自定义函数-Oracle in语法超1000的问题

1 概述

需求：

基于ORACLE 数据库建立查询的时候，会碰到这样的问题：

ORA-01795maximum number of expression in a list is 1000
导致这个问题的原因是因为SQL语句中用到了IN字句，结果IN中的元素个数超过了1000导致了这个错误。

实现方案

开发一个系统函数Process1000LimitOfIn("字段名", 树或下拉参数)，让其输出

(字段 in (list1) or 字段 in (list2))

函数有两个参数：

第一个参数：字段名，如果是可视化，直接拖字段，如果是原生sql最好用英文双引号括起来，如"table1.A"
第二个参数：smartbi的多选树或下拉框参数，直接拖参数即可

2 自定义函数

Smartbi自定义函数是按照以下规范实现对应接口,然后可以通过升级类或手动将函数添加到资源树中供用户使用。

1，自定义函数可继承类：smartbi.freequery.expression.function.Function；

2，这个实现类必须放在smartbi.freequery.expression.function包下面，因为系统运行自定义函数时会自动加上这个包路径查找函数对应的实际类。

定义好的自定义函数可以手动添加到资源树中，也可以使用升级类添加。

自定义函数的设置方法见wiki示例：

系统函数——自定义函数

2.1 编写自定义系统函数类Process1000LimitOfIn.java

用于当IN字句中元素个数超过1000时，将其转化为 字段 in (list1) or 字段 in (list2)的形式

Process1000LimitOfIn.java

```
package smartbi.freequery.expression.function;
import smartbi.SmartbiException;
import smartbi.freequery.metadata.SQLPart;
import smartbi.freequery.metadata.SQLPartList;
import smartbi.freequery.util.SQLPartType;
import smartbi.util.StringUtil;
public class Process1000LimitOfIn extends Function {
        @Override
        public boolean checkParams() {
                if (params.length==2)
                        return true;
                else
                        return false;
        }
        protected void prepareParams() {
                try {
                        super.prepareParams();
                } catch (SmartbiException e) {
                        result = new SQLPartList();
                }
        }
        @Override
        public void execute() {
                try {
                        String fieldName  = params[0].getSqlStr().trim();
                        fieldName = trim(fieldName,"\"");
                        fieldName = trim(fieldName,"'");
                        String fieldValue = params[1].getSqlStr().trim();
                        String sep = ",";
```

```
                        String[] fieldValueArray = fieldValue.split(sep);
                        int size = 1000;
                        long count = Math.round(Math.ceil(((double)fieldValueArray.length)/size));
                        if(count ==1){
                                result.add(new SQLPart(SQLPartType.SQLSTR, fieldName + " in (" + fieldValue
+")"));
                                return;
                        }
                        //TODO
                        StringBuilder sqlSb = new StringBuilder();
                        sqlSb.append("(");
                        for(int i =0; i<count; i++){
                                int jLen = (i+1)*size ;
                                jLen = (jLen>fieldValueArray.length)?fieldValueArray.length:jLen;
                                StringBuilder sb = new StringBuilder();
                                for(int j = i*size; j<jLen; j++){
                                        sb.append(fieldValueArray[j]).append(sep);
                                }
                                String sbStr = sb.toString();
                                if(i>0){
                                        sqlSb.append(" or ");
                                }
                                sqlSb.append(fieldName + " in (" + sbStr.substring(0, sbStr.length()-sep.
length())+")");
                        }
                        sqlSb.append(" ) ");
                        result.add(new SQLPart(SQLPartType.SQLSTR, sqlSb.toString()));
                } catch (Exception e) {
                        result.add(new SQLPart(SQLPartType.SQLUnknown," 1=1 "));
                }
        }

        private String trim(String srcStr, String toTrimStr){
                if(StringUtil.isNullOrEmpty(srcStr)){
                        return srcStr;
                }
                srcStr = srcStr.trim();
                while(srcStr.startsWith(toTrimStr)){
                        srcStr = srcStr.substring(toTrimStr.length());
                        srcStr = srcStr.trim();
                }
                while(srcStr.endsWith(toTrimStr)){
                        srcStr = srcStr.substring(0, srcStr.length()-toTrimStr.length());
                        srcStr = srcStr.trim();
                }
                return srcStr;
        }
        @Override
        public String getMDXValue() {
                return "";
        }
}
```

2.2 使用升级类将自定义函数类添加到资源树中

编写好的自定义函数类，需要添加到资源树中，供用户使用，本节是使用升级类自动在指定的系统函数中的树节点下，增加自定义函数，执行了这一步就不需要2.3 手动增加自定义函数。

该升级类的操作是向知识库t_restree（资源树）表中插入一条数据，父节点为catalog_string（字符串），资源类型为FUNCTION（函数）
该操作用于后面在资源树中新建同名的自定义函数

UpgradeTask_New.java

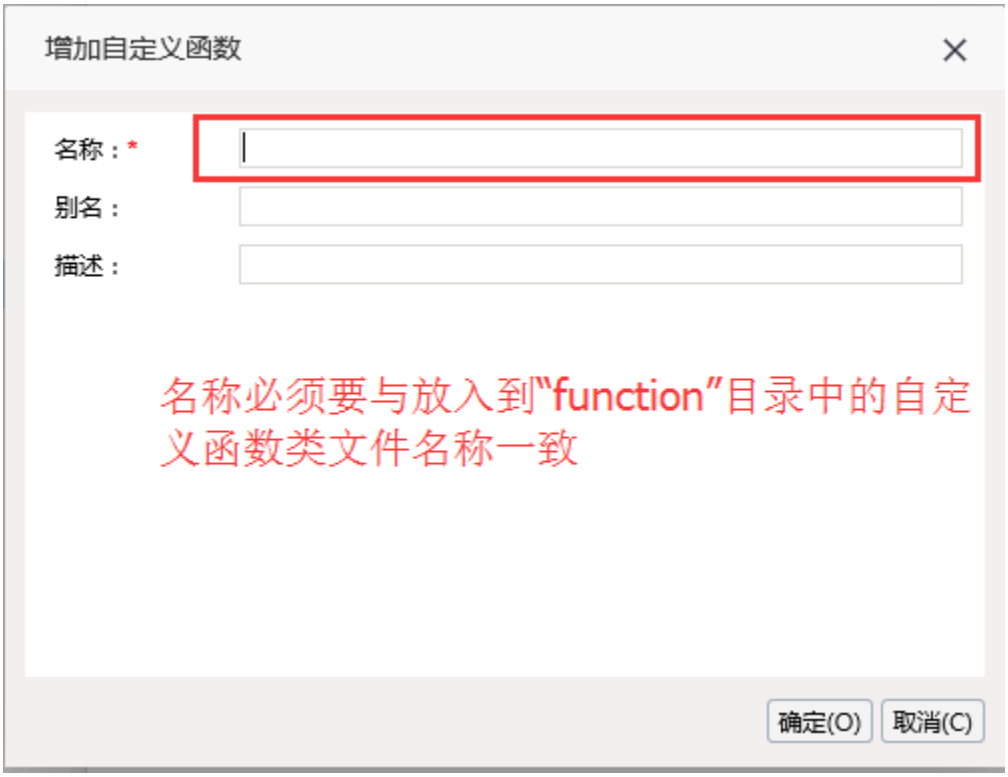
```
package smartbi.ext.function.upgrade;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.Timestamp;
import org.apache.log4j.Logger;
import smartbi.repository.UpgradeTask;
import smartbi.util.DBType;
/**
 *
 */
public class UpgradeTask_New extends UpgradeTask {
        /** */
        private static final Logger LOG = Logger.getLogger(UpgradeTask_New.class);
        @Override
        public boolean doUpgrade(Connection conn, DBType type) {
                PreparedStatement prep = null;
                try {
                        Statement sta = conn.createStatement();
                        ResultSet rs = sta.executeQuery("select c_resid from t_restree where c_resid =
'func_Process1000LimitOfIn'");
                        if(!rs.next()){
                                prep = conn.prepareStatement("insert into t_restree(c_resid, c_resname,
c_resalias, c_pid, c_restype, c_order, c_perm, c_resdesc, c_created, c_lastmodified) values(?, ?, ?, ?, ?, ?,
?, ?, ?, ?)");
                                prep.setString(1, "func_Process1000LimitOfIn");
                                prep.setString(2, "Process1000LimitOfIn");
                                prep.setString(3, "Process1000LimitOfIn");
                                prep.setString(4, "catalog_string");
                                prep.setString(5, "FUNCTION");
                                prep.setInt(6, 9000);
                                prep.setString(7, "<Permissions inherited=\"YES\" role=\"ADMINS\"><Permission
perm=\"READ\" descend=\"FOLDER_FILE\"/></Permissions>");
                                prep.setString(8, "1000Process1000LimitOfIn(\"A\",)<br/> 1 table1.Asql,\"table1.
A\"2 <br/>1000or((A in (a,b,c) or A in (d,e,f)) )");
                                prep.setTimestamp(9, new Timestamp(new java.util.Date().getTime()));
                                prep.setTimestamp(10, new Timestamp(new java.util.Date().getTime()));
                                prep.executeUpdate();
                                prep.close();
                        }
                        rs.close();
                        sta.close();
                        return true;
                } catch (SQLException e) {
                        LOG.error("Upgrade UpgradeTask_New '" + this.getClass().getPackage().getName() + "' to
" + getNewVersion() + " fail.", e);
                        return false;
                }
        }
        @Override
        public String getNewVersion() {
                return "0.0.1";
        }
}
```

2.3 手动方式增加自定义函数

前面讲了使用升级类添加自定义函数方式，假如不想写升级类（升级类有个好处就是只要加载了扩展包即可，不用手动添加便于迁移），这里介绍手动添加方式。

1，在定制管理界面，找到 函数列表 > 系统函数 > 字符串节点。

2， 在“字符串”的右键菜单中选择 增加自定义函数，弹出“自定义函数”窗口。

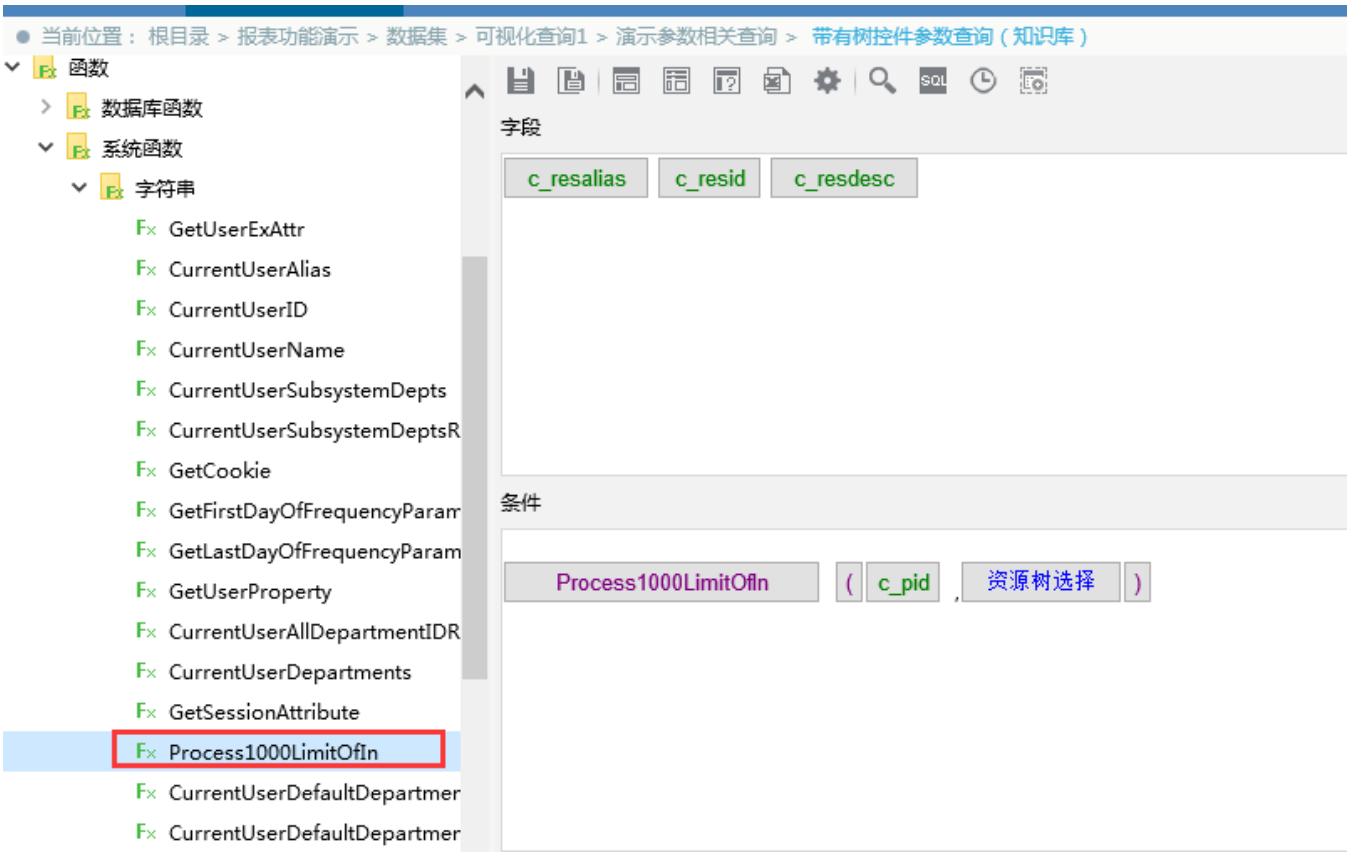


3 在“自定义函数”窗口输入名称，该名称必须要与上面自定义函数类文件名称一致，譬如本例就是Process1000LimitOfIn。

4 创建完成后，就可以和系统内置函数一样在产品中拖拽使用了。

2.4 使用自定义函数

2.4.1 可视化查询中使用



2.4.2 原生SQL查询中使用

3 相关资源

Process1000LimitOfIn.ext，源码：Process1000LimitOfIn_Src.rar。