Java数据集字段支持分类

需求背景

项目中可能出现Java数据集字段比较多的情况，数量可能达到几百或几千个，这样在Java数据集对象定义界面中显示参数及结果集字段时，浏览器界面可能无法响应，另外新建Java数据集界面的资源树中显示太多的字段也不方便查找与使用。因此，需要Java数据集能够支持数据集字段按分类显示。

实现方案

在Java数据集的接口(smartbi.freequery.metadata.IJavaQueryData)中添加支持字段分类的相关接口，如下所示：

```
/**
 *
 *
 * @param parentId
 *            IDIDnull
 * @return elements list
 */
public List<JavaQueryCatalogElement> getCatalogElements(String parentId);
/**
 *
 *
 * @param catalogId
 *            ID
 * @return fields list
 */
public List<JavaQueryOutputField> getOutputFields(String catalogId);
/**
 *
 *
 * @return
 */
public List<? extends IField> getSelectedFields();
/**
 *
 *
 * @param feilds
 */
public void setSelectedFields(List<? extends IField> feilds);
```

注意事项

在Java数据集对象定义界面中显示参数及结果集字段时，通过"表格树"控件动态显示结果集字段。

1、在同一个Java数据集实现中由方法getOutputFields返回的所有输出字段(含所有分类下的输出字段)的字段名称不能重复。
2、产品会在适当的时机自动调用方法setSelectedFields以告知实现类当前Java数据集所选择的报表字段。实现类需要根据这些字段信息在方法"public GridData getGridData(int from, int count);"中返回对应字段的数据。
3、实现分类接口后，原接口方法"public List<JavaQueryOutputField> getOutputFields();"可以返回空实现。

实现示例

```
package smartbi.freequery.metadata;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import smartbi.catalogtree.CatalogElement;
import smartbi.catalogtree.CatalogTreeModule;
```

```
import smartbi.freequery.metadata.IJavaQueryData.ICatalogFieldSupport;
import smartbi.freequery.querydata.CellData;
import smartbi.freequery.querydata.GridData;
import smartbi.freequery.repository.BasicField;
import smartbi.freequery.repository.FreeQueryDAOFactory;
import smartbi.freequery.repository.IField;
import smartbi.util.CatalogElementType;
import smartbi.util.StringUtil;
public class TestJavaQueryData implements IJavaQueryData, ICatalogFieldSupport {
        private List<? extends IField> selectedFields = null;
        private static final String ID_PREFIX = "JQID_";
        @Override
        public List<JavaQueryCatalogElement> getCatalogElements(String parentId) {
                List<JavaQueryCatalogElement> result = new ArrayList<JavaQueryCatalogElement>();
                if (StringUtil.isNullOrEmpty(parentId)) {
                        parentId = "DS.SYSTEM";
                }
                parentId = parentId.replaceFirst(ID_PREFIX, "");
                List<CatalogElement> elems = CatalogTreeModule.getInstance()
                                .getChildElements(parentId);
                if (elems != null) {
                        for (CatalogElement elem : elems) {
                                String type = elem.getType();
                                if (!CatalogElementType.SCHEMA.name().equals(type)
                                                && !CatalogElementType.BASETABLE.name().equals(type)) {
                                        continue;
                                }
                                result.add(new JavaQueryCatalogElement(
                                                ID_PREFIX + elem.getId(), elem.getName(), elem
                                                                .getAlias(), elem.getDesc()));
                        }
                }
                return result;
        }
        @Override
        public List<JavaQueryOutputField> getOutputFields(String catalogId) {
                List<JavaQueryOutputField> result = new ArrayList<JavaQueryOutputField>();
                if (StringUtil.isNullOrEmpty(catalogId)) {
                        return result;
                }
                catalogId = catalogId.replaceFirst(ID_PREFIX, "");
                CatalogElement parentElem = CatalogTreeModule.getInstance()
                                .getCatalogElementById(catalogId);
                String parentName = parentElem == null ? "" : parentElem.getName();
                List<CatalogElement> elems = CatalogTreeModule.getInstance()
                                .getChildElements(catalogId);
                if (elems != null) {
                        for (CatalogElement elem : elems) {
                                String type = elem.getType();
                                if (!CatalogElementType.FIELD.name().equals(type)) {
                                        continue;
                                }
                                BasicField f = FreeQueryDAOFactory.getBasicFieldDAO().load(
                                                elem.getId());
                                if (f == null) {
                                        continue;
                                }
                                result.add(new JavaQueryOutputField(ID_PREFIX + f.getId(),
                                                parentName + "_" + f.getName(), f.getAlias(), f
                                                                .getDesc(), f.getDataType(), f.
getDataFormat()));
                        }
                }
                return result;
        }
        @Override
        public void loadConfigs(String configs) {
                //
        }
        @Override
        public String saveConfigs() {
```

```
        return null;
}
@Override
public List<JavaQueryConfig> getConfigs() {
        return new ArrayList<JavaQueryConfig>();
}
@Override
public void setConfigValue(String key, String value) {
        //
}
@Override
public void setConfigValues(Map<String, String> configValues) {
        //
}
@Override
public void init() {
        //
}
@Override
public List<JavaQueryParameter> getParameters() {
        return new ArrayList<JavaQueryParameter>();
}
@Override
public List<JavaQueryOutputField> getOutputFields() {
        List<JavaQueryOutputField> result = new ArrayList<JavaQueryOutputField>();
        // getOutputFields(null, result);
        return result;
}
protected void getOutputFields(String parentId,
                List<JavaQueryOutputField> result) {
        result.addAll(getOutputFields(parentId));
        List<JavaQueryCatalogElement> elems = getCatalogElements(parentId);
        for (JavaQueryCatalogElement elem : elems) {
                getOutputFields(elem.getId(), result);
        }
}
@Override
public void setParameterValue(String id, String value, String displayValue) {
        //
}
@Override
public int getRowCount() {
        return Integer.MAX_VALUE;
}
@Override
public GridData getGridData(int from, int count) {
        GridData gridData = new GridData();
        List<? extends IField> outfields = getSelectedFields();
        List<String> stringHeaders = new ArrayList<String>();
        for (int i = 0; i < outfields.size(); i++) {
                stringHeaders.add(outfields.get(i).getName());
        }
        gridData.setStringHeaders(stringHeaders);
        List<List<CellData>> cells = new ArrayList<List<CellData>>();
        int totalRows = 25;
        int to = Math.min(from + count, totalRows);
        for (int i = from; i < to; i++) {
                List<CellData> row = new ArrayList<CellData>();
                for (int j = 0; j < outfields.size(); j++) {
                        CellData cell = new CellData();
                        cell.setIntValue(j + 1);
                        cell.setStringValue("" + (j + 1));
                        row.add(cell);
                }
                cells.add(row);
        }
        gridData.setData(cells);
        gridData.setTotalRowsCount(totalRows);
        return gridData;
}
@Override
```

```
        public void close() {
                //
        }
        @Override
        public List<? extends IField> getSelectedFields() {
                return selectedFields == null ? new ArrayList<IField>()
                                : selectedFields;
        }
        @Override
        public void setSelectedFields(List<? extends IField> feilds) {
                selectedFields = feilds;
        }
}
```

【本文结束】