ADF Code Corner

030. How-to intercept and modify table filter values

twitter.com/adfcodecorner

Abstract:

A new feature of ADF Faces RC is the ability for users to filter the result set of a table at runtime. For this, the developer selects the filter option on the ADF binding dialog that is shown for the table when dragging a ViewObject onto an ADF Faces RC page. At runtime, application users use an input field in the table header to provide query conditions that are appended to the underlying query defined in teh VO by the developer. A question on the OTN forum was how to intercept the query condition added by the users.

Author:

Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
15-APR-2008

Oracle ADF Code Corner is a loose blog-style series of how-to documents that provide solutions to real world coding problems.

Disclaimer: All samples are provided as is with no guarantee for future upgrades or error correction. No support can be given through Oracle customer support.

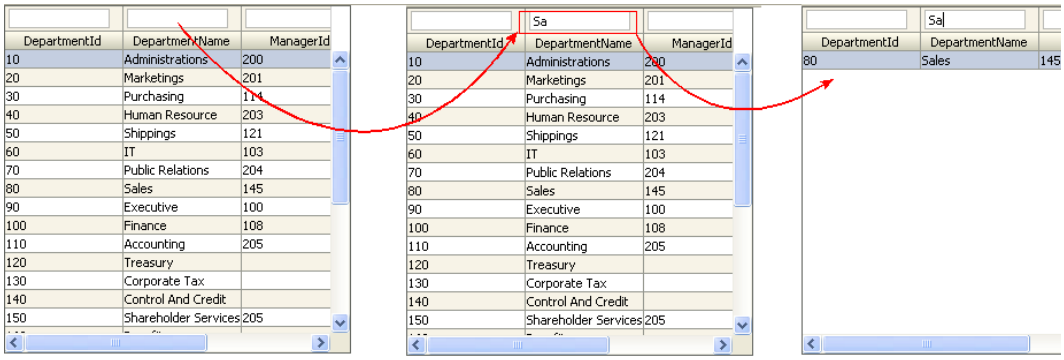
Please post questions or report problems related to the samples in this series on the OTN forum for Oracle JDeveloper: http://forums.oracle.com/forums/forum.jspa?forumID=83

Introduction

This how-to explains how a query listener on an af:table component is used to intercept table queries so the developer can programmatically modify the query conditions provided by the application user, e.g. when using the table filter.

Example

The screenshot below shows the ADF Faces RC table at runtime with the filter property enabled on its table columns. The application user can filter the table data e.g. to show all rows that have "Sa" in their department name. A developer might want to handle the cases where a user types a non numeric value into the DepartmentId filter or a mixed case string for the DepartmentName search.



Solution

The ADF Faces RC table (af:table) contains a QueryListener property that can be used to add pre- and post-processing instructions to a query using a managed bean.

```
<af:table value="#{bindings.DepartmentsView1.collectionModel}"
var="row"
```

```
...
queryListener="#{QueryFilterBean.onQuery}">
```

In the example above, the query listener of the table points to the "QueryFilterBean" managed bean. The content of the QueryFilterBean looks as follows:

```
import java.util.Map;
import javax.el.ELContext;
import javax.el.ExpressionFactory;
import javax.el.MethodExpression;
import javax.faces.context.FacesContext;
import oracle.adf.view.rich.event.QueryEvent;
import oracle.adf.view.rich.model.FilterableQueryDescriptor;
public class QueryFilterBean {

public QueryFilterBean() {}

public void onQuery(QueryEvent queryEvent) {

 // pre-processing code here

 boolean invokeQuery = true;
 /*
  * Method called by the Query Listener. This method checks if
  * the DepartmentId parameter contains a valid number and puts
  * the DepartmentName into the expected case
 */
 FilterableQueryDescriptor fqd =
          (FilterableQueryDescriptor)
 queryEvent.getDescriptor();
 Map map = fqd.getFilterCriteria();

 // ensure DepartmentId contains a Number
 String departmentId = (String) map.get("DepartmentId");

 if (departmentId != null && departmentId.length()>0){

 try {
   // try to parse String to integer
    Long.parseLong(departmentId);
 } catch (Exception ex) {
    // not a string
    System.out.println("Not a string");
    // add some error message here
    // unset selection
    map.remove("DepartmentId");
    invokeQuery = false;
   }
  }

  // ensure the initial character is in uppercase
```

```
    String departmentName = (String) map.get("DepartmentName");
    if (departmentName != null && departmentName.length()>0){
      StringBuffer sbuf = new StringBuffer();
      sbuf.append(departmentName.substring(0,1).toUpperCase());
      sbuf.append(departmentName.substring(1).toLowerCase());
      map.put("DepartmentName",sbuf.toString());
    }

    if (invokeQuery) {
      // execute the defaul QueryListener code added by JDeveloper
      //when creating the table
      invokeMethodExpression(
         "#{bindings.DepartmentsView1Query.processQuery}",
          Object.class,QueryEvent.class,queryEvent);
       }
    // post processing code here
  }

  // The code below should be in a Utility clas
public Object invokeMethodExpression(
       String expr, Class returnType,
       Class[] argTypes,Object[] args){

    FacesContext fc = FacesContext.getCurrentInstance();
    ELContext elctx = fc.getELContext();
    ExpressionFactory elFactory =
    fc.getApplication().getExpressionFactory();
    MethodExpression methodExpr =
    elFactory.createMethodExpression(
          elctx,
          expr,
          returnType,
          argTypes);
    return methodExpr.invoke(elctx,args);
   }

   public Object invokeMethodExpression(
                String expr, Class returnType,
                Class argType, Object argument){
    return invokeMethodExpression(expr, returnType,new
    Class[]{argType}, new Object[]{argument});
    }
  }
```

The code above is called whenever a user submits a query on the table. The DepartmentId and DepartmentName filters are checked for a search string and, if provided, if the condition for the DepartmentId is of type number and the case of the DepartmentName filter string starts with a capital letter followed by all lower case.

After correcting or ignoring the user query filter, the managed bean invokes the QueryListener method that JDeveloper added by default when binding the table to ADF.

Download

You can download the sample workspace from the ADF Code Corner website:

http://www.oracle.com/technetwork/developer-tools/adf/learnmore/index-101235.html