Tapestry5HibernateGridDatasource3

Grid datasource for hibernate queries. Abstract methods are used for creating a Query Object, so it is quite flexible; there is no requirement to use a DAO, HQL, or Criteria. It is compatible with any org.hibernate.Query object.

It was originally designed for use with HQL. It probably isn't as optimal for DaO as something specifically designed for that (unless your DaO object returns a org.hibernate.Query object).

To use it, extend it and implement createQuery(...) and createCountQuery(). Optionally use generateOrderBy(...) inside your createQuery(...) method to generate an ORDER BY clause supporting any weird property mapping.

It is intended to be extended for each page/component where it is used, and a new instance is created for each render. Theoretically, you could just call initAvailableRows() instead of creating a new one, or just remove the "if (this.rowCount == -1)" condition in getAvailableRows().

```
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

import org.apache.log4j.Category;
import org.apache.tapestry.grid.ColumnSort;
import org.apache.tapestry.grid.GridDataSource;
import org.apache.tapestry.grid.SortConstraint;
import org.hibernate.Query;

/**
 * Grid datasource for hibernate queries. Abstract methods are used for creating
 * a Query Object, so it is quite flexible; there is no requirement to use a
 * DAO, HQL, or Criteria. It is compatible with any org.hibernate.Query object.<br>
 * <br>
 * It was originally designed for use with HQL. It probably isn't as optimal for
 * DaO as something specifically designed for that (unless your DaO object
 * returns a org.hibernate.Query object). <br>
 * <br>
 * To use it, extend it and implement createQuery(...) and createCountQuery().
 * Optionally use generateOrderBy(...) inside your createQuery(...) method to
 * generate an ORDER BY clause supporting any weird property mapping. <br>
 * <br>
 * It is intended to be extended for each page/component where it is used, and a
 * new instance is created for each render. Theoretically, you could just call
 * initAvailableRows() instead of creating a new one, or just remove the
 * &amp;quot;if (this.rowCount == -1)&amp;quot; condition in getAvailableRows().<br>
 * <br>
 * Tested with Tapestry 5.0.11, JDK1.6
 *
 * @author pmaloney, tli
 * @version $Id: HibernateGridDataSource.java,v 1.6 2008/06/10 17:37:07 pmaloney
 *          Exp $
 */
public abstract class HibernateGridDataSource implements GridDataSource {
    private static Category log = Category.getInstance( HibernateGridDataSource.class );

    private int rowCount = -1;
    private int startIndex = -1;
    private List<?> pageRowList;

    private Class<?> rowType;


    public HibernateGridDataSource(Class<?> rowType) {
        this.rowType = rowType;
        this.rowCount = -1;
    }

    /**
     * @return a Query for fetching rows for a page. There should be no OFFSET
     *         or LIMIT clause, since setFirstResult(...) and setFetchSize(...)
     *         will be called by HibernateGridDataSource. Call
     *         HibernateGridDataSource.generateOrderBy(List,Map) to generate an
     *         order by clause.
```

```
     */
    public abstract Query createQuery(List<SortConstraint> sortConstraints);

    /**
     * @return a Query that would be suitable for finding a count. Its result
     *         should be one java.lang.Number (Such as Integer, or Long) (one
     *         row, one cell).
     */
    public abstract Query createCountQuery();

    /**
     * @param sortConstraints The List of SortConstraint objects to sort with
     * @param propertyToHql maps property names from the grid bean model to hql
     *        expressions. If the map is not null and the property name is in
     *        the map, the expression is used, else the property name is used
     *        normally.
     */
    public static String generateOrderBy(List<SortConstraint> sortConstraints, Map<String,String>
propertyToHql) {
        StringBuilder hql = new StringBuilder();
        ArrayList<String> orderByExpressions = new ArrayList<String>();

        for( SortConstraint sc : sortConstraints ) {
            if( sc.getColumnSort() == ColumnSort.UNSORTED )
                continue;

            String name = sc.getPropertyModel().getPropertyName();

            if( propertyToHql != null ) {
                String hqlEx = propertyToHql.get(name);
                if( hqlEx != null ) {
                    name = hqlEx;
                }
            }

            String ex;
            if( sc.getColumnSort() == ColumnSort.DESCENDING )
                ex = name + " desc";
            else
                ex = name;

            orderByExpressions.add( ex );
        }

        int i=0;
        for( String ex : orderByExpressions ) {
            if( i == 0 )
                hql.append(" ORDER BY ");
            else
                hql.append(", ");
            hql.append( ex );
            i++;
        }

        if( log.isDebugEnabled() )
            log.debug( "In generateOrderBy(...), hql = " + hql );
        return hql.toString();
    }

    /**
     * Calls generateOrderBy(sortConstraints, null)
     *
     * @See generateOrderBy(List<SortConstraint>, Map<String,String>)
     */
    public static String generateOrderBy(List<SortConstraint> sortConstraints) {
        return generateOrderBy(sortConstraints, null);
    }

    public void initAvailableRows(){
        Query q = createCountQuery();
        Number count = (Number) q.uniqueResult();
```

```
    if( log.isDebugEnabled() )
        log.debug("count = " + count);

    this.rowCount = count.intValue();
}

/**
 * Returns the number of rows available in the data source.
 */
public int getAvailableRows() {
    if (this.rowCount == -1)
        initAvailableRows();
    return rowCount;
}

/**
 * Invoked to allow the source to prepare to present values. This gives the
 * source a chance to pre-fetch data (when appropriate) and informs the
 * source of the desired sort order.
 *
 * @param startIndex
 *            the starting index to be retrieved
 * @param endIndex
 *            the ending index to be retrieved
 * @param sortConstraints
 *            the property model that defines what data will be used for
 *            sorting, or null if no sorting is required (in which case,
 *            whatever natural order is provided by the underlying data
 *            source will be used)
 */
public void prepare(int startIndex, int endIndex, List<SortConstraint> sortConstraints) {
    if( log.isDebugEnabled() )
        log.debug(
            "Processing prepare(...) startIndex = " + startIndex + ", endIndex = " + endIndex );
    int maxResults = endIndex - startIndex + 1;

    Query q = createQuery(sortConstraints);
    q.setFirstResult(startIndex);
    q.setFetchSize(maxResults);
    q.setMaxResults(maxResults);
    List<?> rows = q.list();
    pageRowList = rows;
    this.startIndex = startIndex;
}

/**
 * Returns the row value as rows on evt the provided index. This method will
 * be invoked in sequential order.
 *
 * @param databaseIndex index based on all rows (superset of what was
 *        fetched in prepare), so subtract startIndex to get the index in
 *        the result set.
 */
public Object getRowValue(int databaseIndex) {
    int collectionIndex = databaseIndex - startIndex;
    Object entityObject = null;

    if (pageRowList.size() > collectionIndex)
        entityObject = pageRowList.get(collectionIndex);
    else {
        // This else is a work around for getRowValue(...) with incorrect index.
        // The theory is that the grid is caching the number of rows so if the row count
        // on the previous request was larger, then index can be out of bounds.
        try {
            return getRowType().newInstance();
        } catch (Exception e) {
            throw new RuntimeException("index was invalid, and failed to create a dummy row.", e);
        }
    }
    return entityObject;
```

```
    }

    /**
     * Returns the type of value in the rows, or null if not known.
     *
     * @return the row type, or null
     */
    public Class<?> getRowType() {
        return this.rowType;
    }

}
```