Event - DatasourceRule

Allgemein

Regeln des Typs "DatasourceRule" dienen zur Bereitstellung von Daten für eine Datenquelle vom Typ "Dynamische Taskliste" oder "Report & Formular" durch die Implementierung des Interfaces `org.nuclos.api.rule.DatasourceRule` .

Innerhalb des Baums mit der Regelbibliothek werden sie dem Knoten "DatasourceRule" zugeschrieben.

Struktur

```
package example.rest;

import org.nuclos.api.rule.DatasourceRule;
import org.nuclos.api.UID;
import org.nuclos.api.datasource.DatasourceColumn;
import org.nuclos.api.datasource.DatasourceResult;
import org.nuclos.api.provider.DatasourceProvider;
import org.nuclos.api.provider.QueryProvider;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Locale;
import java.util.Map;

public class DynamicArticleTasklistDatasourceRule implements DatasourceRule {

        /**
        * Diese Methode liefert die Daten für die Datenquelle in Form eines Objekts vom Type
"DatasourceResult" zurück
        * und muss als einzige des Interfaces implementiert werden.
        * Zur Einschränkung der Ergebnisse können die Parameter ausgewertet werden, die wie üblich in Form
einer "Map<String, Object>" übergeben werden.
        * Das zurückgegebene Objekt beinhaltet die Spaltendefinitionen als "List<DatasourceColumn>" und die
Menge der Zeilen als "List<Object[]>".
        * Objekte vom Typ "DatasourceColumn" beinhalten die Information über den Namen der Spalte und den
Datentyp und
        * können über den Aufruf von "DatasourceProvider.createResultColumn(name, type)" erzeugt werden.
        *
        **/
    @Override
    public DatasourceResult getData(Map<String, Object> params) {
        List<DatasourceColumn> columns = Arrays.asList(
            DatasourceProvider.createResultColumn("number", Integer.class),
            DatasourceProvider.createResultColumn("name", String.class),
            DatasourceProvider.createResultColumn("active", Boolean.class),
            DatasourceProvider.createResultColumn("intid", Long.class)
        );

        List<Object[]> rows = new ArrayList<>();

        List<Article> lstArticles = QueryProvider.execute(QueryProvider.create(Article.class).where(Article.
Active.eq(true)));

        for (Article article : lstArticles) {
            rows.add(new Object[] {article.getArticleNumber(), article.getName(), article.getActive(),
article.getId()});
        }

        return DatasourceProvider.createResult(rows, columns);
    }

        /**
```

```
        * Die nachfolgenden Methoden können implementiert werden, haben allerdings eine
Standardimplementierung im Interface. Diese ist in
        * dieser exemplarischen Implementierung der Klasse angegeben.
        **/

        /**
        * Liefert die Anzahl der Datensätze aus der Datenquelle unter Berücksichtigung der Suchbedingung und
der Parameter.
        **/
        @Override
    public Long count(SearchExpression se, Map<String, Object> params) {
        return (long)this.getData(params).getRows().size();
    }

        /**
        * Liefert alle Datensätze aus der Datenquelle unter Berücksichtigung der Suchbedingung und der
Parameter.
        **/
    @Override
    public DatasourceResult getAll(SearchExpression se, Map<String, Object> params) {
        return this.getData(params);
    }

        /**
        * Liefert die IDs aller Datensätze aus der Datenquelle unter Berücksichtigung der Suchbedingung und
der Parameter.
        **/
    @Override
    public List<Long> getAllIds(SearchExpression se, Map<String, Object> params) {
        DatasourceResult data = this.getData(params);

        int i;
        for(i = 0; i < data.getColumns().size(); ++i) {
            DatasourceColumn column = (DatasourceColumn)data.getColumns().get(i);
            if ("INTID".equals(column.getName())) {
                break;
            }
        }

        return (List)data.getRows().stream().map((row) -> {
            return (Long)row[i];
        }).collect(Collectors.toList());
    }

        /**
        * Liefert einen Datensatz mit einer bestimmten ID. Diese ID wird als Parameter "INTID" in der Map
übergeben.
        **/
    @Override
    public DatasourceResult getById(Map<String, Object> params) {
        DatasourceResult data = this.getData(params);

        int i;
        for(i = 0; i < data.getColumns().size(); ++i) {
            DatasourceColumn column = (DatasourceColumn)data.getColumns().get(i);
            if ("INTID".equals(column.getName().toUpperCase())) {
                break;
            }
        }

        return (DatasourceResult)data.getRows().stream().filter((row) -> {
            return row[i] != null && row[i].equals(params.get("INTID"));
        }).map((row) -> {
            List<Object[]> rows = new ArrayList();
            rows.add(row);
            return DatasourceProvider.createResult(rows, data.getColumns());
        }).findFirst().orElse(DatasourceProvider.createResult(new ArrayList(), data.getColumns()));
    }

        /**
        * Liefert eine Menge von Datensätzen mit bestimmten IDs. Diese IDs werden als "List<Long>" im
```

```
Parameter "INTID" in der Map übergeben.
        **/
    @Override
    public DatasourceResult getByIds(Map<String, Object> params) {
        DatasourceResult data = this.getData(params);

        int i;
        for(i = 0; i < data.getColumns().size(); ++i) {
            DatasourceColumn column = (DatasourceColumn)data.getColumns().get(i);
            if ("INTID".equals(column.getName().toUpperCase())) {
                break;
            }
        }

        return DatasourceProvider.createResult((List)data.getRows().stream().filter((row) -> {
            return ((List)params.get("INTID")).contains(row[i]);
        }).collect(Collectors.toList()), data.getColumns());
    } }
```