data to be sorted.
sortDirection	The sort direction is an enumeration value that describes the order in which data should be sorted. In the general case, this is one of ASCENDING or DESCENDING.

In more concrete terms, a sort expression of "name" and sort direction of `SortDirection.ASCENDING` could be used to produce a SQL `ORDER BY` fragment like `ORDER BY name ASC`.

2.1. Data grid Support for Sorting

The data grid JSP tags can be used to manipulate the sort state for a data grid in a URL. This allows a URL to explicitly describe the sort appearance of a data grid and makes for easy, transparent bookmarking. A sort is often applied to a *column* in a data grid and can be specified by setting the `sortExpression` attribute for any data grid `headerCell` tag. By default, a sort can be activated by clicking on a column's header in the rendered data grid; this will cycle the sorted state through a series of states from NONE to ASCENDING to DESCENDING. The change in state can be observed by watching the URL. By default, a sort appears in the URL as:

```
netui_sort=<namespace>;(|-)<sortExpression>
```

The namespace is taken from the data grid's name attribute in order to scope a sort to a particular data grid. The sort expression is explicitly in the parameter value; the default sort direction is ASCENDING unless a `-` is present to change the sort direction to DESCENDING.

The list of Sort(s) available on the URL can be read using a `DataGridState` object that parses state information from a query string and can return the list of states. This can be done using the following code:

```
DataGridState dataGridState =
DataGridStateFactory.getInstance(httpServletRequest).getDataGridState("<namespace>");
List sorts = dataGridState.getSortModel().getSorts();
```

Although the data grid may use a sort expression, the data will not be sorted until code executes to actually sort the data. *The data grid does not automatically sort data*. In order to sort data, controller code must be implemented to apply Sort objects to a data grid.

The sort state of a particular sort expression can also be used to configure the styles of a data grid column's header cell and data cells. When a column is sorted, it will render a `sorted` style for both the header and data cells.

2.2. Sorting a Data Set

Sorts must be manually applied to a data set in order to cause data to be sorted in a particular direction. This sorting can be implemented in several ways including converting a Sort object into a query language fragment and letting a query engine sort a data set or manually writing code to sort a data set. To convert Sort(s) into a query language fragment, a converter must be built to produce the fragment from the Sort(s). A simple SQL converter called

SQLSupport can be used for this purpose. A sort of the form *netui_sort=customers;-customerid* will be converted into a SQL fragment of the form `ORDER BY customerid DESC` using the code:

```
List sorts = dataGridState.getSortModel().getSorts();
String sort = SQLSupport.getInstance().createOrderByClause(sorts);
```

A list of Sort(s) can also be used to manually filter a data set, particularly when sorting on a single sort expression (column of data). In this case, a reasonably sized data set can be sorted in-memory quickly using a custom `Comparator` and the `java.util.Collections.sort(...)` method. An example of this is available in the data grid sort / filter sample in the distribution.

2.3. Creating a Sort

Sort objects can also be created programmatically. When creating a Sort object manually, the Sort object should be created from the DataGridConfig object for a data grid. The DataGridConfig object is used as a configuration object that can be used in its default state or can be extended to provide, extend, or change the operation of the data grid. In most cases, the DataGridConfig object can be created with:

```
DataGridConfig dataGridConfig = DataGridConfigFactory.getInstance();
```

And, the DataGridConfig can be used to create a Sort with:

```
Sort sort = dataGridConfig.createSort();
```

Once a Sort is created, it can be configured by setting its JavaBean properties and can be applied to a data set as described here.

3. Filters

A data grid filter is represented by the Filter JavaBea which has several properties:

Property Name	Description
filterExpression	The filter expression is a String that describes a property from the data set to filter.
filterOperation	A query language specific representation of a filter operation. A filter operation may provide an operator that is used when building a query string. For example, some languages may represent equals as '=' or as 'eq'.
filterOperationHint	A query language neutral hint of the type operation to perform. Not all operation hints will be supported for all query languages.

typeHint	A hint provided to describe the type of the filter value to a query engine. This is needed in order to correctly build a query string describing a filter or to correctly filter a value of a particular type. For example, when building a filter expression for a String, the String value may need to be wrapped in quotes to be interpreted by a query engine.
value	The value of an operation that provides a constraint to a fliter. For example, when filtering to a specific integer value, this is the value of that integer.

The filter operation hint can be one of many values that represent filtering options such as:

- EQUAL
- GREATER_THAN
- IS_ONE_OF
- STARTS_WITH
- CONTAINS

As a concrete example, a filter on a value of "companyname" with an operator of "contains" and a value "wheel" with type String could be used to produce a SQL `WHERE` clause like `WHERE companyname LIKE '%wheel%'`.

3.1. Data grid Support for Filtering

The data grid's support for filtering is different than that for sorting; the data grid itself does not provide logic for filtering and instead has the ability to mark a column of data as filtered and leaves the construction of filter UI to the developer. A common pattern for building filter UI is to provide an embedded filter form or a filter pop-up that implements filter logic. A data grid header cell can be linked to a filter expression using the filterExpression attribute on the `headerCell`.

The data grid APIs support reading filter information that is encoded in the URL. One benefit of adding this information to a query string as query parameters is that a filtered data grid can be bookmarked, which can make it easy to return to a specific view into a data set. The default format for filters in the URL is:

```
netui_filter=<namespace>;<filterExpression>~<filterOperation>~<value>
```

The namespace of the filter is taken from the data grid's name attribute, and the filter's expression, operation type, and value are encoded in the remainder of the query parameter. The filters on a URL query string can be read using a `DataGridState` which is used to manage a grid's state from some state source. The filters can be extracted from the query string using the following code:

```
DataGridState dataGridState =
DataGridStateFactory.getInstance(httpServletRequest).getDataGridState("<namespace>");
List filters = dataGridState.getFilterModel().getFilters();
```

Filters can be placed on the URL by JavaScript in the page or by extending the data grid itself to support filtering.

3.2. Creating a Filter

Filter objects can be created programmatically and applied to a data set either manually or by converting Filter{s) into query fragments to be executed by a query engine. A DataGridConfig object can be used get a factory that can provide a `Filter` object for a specific type of data grid. This can usually be created with:

```
DataGridConfig dataGridConfig = DataGridConfigFactory.getInstance();
```

And, the DataGridConfig can be used to create a Filter with:

```
Filter filter = dataGridConfig.createFilter();
```

This `Filter` object can be configuerd usints its JavaBean properties to set the expression, type hint, value, and operation hint. Then, the filter can be used to filter a data set as described in here

3.3. Filtering a Data Set

Once a set of Filter objects have been obtained, they can be used to filter a data set by using a query engine or by manually filtering a data set. A query engine can be used by converting a set of Filter objects into a fragment of a query string. For example, a filter can be configured and converted into a SQL `WHERE` clause `WHERE companyname LIKE '%wheel%'` using the following code:

```
// create the DataGridConfig object for a grid
            DataGridConfig dataGridConfig =
DataGridConfigFactory.getInstance();

            // configure the filter
            Filter filter = dataGridConfig.createFilter();
            filter.setFilterExpression("companyname");
filter.setOperation(SQLSupport.mapFilterHintToOperation(FilterOperationHint.CONTAINS);
            filter.setTypeHint(FilterTypeHint.STRING);
            filter.setValue("wheel");
            List filterList = new LinkedList();
            filterList.add(filter);

            // create the WHERE clause
            String whereClause =
SQLSupport.getInstance().createWhereClause(filters);
```

Note:

If the Filter(s) are encoded in the URL, the Filter creation / configuration above can be replaced by the code to parse filters from the query string.

Notice that the type hint is explicitly specified in the example above; the type of the *companyname* could be read from a relational database's DatabaseMetaData object, but this is a very expensive way to determine the type of a column of data. Once the where clause has been obtained, it can be used to parameterize a SQL query statement.

It is also possible to programmatically filter a data set. As an example, a simple set of filter predicate objects are available in the Beehive sample listed above. In this case, the filters are created manually, and a data set is filtered in-memory to provide a subset matching the filter criteria that should be rendered.