报表开发

参考手册

（v 1.0.2）

上海安硕信息技术有限公司

二〇〇五年一月

目　录

1 引言

1.1 编写目的

为了能更好地说明报表开发解析环境 ARS-4E（AmarSoft Report System - 4Engine）及其开发工具 ReportStudio 的具体应用，本用户手册意在为报表开发人员和报表维护人员提供一份可供参考的技术文档，同时也为前期的培训移交工作提供参考。

1.2 定义

为了避免混淆，对于一些在本文中提及到的特定称谓，作出一下说明：

“恢复”：将 ReportStudio 备份本地数据文件中的数据复制到连接数据库中。

“备份”：将服务器中的数据复制到本地备份数据文件中。

“本机”：运行报表设置工具 ReportStudio 的计算机。

“服务器”：数据库运行所在的服务器计算机。

“本地表”：本地数据文件中的数据表。

“服务器表”：数据服务器中的数据表。

“脚本”：由 ReportStudio 导出生成的.sql 执行脚本。

“文本”： 由 ReportStudio 导出生成的文本文件。

1.3 参考资料

配合本《开发手册》，可参考一下技术文档：

1、《报表部署文档》

2、《设计概要》

3、《展示页面调整参考》

4、《ReportStudio 用户手册》

2 概述

2.1 报表开发方式

通过 DataManger 正确设置了“报表模板”、“报表元素”及“报表元素关联”之后，经过后台的元素解析模块，报表展示页面将解析之后的结果展示在 web 浏览器上。

将报表开发方式由之前的逐张对应编写.jsp 页面，将业务处理逻辑嵌套在页面中的开发模式转变成为从烦杂的交叉嵌套开发剥离出报表最为关心的数据提取工作，数据提取外的逻辑处理由解析模块进行处理。

2.2 特点

2.2.1 免于重复的页面维护

经过业务逻辑的分离候，页面展示处理部分不再需要开发人员进行重复维护，通用的处理页面大大降低了局部变更所造成的影响。

2.2.2 专注于数据获取

开发人员可以主要精力集中在数据库数据的过滤提取上，只要保证 SQL 语句的正确及相应需要的定义、操作处理，就能得到所需要的数据展示格式。

2.2.3 通过设置工具设置报表元素

通过分析，将一张报表分解成为几个部分，称之为“元素”，通常情况下，一张报表模板对应多个报表元素。

通过 ReportStudio 将报表模板的定义，报表元素的定义及其关联可以很方便地进行设置和修改。

2.2.4 较便利的数据再处理

在对报表元素的处理过程中，涉及到很多“处理定义”及“处理操作”，利用这些操作对通过 SQL 取出得源数据集进行再处理。

2.2.5 Javasript 脚本支持

对 javascript 脚本得支持使得元素的处理流程可以得到更好的控制。

3 涉及变量说明

作为查询条件的一部分，DataManger 本身提供一些可在元素中使用的自定义变量，这些变量可以在元素设置中使用。

3.1 系统变量

为了满足基本的查询条件，系统提供如下的变量，以便构建满足相应要求的统计查询条件。

```
OrgID                   //查询机构
InputDate               //查询日期
Currency                //查询转换币种（可选）
Unit                    //查询转换金额单位（可选）
```

3.2 时间变量

为满足报表查询时间跨度的要求，提供一下时间变量以供使用，如果需要增加新的时间变量，参看《展示模块调整说明》。

```
Tomorrow            //明天（yyyy/MM/dd）
Yesterday           //昨天（yyyy/MM/dd）
NextMonth           //下月（yyyy/MM）
PreMonth            //上月（yyyy/MM）
CurrentMonth        //当月（yyyy/MM）
CurrentMonthEnd     //当月末（yyyy/MM）
PreQuarter          //上季（yyyy/MM）
PreQuarter2         //上 2 季（yyyy/MM）
PreQuarter3         //上 3 季（yyyy/MM）
PreQuarter4         //上 4 季（yyyy/MM）
NextQuarter         //下季（yyyy/MM）
LastYear            //去年（yyyy）
ThisYear            //今年（yyyy）
NextYear            //明年（yyyy）
PreYearEnd          //上年末（yyyy/MM）
PreHalfYear         //前半年（yyyy/MM）
NextHalfYear        //下半年（yyyy/MM）
```

3.3 CELL 变量

针对报表的表头，如“填报单位”、“填报日期”及“X 月”等的需要局部填充的变量，提供 CELL 变量的解决方法，变量如下：

```
FillinDate              //填报日期
FillinQuarter           //填报日期所属季度
FillinOrgName           //填报机构
FillinMonthOnly         //填报所处月份
FillinYearOnly          //填报所处年份
UnitName                //单位：元，......
```

3.4 表达式变量

表达式变量是为了处理将由于不同统计条件选择带来对查询的影响一般化。例如：通过币种条件的选择区分统计不同币种的授信业务，在 sql 中体现为币种字段取不同的值，造成的结果是同一取数的业务逻辑要重复定义两句 sql 语句。

为了便于维护和开发的一致性，采用表达式变量来处理由于条件造成的表达式差异，通过拼 sql 表达式的方式实现不同条件的兼容。

3.4.1 汇率 ERate

由于统计币种条件的不同，造成对应汇率的条件的不同。

（字段 ERateUSD 为外币兑美元汇率；ERate 为外币兑人民币汇率）

举例：统计本期余额，要求能兼容本外币条件。

```
Var sSql = “SELECT Balance”+ERate
+” …”
+” FROM …”
+” WHERE …”
;
```

通过余额表达式变量 ERate 在不同的币种条件下拼出不同的 sql。如下：

币种条件	表达式变量值
人民币	Balance
外币转美元	Balance***ERateUSD**
本外币转人民币	Balance***ERate**

3.4.2 币种 CurrencyCondition

由于统计币种条件的不同，造成对应币种条件的不同。在 sql 中体现为在 where 条件下的币种条件过滤。*（字段 BusinessCurrency 为业务币种字段）*

举例：统计本期余额，要求能兼容本外币条件。

```
Var sSql = "SELECT …"
+" FROM …"
+" WHERE …"
+ CurrencyCondition
;
```

通过币种表达式变量 CurrencyConditio 不同的币种条件下拼出不同的 sql。如下：

币种条件	表达式变量值
具体币种(代码 xx)	AND BusinessCurrency='xx'
外币	AND BusinessCurrency!='01'
本外	

3.4.3 机构 OrgCondition

由于统计机构范围的不同，造成 sql 中统计通过范围的条件的不同。

（字段 OrgID 为授信业务所属管户机构）

举例：统计本期余额，要求能兼容本不同机构（总行或支行）。

```
Var sSql = "SELECT …"
+" FROM …"
+" WHERE …"
+ OrgCondition
;
```

通过机构表达式式变量 OrgCondition 在不同的统计范围条件下拼出不同的 sql。如下：

币种条件	表达式变量值
全行范围	AND OrgID NOT LIKE '11%'
一级机构（xx）	AND OrgID LIKE 'xx%'
二级支行（yy）	AND OrgID = 'yy'

3.5 条件变量

开发环境支持 Javascript 脚本，相应的 javascript 语法都能使用，如：

```
//定义变量
var s = "";
//条件判断
if (s == null){
        …
} else {
        …
}
//定义函数
function function_name () {
        …
}
//for  循环
for (i=0; i<10; i++) {
        …
}
//try
try {
        …
} catch (e) {
        …
}
//while  循环
while (s == true) {
        …
       //break;
}
```

4 定义、操作方法说明

4.1 简述

为了在通过 SQL 语句从数据库表中选取数据（称为“源数据”）的基础上继续做进一步的处理排序、对称等，系统定义了一系列的**面向数据集**的定义和操作，用以协助完成固定报表的开发。

为了提高报表开发中报表元素的维护和调试，为报表开发过程作出了的一些约定，力求能清晰地表述出原有的开发思路和开发结果。

4.2 定义说明

4.2.1 方法头

为了能明确表述出每个元素的相关信息，在报表开发过程中，要求在没有元素的顶部写明“方法头”：

【语法说明】

```
/* Title:                    //报表名称
 * Description:              //描述
 * Content:                  //指明方法的功能及需说明的地方
 * Developer:                //开发者
 * DateTime:                 //开发日期
 * Tester:                   //测试者
 * History Log:              //修改日志
 */
```

4.2.2 Var

作为通过运行 SQL 语句来获取数据的方式来说，SQL 语句的定义是个几乎必须的定义，在符合 javascript 语法下，定义变量通过关键字 **var** 来声明变量及其值，不区分具体数据类型。

【语法说明】

```
var v_Name = v_Value;
```

4.2.3 setParam 与 setSheetParam

设置局部（全局）变量

将一个变量的值设置成为局部（全局）变量，可以在许可的范围内通过相应的方法（getParam 或 getSheetParam）取得相应的变量值。

通过 setParam 设置的局部变量只能在本元素的范围内可见，也就是说使用范围在本元素内。

而通过 setSheetParam 设置的全局变量可在整个报表范围内可见，也就能在报表内任一个元素内调用这个全局变量的值。

【语法说明】

```
setParam(“局部变量名”,”局部变量值”);
```

或

```
setSheetParam(“全局变量名”,”全局变量值”);
```

```
/* setParam
 * 设置变量值
 * sParamName 变量名
 * sParamValue 变量值
 */
setParam("","");
```

```
/* setSheetParam
 * 设置 Sheet 变量值
 * sParamName 变量名
 * sParamValue 变量值
 */
setSheetParam("","");
```

【示例】

```
…
var iExample = 20;
//设定局部变量 Example，值为 iExample
setParam(“Example”, iExample);
…
```

4.2.4 setProperty 与 setSheetProperty

类似 setParam 和 setSheetParam，setProperty 和 setSheetProperty 为设置局部（全局）属性。

其变量定义可参看 *4.2.3*

4.2.5 setPosition

对于经过处理后的数据集，在展示在页面中的 MS OWC 控件中时，需要一个起始坐标点，这个坐标称为“起始点”。

在一个结果集到达展示的步骤时，将根据起始点标明的坐标值，依次展开显示整个元素的结果集合，如下图：

【语法说明】

setPosition 传递的参数为 OWC 控件中的实际展示坐标点（行，列）。

```
/* setPosition
  *  指定起始位置
  * sPosition Excel 位置
  *        行数,列数
  */
setPosition(",");
```

【示例】

```
...
setPosition(“5,1”);                    //设置第 5 行第 1 列为元素起始点
...
```

4.2.6 setCell

对于类似“填报单位”、“填报日期”这样的报表范围内的固定变量，不同的查询单位统计得出不同的查询结果，变量值由客户统计输入值确定，但通常都在一个定点上显示。针对这类变量，通过 setCell 实现定点变量展示。

需要说明的是，这类变量都是针对特定需求提供的，使用的变量也是特定的。

参看【3.3 CELL 变量】 。

【语法说明】

setCell(“定点坐标”,单元格值)

```
/* setCell
 * 设定单元格内容
 * sPosition 单元格位置
 *       行数,列数
 * sText 单元格内容
 */
setCell(",","");
```

【示例】

…

```
setCell(“3,1”,”setCell 示例”);      //值为字符串
setCell(“2,1”,FillinDate)              //值为固定变量
```

…

4.3 操作说明

【原理】

将选取出来的数据转换成为二维数组，之后针对结果二维数据集进行格式等的转换，得出所需的结果。

需要强调的是：所有操作处理的对象都是整个数据集，而不只是单行的数据或单列的数据。

4.3.1 apartResult 与 apartSetResult

【作用】

将传入的数据集按照指定拆分序列拆分之后的数据集。

【说明】

例值 1	例值 2	例值 3	例值 4	例值 5	例值 6	例值 7	例值 8	例值 9	例值 10
…	…	…	…	…	…	…	…	…	…

经过 **apartResult("3,3,4")**之后，得到新的结果集中，结果集宽度为拆分序列中的最大值：

例值 1	例值 2	例值 3	
…	…	…	
例值 4	例值 5	例值 6	
…	…	…	
例值 7	例值 8	例值 9	例值 10
…	…	…	…

而经过 **apartResult(“3,3,3”)**之后，得到新的结果集中，“例值 10”则由于在拆分序列外而遭到滤除：

例值 1	例值 2	例值 3
…	…	…
例值 4	例值 5	例值 6
…	…	…
例值 7	例值 8	例值 9
…	…	…

【语法】

apartResult(“拆分序列”)

或

apartSetResult(“拆分数据集名称”,”拆分序列”)

```
/* apartResult(sArea)
 * 进行拆散排列
 * sArea: 拆分元素序列 ,指明拆分的序列长度
 * 例子：apartResult ("2,3,4,5")
 * 将原序列中的前 2 个元素置于第一列
 * 紧接的 3 元素置于第二列
 * 之后的 4 个元素置于第三列
 * 最后的 5 个元素置于第四列
 * 对于长度不足的补空位
 */
apartResult ("");
```

【示例】

4.3.2 arrangeResult 与 arrangeArray

【作用】

将传入的源数据集进行重新翻转组合，返回一个重新组合之后的结果集。

【说明】

arrangeResult 处理系统默认的结果数据集；

arrangeArray 处理自定义结果数据集。

`arrangeResult(int iType);`

源数据集：

例值 11	例值 12	例值 13	例值 14
例值 21	例值 22	例值 23	例值 24
例值 31	例值 32	例值 33	例值 34

iType = 1 ： **`arrangeResult(1);`**将数据集沿着左上右下对角线翻转

例值 11	例值 21	例值 31
例值 12	例值 22	例值 32
例值 13	例值 23	例值 33
例值 14	例值 24	例值 34

iType = 2： **`arrangeResult(2);`**将数据集的多行数据重新组合成一行数据

值 11	值 12	值 13	值 14	值 21	值 22	值 23	值 24	值 31	值 32	值 33	值 34

iType = 3 ： **`arrangeResult(3);`**将数据集的多行数据重新组合成一列数据

例值 11
例值 21
例值 31
例值 12
例值 22
例值 32
例值 13
例值 23
例值 33
例值 14
例值 24
例值 34

【语法】

arrangeResult(*组合类型*)

或

arrangeArray(*组合类型*)

```
/* arrangeResult(iType)
 * 进行转换排列
 * iType 转换方式
 *       1: 将 X 轴和 Y 轴数据进行对换
 *       2: 以 X 轴进行 Y 轴数据对换
 *       3: 以 Y 轴进行 X 轴数据对换
 */
arrangeResult(2);
```

【示例】

4.3.3 decodeWithDefault 与 decodeWithSeparator

【作用】

『配合 fetchColumn 和 fetchSetColumn 使用』

将由 fetchColumn 或 fetchSetColumn 提取出来的列值序列进行如分隔符替换等操作。

主要使用在需要多个元素都存在同样的条件下，比如统计“十大户情况”中各元素都应该受到**同样 10 户客户号**的限定。

【说明】

例值 11	例值 12	例值 13	例值 14
例值 21	例值 22	例值 23	例值 24
例值 31	例值 32	例值 22	例值 24
例值 41	例值 42	例值 43	例值 44
例值 51	例值 52	例值 53	例值 54
例值 61	例值 62	例值 63	例值 64

```
var sExampleValue = fetchColumn(2);
var sAfterDecodeDefault = decodeWithDefault(sExampleValue);
```

```
sAfterDecodeDefault:
‘例值 12’,’ 例值 22’,’ 例值 32’,’ 例值 42’,’ 例值 52’,’ 例值 62’
```

```
var sAfterDecodeSep = decodeWithSeparator(sExampleValue,”);
```

```
sAfterDecodeSep:
例值 12, 例值 22, 例值 32, 例值 42, 例值 52, 例值 62
```

```
var sAfterDecodeSep =decodeWithSeparator(sExampleValue,”%”);
```

```
sAfterDecodeSep:
%例值 12%, %例值 22%, %例值 32%, %例值 42%, %例值 52%, %例值 62%
```

【语法】

decodeWithDefault(sStr) 等价于 decodeWithSeparator(sStr,”’”)

decodeWithDefault(*处理序列*)

或

decodeWithSeparator(*处理序列，“包括符”*)

```
/*decodeWithDefault
 *用默认的分隔符""将源字符串编码为格式：
 * A,B,C,D,E,F,G   ->   'A','B','C','D','E','F','G'
 * sStr: 源字符串：通过 fetchColumn()获得
 */
decodeWithDefault("");
```

```
/*decodeWithSeparator
 *用默认的分隔符""将源字符串编码为格式：
 * A,B,C,D,E,F,G   ->   'A','B','C','D','E','F','G'
 * sStr: 源字符串：通过 fetchColumn()获得
 * sSeparator:分隔符
 */
decodeWithSeparator("","");
```

【示例】

4.3.4 deleteColumn 与 deleteArrayColumn

【作用】

删除数据结果集的指定列。

【说明】

deleteColumn 处理系统默认的数据结果集

deleteArrayColumn 处理自定义的数据结果集

例值 11	例值 12	例值 13	例值 14
例值 21	例值 22	例值 23	例值 24
例值 31	例值 32	例值 22	例值 24
例值 41	例值 42	例值 43	例值 44
例值 51	例值 52	例值 53	例值 54

```
deleteColumn(2);
```

例值 11	例值 13	例值 14
例值 21	例值 23	例值 24
例值 31	例值 22	例值 24
例值 41	例值 43	例值 44
例值 51	例值 53	例值 54

【语法】

deleteColumn(指定列号)

或

deleteArrayColumn("自定义数据集名",指定列号)

```
/* deleteColumn
 * 删除指定列
 * sColumnList 列数数组
 *       列数 1,列数 2
 */
deleteColumn("1");
```

```
/* deleteArrayColumn
 * 删除指定列
 * sArrayName 数组名称
 * sColumnList 列数数组
 *       列数 1,列数 2
 */
deleteArrayColumn("","1");
```

【示例】

4.3.5 deleteRow 与 deleteArrayRow

【作用】

删除数据结果集的指定行。

【说明】

deleteRow 处理系统默认的数据结果集；

deleteArrayRow 处理自定义的数据结果集。

例值 11	例值 12	例值 13	例值 14
例值 21	例值 22	例值 23	例值 24
例值 31	例值 32	例值 22	例值 24
例值 41	例值 42	例值 43	例值 44
例值 51	例值 52	例值 53	例值 54

```
deleteRow (2);
```

例值 11	例值 12	例值 13	例值 14
例值 31	例值 32	例值 22	例值 24
例值 41	例值 42	例值 43	例值 44
例值 51	例值 52	例值 53	例值 54

【语法】

deleteRow （指定列号）

或

deleteArrayRow （“自定义数据集名”,指定列号）

```
/* deleteRow
 * 删除指定行
 * sRowList 行数数组
 *        行数 1,行数 2
 */
deleteRow("1");
```

```
/* deleteArrayRow
 * 删除指定行
 * sArrayName 数组名称
 * sRowList 行数数组
 *        行数 1,行数 2
 */
deleteArrayRow("","1");
```

【示例】

4.3.6 fetchColumn 与 fetchSetColumn

【作用】

配合『decodeWithDefault 与 decodeWithSeparator 使用』

fetchColumn(int iColumnNum)或 fetchSetColumn(int iColumnNum)

抽取结果集中的第 iColumnNum 列并转换为形如 "A|B|C|D" 的字符串以供 decodeWithDefault（decodeWithSeparator）加以转换使用。

【说明】

fetchColumn 处理系统默认的数据结果集；

fetchSetColumn 处理自定义的数据结果集。

例值 11	例值 12	例值 13	例值 14
例值 21	例值 22	例值 23	例值 24
例值 31	例值 32	例值 22	例值 24
例值 41	例值 42	例值 43	例值 44
例值 51	例值 52	例值 53	例值 54

```
var sExampleString = fetchColumn(2);
```

```
sExampleString :
     “例值 12|例值 22|例值 32|例值 42|例值 52”
```

【语法】

```
fetchColumn(指定列号);
```

或

```
fetchSetColumn(“自定义数据集名”,指定列号);
```

```
/*fetchColumn(i)
 *抽取第 i 列并转换为形如 "A|B|C|D" 的字符串
 *i ： 需要作为抽取的列
*/
fetchColumn(1);
```

```
/*fetchSetColumn(ArrayName,i)
 *从 ArrayNmae 中抽取第 i 列并转换为形如 "A|B|C|D"的字符串
 *i ： 需要作为抽取的列
 *ArrayName ：抽取对象数据集
*/
fetchSetColumn("",1);
```

【示例】

4.3.7 genResult 与 genArray

【作用】

通过指定的方式生成结果数据集，是生成源数据的基本操作。

目前只支持通过 SQL 语句的方式生成数据集。

【说明】

genResult 处理系统默认的数据结果集；

genArray 处理自定义的数据结果集。

```
var sQuerySQL = "SELECT col_Name1,… FROM tbl_Name
{WHERE …{GROUP BY … {ORDER BY …}}}";
```

//通过 SQL 的方式获取由 sQuerySQL 定义的结果数据集

genResult("SQL", sQuerySQL);

通过这种方式得到的结果是完全地将 SQL 结果转换成为了二维的结果数组。

【语法】

genResult("SQL","SQL 语句");

或

genArray("自定义数据集名","SQL","SQL 语句");

```
/* genResult
 * 获取数据
 * sSourceType 源数据类型
 * SQL: 从 SQL 语句获取数据
 * sSource 数据来源语句
 */
genResult("SQL","");
```

```
/* genArray
 * 获取数据
 * sArrayName 数组名称
 * sSourceType 源数据类型
 * SQL: 从 SQL 语句获取数据
 * sSource 数据来源语句
 */
genArray("","SQL","");
```

【示例】

4.3.8 getParam 与 getSheetParam

【作用】

『配合 setParam 和 setSheetParam 使用』

返回由 setParam 或者 setSheetParam 设置的局部（全局）变量值，如果获取之前没有设置相应变量名，则返回 null。

【说明】

getParam 返回局部变量的值；

getSheetParam 返回定义的全局变量值。

假设在另一元素（或本元素）中通过 setSheetParam(“ExampleValue”,“TestVaule”)设置了全局变量.

获取“ExampleValue”的变量值通过如下方式：

```
var sExampleGetValue = getSheetParam(“ExampleValue”);
```

sExampleGetValue 的值为：TestVaule。

【语法】

```
getParam(“局部变量名”);
```

或

```
getSheetParam(“全局变量名”);
```

```
/* getParam
 * 获取变量值
 * sParamName 变量名
 */
getParam("");
```

```
/* getSheetParam
 * 获取 Sheet 变量值
 * sParamName 变量名
 */
getSheetParam("");
```

【示例】

4.3.9 getProperty 与 getSheetProperty

【作用】

『配合 setProperty 和 setSheetProperty 使用』

返回由 setProperty 或者 setSheetProperty 设置的局部（全局）变量值，如果获取之前没有设置相应变量名，则返回 null。

【说明】

参看【4.3.8 getParam 与 getSheetParam】。

【语法】

```
getProperty(“局部属性名”);
```

或

```
getSheetProperty(“全局属性名”);
```

```
/* getProperty
 *  获取属性值
 * sPropertyName  变量名
 */
getProperty("");
```

```
/* getSheetProperty
 *  获取 Sheet 属性值
 * sPropertyName  变量名
 */
getSheetProperty("");
```

【示例】

4.3.10 getSQLValue

【作用】

返回所定义的 SQL 语句的第一行第一列(1,1)的字段值，其他字段值都将被忽略。

【说明】

```
var sSQL = "SELECT col_Name FROM tbl_Name {WHERE …}";
var sReturnValue = getSQLValue(sSQL);
```

【语法】

```
getSQLValue("SQL 语句");
```

```
/*getSQLValue(sSQL)
 *通过 SQL 的方式获得一个变量值
 *sSQL  数据获取 SQL 脚本
*/
getSQLValue("SELECT FROM ");
```

【示例】

4.3.11 insertBlankColumn 与 insertArrayBlankColumn

【作用】

在结果数据集中插入由“插数序列”指定的列号处插入一列，之中的值均为''，目的主要为扩充数据集，使之和报表的表样对应。

【说明】

insertBlankColumn 处理系统默认的数据结果集；

insertArrayBlankColumn 处理自定义的数据结果集。

（源数据集）

例值 11	例值 12	例值 13	例值 14
例值 21	例值 22	例值 23	例值 24
例值 31	例值 32	例值 22	例值 24
例值 41	例值 42	例值 43	例值 44
例值 51	例值 52	例值 53	例值 54

insertBlankColumn(“2,4,6”);

例值 11		例值 12		例值 13		例值 14
例值 21		例值 22		例值 23		例值 24
例值 31		例值 32		例值 22		例值 24
例值 41		例值 42		例值 43		例值 44
例值 51		例值 52		例值 53		例值 54

当遇到设置的列值超过了数据集自身的宽度时，数据集自动扩展其宽度：

insertBlankColumn(“2,4,8”);

例值 11		例值 12		例值 13	例值 14		
例值 21		例值 22		例值 23	例值 24		
例值 31		例值 32		例值 22	例值 24		
例值 41		例值 42		例值 43	例值 44		
例值 51		例值 52		例值 53	例值 54		

【语法】

insertBlankColumn(“插值序列”);

或

insertArrayBlankColumn(“自定义数据集名”,” 插值序列”);

```
/* insertBlankColumn(sColumnList)
 * 插入指定空列
 * sColumnList 列数数组
 *        列数 1,列数 2
 */
insertBlankColumn('','');
```

```
/* insertArrayBlankColumn(sArrayName,sColumnList)
 * 插入指定空列
 * sArrayName 数组名称
 * sColumnList 列数数组
 *        列数 1,列数 2
 */
insertArrayBlankColumn('','','');
```

【示例】

4.3.12 insertBlankRow 与 insertArrayBlankRow

【作用】

在结果数据集中插入由“插数序列”指定的行号处插入一行，之中的值均为''，目的主要为扩充数据集，使之和报表的表样对应。

与 4.3.11 类似。

【说明】

insertBlankRow 处理系统默认的数据结果集；

insertArrayBlankRow 处理自定义的数据结果集。

（源数据集）

例值 11	例值 12	例值 13	例值 14
例值 21	例值 22	例值 23	例值 24
例值 31	例值 32	例值 22	例值 24
例值 41	例值 42	例值 43	例值 44
例值 51	例值 52	例值 53	例值 54

```
insertBlankRow("2,4,6");
```

例值 11	例值 12	例值 13	例值 14
例值 21	例值 22	例值 23	例值 24
例值 31	例值 32	例值 22	例值 24
例值 41	例值 42	例值 43	例值 44
例值 51	例值 52	例值 53	例值 54

当遇到设置的列值超过了数据集自身的高度时，数据集自动扩展其宽度：

```
insertBlankRow ("2,7");
```

例值 11	例值 12	例值 13	例值 14
例值 21	例值 22	例值 23	例值 24
例值 31	例值 32	例值 22	例值 24
例值 41	例值 42	例值 43	例值 44
例值 51	例值 52	例值 53	例值 54

【语法】

insertBlankRow (“插值序列”);

或

insertArrayBlankRow(“自定义数据集名”,”插值序列”);

```
/* insertBlankRow
 * 插入指定空行
 * sColumnList 行数数组
 *        行数 1,行数 2
 */
insertBlankRow("","");
```

```
/* insertArrayBlankRow(sArrayName,sColumnList)
 * 插入指定空行
 * sArrayName 数组名称
 * sColumnList 行数数组
 *        行数 1,行数 2
 */
insertArrayBlankRow("","","");
```

【示例】

4.3.13 insertFormulaColumn 与 insertArrayFormulaColumn

【作用】

利用 EXCEL 控件自身的公式运算功能，根据指定的列号，插入一个符合 EXCEL 公式格式的**字符串**，实现数据集的横向数据数学求和处理。

专有变量**$CurrentRow** ：对于清单型的报表，实现横向多列求和时涉及的游标下移。就是公式中的“当前行”不是一个固定值，而是随着游标的移动而改变的。为此，提供的专有变量**$CurrentRow** 实现这一行变量，但是**$CurrentRow** 会受到 setPosition 的影响，如果 setPosition 属性设置在前，那么**$CurrentRow** 的初始值为指定的 position，但是若设置在后，则以默认的(0,0)作为起始坐标。

需要强调的是：这里涉及到两个比较坐标系的问题。数据集行列号坐标，是以数据集本身作为坐标的；而所插入的公式中的列号，是以 EXCEL 控件坐标系为基准的。

【说明】

insertFormulaColumn 处理系统默认的数据结果集；

insertArrayFormulaColumn 处理自定义的数据结果集。

（源数据）

例值 11	例值 12	例值 13	例值 14
例值 21	例值 22	例值 23	例值 24
例值 31	例值 32	例值 22	例值 24
例值 41	例值 42	例值 43	例值 44
例值 51	例值 52	例值 53	例值 54

```
setPosition(5,1);
insertFormulaColumn(2,"=A$ CurrentRow+C$ CurrentRow");
```

例值 11	=A5+C5	例值 12	例值 13	例值 14
例值 21	=A6+C6	例值 22	例值 23	例值 24
例值 31	=A7+C7	例值 32	例值 22	例值 24
例值 41	=A8+C8	例值 42	例值 43	例值 44
例值 51	=A9+C9	例值 52	例值 53	例值 54

系统本身不解析所生成的公式成为一个具体的值，而将计算工作由服务器端转移分配到每个客户端来执行。

【语法】

insertFormulaColumn(“公式插入列号”,”EXCEL 公式”);

或

insertArrayFormulaColumn(“自定义结果集名”,
“公式插入列号”,
”EXCEL 公式”);

```
/* insertFormulaColumn(iColumnIndex, sFormular)
 * 插入一列公式列
 * iColumnIndex 列数
 * sFormular Excel 公式
 *      $CurrentRow 当前的行数
 */
insertFormulaColumn(1,"=SUM(A$CurrentRow:B$CurrentRow)");
```

```
/* insertArrayFormulaColumn(sArrayName,iColumnIndex, sFormular)
 * 插入一列公式列
 * sArrayName 数组名称
 * iColumnIndex 列数
 * sFormular Excel 公式
 *      $CurrentRow 当前的行数
 */
insertArrayFormulaColumn("",1,"=SUM(A$CurrentRow:B$CurrentRow)");
```

【示例】

4.3.14 insertFormulaRow 与 insertArrayFormulaRow

【作用】

根据计算列列号指定，在数据集的底部增加一行数据，其中的数据组元素的值为插入的公式字符串。

专有变量**$RowCount**：数据集数据行数，针对于清单类报表，数据集的行数是个变动值，为了确定求和公式中的尾部数据所处的行值，引入**$RowCount** 变量。

参看 【4.3.14 作用项】

【说明】

insertFormulaRow 处理系统默认的数据结果集；

insertArrayFormulaRow 处理自定义的数据结果集。

（源数据）

例值 11	例值 12	例值 13	例值 14
例值 21	例值 22	例值 23	例值 24
例值 31	例值 32	例值 22	例值 24
例值 41	例值 42	例值 43	例值 44
例值 51	例值 52	例值 53	例值 54

```
setPosition(5,1);
insertFormulaRow(2,"=SUM(C5:C$ RowCount)");
```

例值 11	例值 12	例值 13	例值 14
例值 21	例值 22	例值 23	例值 24
例值 31	例值 32	例值 22	例值 24
例值 41	例值 42	例值 43	例值 44
例值 51	例值 52	例值 53	例值 54
	=SUM(C1:C10)		

【语法】

```
insertFormulaRow("公式插入列号","EXCEL 公式");
```

或

```
insertArrayFormulaRow("自定义结果集名",
                      "公式插入列号",
                      "EXCEL 公式");
```

```
/* insertFormulaRow(iColumnIndex,sFormular)
 *  末行插入公式列
 * iColumnIndex  列数
 * sFormular Excel 公式
 *        $RowCount  记录数
 */
insertFormulaRow(1,"SUM(C1:C$RowCount)");
```

```
/* insertArrayFormulaRow(sArrayName,iColumnIndex,sFormular)
 *  末行插入公式列
 * sArrayName  数组名称
 * iColumnIndex  列数
 * sFormular Excel 公式
 *        $RowCount  记录数
 */
insertArrayFormulaRow("",1,"SUM(C1:C$RowCount)");
```

『更好的解决方案』：相比使用 insertFormulaRow 或 insertArrayFormulaRow，可以利用 EXCEL 自身的自动扩展功能来实现纵向的求和，无论从效果还是处理速度上考虑都是一个很好的方法。

将清单列表删除到只剩下 3 行，前两行是填充数据，第 3 行是写公式的。当数据往里填充时，之前写的公式会自动地改变下端点值（如下图中的 B5）。

【示例】

4.3.15 matchColumnRowByArray 与 matchArrayColumnRowByArray

【作用】

根据指定的横纵匹配序列重新组合数据集，目前只支持单值排列。

【说明】

matchColumnRowByArray 与 matchArrayColumnRowByArray 调整之后不建议继续使用。

参看【4.3.19】

【语法】

```
matchColumnRowByArray(用于纵向匹配的数据集列号,
                      "纵向匹配序列",
                      用于横向匹配的数据集列号,
                      "横向匹配序列");
```

或

```
matchArrayColumnRowByArray("自定义数据集名",
                      用于纵向匹配的数据集列号,
                      "纵向匹配序列",
                      用于横向匹配的数据集列号,
                      "横向匹配序列");
```

```
/* 方法已经升级，不建议继续使用
 * matchColumnRowByArray(iColumn1,sMatchArray1,iColumn2,sMatchArray2)
 * 根据数组进行列、行匹配
 * iColumn1,iColumn2 列数
 * sMatchArray1,sMatchArray2 匹配数组
 *       值 1,值 2
 */
matchColumnRowByArray(1,",",2,",");
```

```
/* 方法已经升级，本方法不建议继续使用
 * matchArrayColumnRowByArray
 * 根据数组进行列、行匹配
 * sArrayName 数组名称
 * iColumn1,iColumn2 列数
 * sMatchArray1,sMatchArray2 匹配数组
 *        值 1,值 2
 */
matchArrayColumnRowByArray("",1,",",2,",");
```

【示例】

4.3.16 matchColumnRowBySQL 与 matchArrayColumnRowBySQL

【作用】

根据指定的横纵匹配序列重新组合数据集，目前只支持单值排列。

【说明】

matchColumnRowBySL 与 matchArrayColumnRowBySQL 调整之后不建议继续使用。

参看【4.3.20】

【语法】

```
matchColumnRowBySQL(用于纵向匹配的数据集列号,
                    "纵向匹配序列 SQL",
                    用于横向匹配的数据集列号,
                    "横向匹配序列");
```

或

```
matchArrayColumnRowBySQL("自定义数据集名",
                    用于纵向匹配的数据集列号,
                    "纵向匹配序列 SQL",
                    用于横向匹配的数据集列号,
                    "横向匹配序列");
```

```
/* 方法已经升级，本方法不建议继续使用
 * matchColumnRowBySQL(iColumn1, sMatchArray1,iColumn2, sMatchArray2)
 * 根据 SQL 语句进行列、行匹配
 * iColumn1,iColumn2 列数
 * sMatchArray1,sMatchArray2 匹配数组
 *       值 1,值 2
 */
matchColumnRowBySQL(1,"",2,"");
```

```
/* 方法已经升级，本方法不建议继续使用
 * matchArrayColumnRowBySQL(sArrayName,
                                iColumn1, sMatchArray1,iColumn2, sMatchArray2)
 * 根据 SQL 语句进行列、行匹配
 * sArrayName 数组名称
 * iColumn1,iColumn2 列数
 * sMatchArray1,sMatchArray2 匹配数组
 *       值 1,值 2
 */
```

【示例】

4.3.17 matchRowByArray 与 matchArrayRowByArray

【作用】

根据“*纵向匹配序列*”匹配扩充结果数据集。

简单地说，就是“按顺序整理存在的，扩充没有的”。

【说明】

matchRowByArray 处理系统默认的数据结果集；

matchArrayRowByArray 处理自定义的数据结果集。

（源数据）

A	例值 11	例值 12	例值 13	例值 14
B	例值 21	例值 22	例值 23	例值 24
G	例值 31	例值 32	例值 22	例值 24
C	例值 41	例值 42	例值 43	例值 44
E	例值 51	例值 52	例值 53	例值 54

matchRowByArray(1,”A,B,C,D,E,F,G”);

A	例值 11	例值 12	例值 13	例值 14
B	例值 21	例值 22	例值 23	例值 24
C	例值 41	例值 42	例值 43	例值 44
D				
E	例值 51	例值 52	例值 53	例值 54
F				
G	例值 31	例值 32	例值 22	例值 24

在经过了纵向指定序列匹配之后，源数据集的数据以第一列作为标尺按照给出的纵向序列**”A,B,C,D,E,F,G”**做了排列顺序的调整及针对数据缺失项做了数据集的扩充。

matchRowByArray(1,”A,B,G,D,F,C”);

A	例值 11	例值 12	例值 13	例值 14
B	例值 21	例值 22	例值 23	例值 24
G	例值 31	例值 32	例值 22	例值 24
D				
F				
C	例值 41	例值 42	例值 43	例值 44

通过纵向的匹配，可以达到当查询结果数据涵盖面不全时出现的数据显示位置错位的情况，同时也能根据报表表样需求控制数据行的排列顺序。

【语法】

matchRowByArray(数据集匹配标尺列,"纵向匹配基准序列");

或

matchArrayRowByArray("自定义结果数据集",

数据集匹配标尺列,"纵向匹配基准序列");

```
/* matchRowByArray(iColumn,sMatchArray)
 * 根据数组进行列匹配,X、Y 数据对换
 * iColumn 列数
 * sMatchArray 匹配数组
 *       值 1,值 2
 */
matchRowByArray(1,",");
```

```
/* matchArrayRowByArray
 * 根据数组进行列匹配,X、Y 数据对换
 * sArrayName 数组名称
 * iColumn 列数
 * sMatchArray 匹配数组
 *       值 1,值 2
 */
matchArrayRowByArray("",1,",");
```

【注意】

只所以能进行匹配，匹配列必须存在且有相应的对应关系，没有对应关系的序列和结果集进行匹配返回只有匹配列值的结果集。

【示例】

4.3.18 matchRowBySQL 与 matchArrayRowBySQL

【作用】

根据“*纵向匹配 SQL 结果序列*”匹配扩充结果数据集。

与【4.3.17】相比，纵向匹配序列通过 SQL 语句产生。

【说明】

matchRowBySQL 处理系统默认的数据结果集；

matchArrayRowBySQL 处理自定义的数据结果集。

（源数据）

机构 1	例值 11	例值 12	例值 13	例值 14
机构 3	例值 21	例值 22	例值 23	例值 24
机构 4	例值 31	例值 32	例值 22	例值 24
机构 5	例值 41	例值 42	例值 43	例值 44
机构 8	例值 51	例值 52	例值 53	例值 54

```
var sFilterSql = "SELECT OrgID "
               +"FROM ORG_INFO "
               +"{WHERE … }"
               +"{ORDER BY OrgID}";
matchRowBySQL(1, sFilterSql);
```

（假设执行 sFilterSql 得到的结果是"机构 1,机构 2,…,机构 N"）

机构 1	例值 11	例值 12	例值 13	例值 14
机构 2				
机构 3	例值 21	例值 22	例值 23	例值 24
机构 4	例值 31	例值 32	例值 22	例值 24
机构 5	例值 41	例值 42	例值 43	例值 44
机构 6				
机构 7				
机构 8	例值 51	例值 52	例值 53	例值 54
机构 9				
机构…				

参看【4.3.17】

【语法】

matchRowBySQL(匹配列号,”基准列 SQL”);

或

matchArrayRowBySQL(“自定义结果集”,匹配列号,”基准列 SQL”);

```
/* matchRowBySQL(iColum, sSQL)
 * 根据数组进行列匹配,X、Y 数据对换
 * iColumn 列数
 * sSQL SQL 语句
 */
matchRowByArray(1,"SELECT  FROM ");
```

```
/* matchArrayRowBySQL
 * 根据数组进行列匹配,X、Y 数据对换
 * sArrayName 数组名称
 * iColumn 列数
 * sSQL SQL 语句
 */
matchArrayRowBySQL("",1,"SELECT  FROM ");
```

【示例】

4.3.19 matchRowByArrayColumnByArray 与 matchSetRowByArrayColumnByArray

【作用】

根据通过 Array 方式生成的指定匹配**行序列**值匹配扩充行，根据通过 Array 方式生成的指定匹配**列序列**值匹配扩充列。

主要使用在横纵都需要进行匹配的情况。

【说明】

matchRowByArrayColumnByArray 处理系统默认的数据结果集；

matchSetRowByArrayColumnByArray 处理自定义的数据结果集。

（源数据）

A	a	例值 1
A	b	例值 2
A	c	例值 3
B	b	例值 4
B	c	例值 5
C	a	例值 6
C	c	例值 7
D	a	例值 8
D	d	例值 9
E	b	例值 10

matchRowByArrayColumnByArray (1,"A,B,C,D,E,F",2,"a,b,c,d");

	a	b	c	d
A	例值 1	例值 2	例值 3	
B		例值 4	例值 5	
C	例值 6		例值 7	
D	例值 8			例值 9
E		例值		
F				

行序列："a,b,c,d"

列序列："A,B,C,D,E,F"

得到的结果是以列序列为纵向列值，而以行序列为横向行值的一个新的结果数据集。

【语法】

matchRowByArrayColumnByArray(

数据集匹配列 1,"纵向匹配基准序列"

数据集匹配列 2,"横向匹配基准序列");

或

matchSetRowByArrayColumnByArray("自定义结果数据集",

数据集匹配列 1,"纵向匹配基准序列"

数据集匹配列 2,"横向匹配基准序列");

```
/**matchRowByArrayColumnByArray(iColumn1,sMatchArray1,
 *                                iColumn2,sMatchArray2)
 * 纵向通过 Array 的方式、横向通过 Array 方式匹配数据结果集。
 * iColumn1: 与 sMatchArray1 相匹配的数据集列数
 * sMatchArray1: 纵向匹配基准序列
 * iColumn2: 与 sMatchArray2 相匹配的数据集列数
 * sMatchArray2: 横向匹配基准序列
 */
matchRowByArrayColumnByArray(1,",",2,",");
```

```
/**matchSetRowByArrayColumnByArray(sArrayName,
 *                                iColumn1,sMatchArray1,
 *                                iColumn2,sMatchArray2)
 * 纵向通过 Array 的方式、横向通过 Array 方式匹配数据结果集。
 * sArrayName: 自定义数据集名称
 * iColumn1: 与 sMatchArray1 相匹配的数据集列数
 * sMatchArray1: 纵向匹配基准序列
 * iColumn2: 与 sMatchArray2 相匹配的数据集列数
 * sMatchArray2: 横向匹配基准序列
 */
matchSetRowByArrayColumnByArray("",1,",",2,",");
```

【示例】

4.3.20 matchRowByArrayColumnBySQL 与 matchSetRowByArrayColumnBySQL

【作用】

根据通过 Array 方式生成的指定匹配**行序列**值匹配扩充行，根据通过 SQL 方式生成的指定匹配**列序列**值匹配扩充列。

使用在纵向纬度固定，横向变动的匹配情况下。

【说明】

matchRowByArrayColumnBySQL 处理系统默认的数据结果集；

matchSetRowByArrayColumnBySQL 处理自定义的数据结果集。

【参看 4.3.18】

【参看 4.3.19】

【语法】

matchRowByArrayColumnBySQL(

数据集匹配列 1,"纵向匹配基准序列"

数据集匹配列 2,"横向匹配基准序列");

或

matchSetRowByArrayColumnBySQL("自定义结果数据集",

数据集匹配列 1,"纵向匹配基准序列"

数据集匹配列 2,"横向匹配基准序列");

```
/**matchRowByArrayColumnBySQL(iColumn1,sMatchArray,
 *                              iColumn2,sMatchSQL)
 * 纵向通过 Array 的方式、横向 SQL 通过方式匹配数据结果集。
 * iColumn1: 与 sMatchArray 相匹配的数据集列数
 * sMatchArray: 纵向匹配基准序列
 * iColumn2: 与 sMatchSQL 相匹配的数据集列数
 * sMatchSQL: 横向匹配基准序列
 */
matchRowByArrayColumnBySQL(1,",",2,"SELECT FROM ");
```

```
/**matchSetRowByArrayColumnBySQL(sArrayName,
 *                                  iColumn1,sMatchArray,
 *                                  iColumn2,sMatchSQL)
 *  纵向通过 Array 的方式、横向通过 SQL 方式匹配数据结果集。
 * sArrayName: 自定义数据集名称
 * iColumn1: 与 sMatchArray 相匹配的数据集列数
 * sMatchArray1: 纵向匹配基准序列
 * iColumn2: 与 sMatchSQL 相匹配的数据集列数
 * sMatchSQL: 横向匹配基准序列
 */
matchSetRowByArrayColumnBySQL("",1,"",2,"SELECT FROM ");
```

【示例】

4.3.21 matchColumnBySQLRowByArray 与 matchSetColumnBySQLRowByArray

【作用】

根据通过 SQL 方式生成的指定匹配**行序列**值匹配扩充行，根据通过 Array 方式生成的指定匹配**列序列**值匹配扩充列。

使用在纵向纬度固定，横向变动的匹配情况下。

【说明】

matchColumnBySQLRowByArray 处理系统默认的数据结果集；

matchSetColumnBySQLRowByArray 处理自定义的数据结果集。

【参看 4.3.18】

【参看 4.3.19】

【语法】

```
matchColumnBySQLRowByArray(
                                数据集匹配列 1,"纵向匹配基准序列"
                                数据集匹配列 2,"横向匹配基准序列");
```

或

```
matchSetColumnBySQLRowByArray("自定义结果数据集",
                                数据集匹配列 1,"纵向匹配基准序列"
                                数据集匹配列 2,"横向匹配基准序列");
```

```
/**matchRowBySQLColumnByArray(iColumn1,sMatchSQL,
 *                                  iColumn2,sMatchArray)
 * 纵向通过 SQL 的方式、横向通过 Array 方式匹配数据结果集。
 * iColumn1: 与 sMatchArray 相匹配的数据集列数
 * sMatchSQL: 纵向匹配基准序列
 * iColumn2: 与 sMatchSQL 相匹配的数据集列数
 * sMatchArray: 横向匹配基准序列
 */
matchRowBySQLColumnByArray(1,"SELECT FROM ",2,",");
```

```
/**matchSetRowBySQLColumnByArray(sArrayName,
 *                                  iColumn1,sMatchSQL,
 *                                  iColumn2,sMatchArray)
 *  纵向通过 SQL 的方式、横向通过 Array 方式匹配数据结果集。
 * sArrayName: 自定义数据集名称
 * iColumn1: 与 sMatchSQL 相匹配的数据集列数
 * sMatchSQL: 纵向匹配基准序列
 * iColumn2: 与 sMatchArray 相匹配的数据集列数
 * sMatchArray2: 横向匹配基准序列
 */
matchSetRowBySQLColumnByArray("",1,"SELECT FROM ",2,",");
```

【示例】

4.3.22 matchColumnBySQLRowBySQL 与 matchSetColumnBySQLRowBySQL

【作用】

根据通过 SQL 方式生成的指定匹配**行序列**值匹配扩充行，根据通过 SQL 方式生成的指定匹配**列序列**值匹配扩充列。

使用在纵向纬度固定，横向变动的匹配情况下。

【说明】

matchColumnBySQLRowBySQL 处理系统默认的数据结果集；

matchSetColumnBySQLRowBySQL 处理自定义的数据结果集。

【参看 4.3.18】

【参看 4.3.19】

【语法】

matchColumnBySQLRowBySQL(

数据集匹配列 1,"纵向匹配基准序列"

数据集匹配列 2,"横向匹配基准序列");

或

matchSetColumnBySQLRowBySQL("自定义结果数据集",

数据集匹配列 1,"纵向匹配基准序列"

数据集匹配列 2,"横向匹配基准序列");

```
/**matchRowBySQLColumnBySQL(iColumn1,sMatchSQL1,
 *                          iColumn2,sMatchSQL2)
 * 纵向通过 SQL 的方式、横向通过 SQL 方式匹配数据结果集。
 * iColumn1: 与 sMatchSQL1 相匹配的数据集列数
 * sMatchSQL1: 纵向匹配基准序列
 * iColumn2: 与 sMatchSQL2 相匹配的数据集列数
 * sMatchSQL2: 横向匹配基准序列
 */
matchRowBySQLColumnBySQL(1,"SELECT FROM ",2,"SELECT FROM ");
```

```
/**matchSetRowBySQLColumnBySQL(sArrayName,
 *                                iColumn1,sMatchSQL1,
 *                                iColumn2,sMatchSQL2)
 *  纵向通过 Array 的方式、横向通过 Array 方式匹配数据结果集。
 * sArrayName: 自定义数据集名称
 * iColumn1: 与 sMatchSQL1 相匹配的数据集列数
 * sMatchSQL1: 纵向匹配基准序列
 * iColumn2: 与 sMatchSQL2 相匹配的数据集列数
 * sMatchSQL2: 横向匹配基准序列
 */
matchSetRowBySQLColumnByArray("",1,"SELECT    FROM    ",2,"SELECT
```

【示例】

4.3.23 rotateResult 与 rotateArray

【作用】

将数据集进行翻转，通过“旋转方向”来指定翻转反向。

【说明】

rotateResult 处理系统默认的数据结果集；

rotateArray 处理自定义的数据结果集。

（源数据）

A	例值 11	例值 12	例值 13
B	例值 21	例值 22	例值 23
C	例值 31	例值 32	例值 22
D	例值 41	例值 42	例值 43
E	例值 51	例值 52	例值 53

```
rotateResult(1);          //顺时针 90°
```

E	D	C	B	A
例值 51	例值 41	例值 31	例值 21	例值 11
例值 52	例值 42	例值 32	例值 22	例值 12
例值 53	例值 43	例值 33	例值 23	例值 13

```
rotateResult(2);          //顺时针 180°
```

例值 53	例值 52	例值 51	E
例值 43	例值 42	例值 41	D
例值 33	例值 32	例值 31	C
例值 23	例值 22	例值 21	B
例值 13	例值 12	例值 11	A

```
rotateResult(3);          //顺时针 270°（逆时针 90°）
```

例值 13	例值 23	例值 33	例值 43	例值 53
例值 12	例值 22	例值 32	例值 42	例值 52
例值 11	例值 21	例值 31	例值 41	例值 51
A	B	C	D	E

【语法】

rotateResult(”旋转方向”);

或

rotateArray(“自定义数据集”,”旋转方向”);

```
/* rotateArray(sArrayName,iDirectType)
 * 进行旋转排列
 * sArrayName 数组名称
 * iDirectType 旋转方向(以第一行、第一列为原点)
 *       1: 顺时针 90°
 *       2: 顺时针 180°
 *       3: 顺时针 270°
 */
rotateArray("",1);
```

```
/* rotateResult(iDirectType)
 * 进行旋转排列
 * iDirectType 旋转方向(以第一行、第一列为原点)
 *       1: 顺时针 90°
 *       2: 顺时针 180°
 *       3: 顺时针 270°
 */
rotateResult(1);
```

【示例】

4.3.24 uniteResult 与 uniteArray

【作用】

将指定的两个结果集通过指定的匹配列进行匹配之后存放在新的结果集中。

【说明】

`uniteResult` 处理系统默认的数据结果集；

`uniteArray` 处理自定义的数据结果集。

（源数据 1：ExampleSet1）

A	例值 11	例值 12	例值 13
B	例值 21	例值 22	例值 23
C	例值 31	例值 32	例值 22
D	例值 41	例值 42	例值 43
E	例值 51	例值 52	例值 53

（源数据 2：ExampleSet2）

A	例值 aa	例值 ab
C	例值 ba	例值 bb
F	例值 ca	例值 cb
D	例值 da	例值 db

```
uniteResult(ExampleSet1,1, ExampleSet2,1);
```

A	例值 11	例值 12	例值 13	例值 aa	例值 ab
B	例值 21	例值 22	例值 23		
C	例值 31	例值 32	例值 22	例值 ba	例值 bb
D	例值 41	例值 42	例值 43	例值 da	例值 db
E	例值 51	例值 52	例值 53		

经过两个数据集的匹配组合之后，源数据 2（ExampleSet2）的数据按照匹配列与源数据 1（ExampleSet1）的匹配列将两数据集进行匹配组合。

对于存在对应关系的数据将追加在源数据 1 之后，而 ExampleSet2 中存在的数据没有相应的匹配位于 ExampleSet1 中则会被滤除，如 ExampleSet2 中的 F 行

F	例值 ca	例值 cb

对于基准数据集（如上例中的 ExampleSet1）中存在而没有反映在匹配数据集如 ExampleSet2 中的数据行之后将补满空字符串（""），如最终结果中的 B 行和 E 行。

【语法】

```
uniteResult("结果集 1","结果集 1 匹配列",
            "结果集 2","结果集 2 匹配列",
            "类型");
```

或

```
uniteArray("自定义结果集",
            "结果集 1","结果集 1 匹配列",
            "结果集 2","结果集 2 匹配列",
            "类型");
```

```
/* uniteResult
 * 匹配合并结果集到最终结果集 Result
 * 结果集 1：          sArrayName1
 * 结果集 1 匹配列：iCol1
 * 结果集 2：          sArrayName2
 * 结果集 2 匹配列：iCol2
 * 类型：              iType(目前为 1)
*/
uniteResult("","","","","")
```

```
/* uniteArray
 * 匹配合并结果集到 sArrayName
 * 结果集名称          sArrayName
 * 结果集 1：          sArrayName1
 * 结果集 1 匹配列：iCol1
 * 结果集 2：          sArrayName2
 * 结果集 2 匹配列：iCol2
 * 类型：              iType(目前为 1)
*/
uniteArray("","","","","","","")
```

【示例】

4.4 其他说明

4.4.1 println

【作用】

将文本内容显示在展示页面上。

【说明】

【语法】

println(展示文本);

```
/* println
 * 向页面输出文本
 * sText 文本内容
 */
println("");
```

【示例】

```
println("Println Example.");
```

Result: Println Example.

```
var sExample = "Another Println Example.";
println(sExample);
```

Result: Another Println Example.

```
println("This is "+ sExample);
```

Result: This is Another Example.

5 涉及的 DBFunciton 说明

5.1 简述

在 SQL 数据查询过程中，将多个分组结果统计在一个分类中，在 ARS 中称之为“归并统计”。

如：要将分组结果”A”,”B”,”C”统计在一个新的分类中，通过 ReportStudio 中的“代码转换”，将”A”,”B”,”C”三个输入值均返回同一新的分类值即可实现我们的需求。

为了便于集合/分组/归并统计，针对纬度和度量，分别提供一部分数据库函数加以使用，以提高数据查询的便捷性。

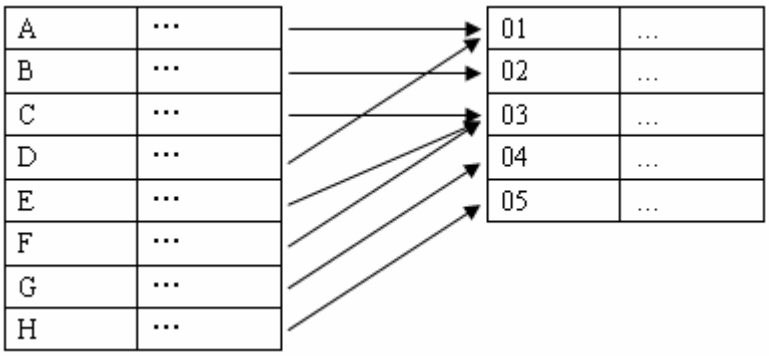
5.2 纬度

5.2.1 getConvertCode

【作用】

将”代转换代码”经过转换后返回设定值，通常使用在分组分类的归并统计上。

getConvertCode 提供的是单一等值的转换。



【说明】

getConvertCode(“调用代码名称”,”待转换代码”,”默认返回代码”);

```
getConvertCode(pModelItemID,pInitCode,pValue)
/* getConvertCode
 *  通过等值获取"代码转换定义"中定义设置返回相应的值。
 * pModelItemID : 转换代码名称。
 * pInitCode : 传入的待转换值。
 * pValue : 默认值，当在代码设置中没有返回值时返回该值。
 */
```

【示例】

（SQL 产生的直接源数据）

ColName1	ColName2	ColName3	ColName4
A	例值 11	例值 12	例值 13
B	例值 21	例值 22	例值 23
C	例值 31	例值 32	例值 22
D	例值 41	例值 42	例值 43
E	例值 51	例值 52	例值 53

报表表样需求：”A”和”C”是归并在同一项中。

方法 1

在 ReportStudio 中“代码转换”作如下设置：

代码名称	传入值/起始值	转换后值
'Example'	A	01
'Example'	B	02
'Example'	C	01
'Example'	D	03
'Example'	E	04

在 SQL 语句中调用 getConvertCode 改变分组 Group 值，没有在代码设置中出现的值默认返回’01’：

```
SELECT getConvertCode('Example', ColName1,’01’) AS Code,…
FROM …
{WHERE …}
{GROUP BY Code}
```

在方法 1 中，情形”A”和”C”归并在”01”中，其他的非常规情形可以自由归类（如”09”），或者不予处理。

方法 2

在 ReportStudio 中“代码转换”作如下设置：

代码名称	传入值/起始值	转换后值
'Example'	B	02
'Example'	D	03
'Example'	E	04

```
SELECT getConvertCode('Example', ColName1,’01’) AS Code,…
FROM …
{WHERE …}
{GROUP BY Code}
```

方法 2 情况下，情形”A”、”C”和其他非常规情况都归并到’01’中了，与方法 1 相比区别是不能另外处理非常规情形。

5.2.2 getConvertCode2

【作用】

与 getConvertCode 处理单一等值相比，getConvertCode2 针对所属区间对应返回值的转换。

【说明】

`getConvertCode2(“调用代码名称”,”待转换代码”,”默认返回代码”);`

```
getConvertCode2(pModelItemID,pInitCode,pValue)
/* getConvertCode2
 *  通过区间获取"代码转换定义"中定义设置返回相应的值。
 * pModelItemID：转换代码名称。
 * pInitCode：传入的待转换值。
 * pValue：默认值，当在代码设置中没有返回值时返回该值。
 */
```

参看【5.2.1】

【示例】

（SQL 产生的直接源数据）

ColName1	ColName2	ColName3	ColName4
15	例值 11	例值 12	例值 13
20	例值 21	例值 22	例值 23
35	例值 31	例值 32	例值 22
50	例值 41	例值 42	例值 43
70	例值 51	例值 52	例值 53

报表表样需求：　0～15(包含)　归并为一类；

15～30(包含) 归并为一类；

30～50(包含) 归并为一类；

＞50　　归并为一类。

此情形例如“按贷款期限”或“按逾期时间”作为纬度的报表统计。

方法 1

在 ReportStudio 中“代码转换”作如下设置：

代码名称	传入值/起始值	终止值	转换后值
'Example'	0	15	01
'Example'	15	30	02
'Example'	30	50	03

在 SQL 语句中调用 getConvertCode 改变分组 Group 值，没有在代码设置中出现的值默认返回'04'：

```
SELECT getConvertCode('Example', ColName1,'04') AS Code,…
FROM …
{WHERE …}
{GROUP BY Code}
```

所有值大于 50 的值都归并在"04"中，其中也包括其他的非常规情形。

方法 2，

在 ReportStudio 中“代码转换”作如下设置：

代码名称	传入值/起始值	终止值	转换后值
'Example'	0	15	01
'Example'	15	30	02
'Example'	30	50	03
'Example'	50	(一个足够大的值)	04

```
SELECT getConvertCode('Example', ColName1,'09') AS Code,…
FROM …
{WHERE …}
{GROUP BY Code}
```

方法 2 情况下，是利用了出现概率小的情形实现需求，虽然几乎都可以达到所要的结果，但很明显存在逻辑上的漏洞而不建议使用。

5.3 度量

5.3.1 isBetween

【作用】

判断传入数值在指定的封闭类型（Type）下是否存在一个区间中，满足条件返回整数 1，否则返回整数 0。

【说明】

isBetween(传入待比较值,封闭类型,区间起点,区间终点);

```
ISBETWEEN(pSRCValue,pType,pValue1,pValue2)
/* 判断一个值是否包含在一个期间内
 * 左右包含方式 由 pType 指定
 * pType ： '00' 左开右开 ( )
 *          '01' 左闭右开 [ )
 *          '02' 左开右闭 ( ]
 *          '03' 左闭右闭 [ ]
 */
```

【示例】

（报表表样需求）

	逾期贷款余额	
	0～6 个月	6 个月～1 年
农业		
工业		
…		

实现类似的需求只要能做到判断逾期时间归属哪个分类即可实现。

方法

```
SELECT SUBSTR(IndustryType,1,1) AS IndustryType,
   isBetween(OverTermMonth,'02',0,6) * Balance AS Balance1,
   isBetween(OverTermMonth,'02',6,12) * Balance AS Balance2
FROM …
{WHERE …}
GROUP BY 1
ORDER BY 1;
```

5.3.2 isLarger

【作用】

判断传入数值是否大于基准比较值中，满足条件返回整数 1，否则返回整数 0。

【说明】

isLarger(传入待比较值,比较基准值);

```
ISLARGER(pValue1,pValue2)
/* 判断 pValue1 是否大于 pValue2
 *     如是，返回 1
 *     若否，返回 0
 */
```

【示例】

（报表表样需求）

	贷款期限		
	0～6 个月	6 个月～1 年	1 年以上
农业			
工业			
…			

实现类似的需求只要能做到判断贷款期限归属哪个分类即可实现。

方法

```
SELECT SUBSTR(IndustryType,1,1) AS IndustryType,
   isBetween(TermMonth,'02',0,6) * Balance AS Balance1,
   isBetween(TermMonth,'02',6,12) * Balance AS Balance2,
   isLarger(TermMonth,12) * Balance AS Balance3
FROM …
{WHERE …}
GROUP BY 1
ORDER BY 1
;
```

5.3.3 isLesser

【作用】

判断传入数值是否小于基准比较值中，满足条件返回整数 1，否则返回整数 0。

【说明】

isLesser(传入待比较值,比较基准值);

```
ISLESSER(pValue1,pValue2)
/* 判断 pValue1 是否小于 pValue2
 *     如是，返回 1
 *     若否，返回 0
 */
```

【示例】

（报表表样需求）

	贷款期限		
	6 个月以下	6 个月～1 年	1 年以上
农业			
工业			
…			

实现类似的需求只要能做到判断贷款期限归属哪个分类即可实现。

方法

```
SELECT SUBSTR(IndustryType,1,1) AS IndustryType,
    isLesser(TermMonth, 6) * Balance AS Balance1,
    isBetween(TermMonth,'02',6,12) * Balance AS Balance2,
    isLarger(TermMonth,12) * Balance AS Balance3
FROM …
{WHERE …}
GROUP BY 1
ORDER BY 1
;
```

5.3.4 isLike

【作用】

判断传入值是否包含有基准比较值，满足条件返回整数 1，否则返回整数 0。

如：isLike('01001','010') 返回 1

isLike('01001','011') 返回 0

【说明】

isLike(传入待比较值,比较基准值);

```
ISLIKE(pValue1,pValue2)
/* 判断两个值是否相似
 * 如果 pValue1 包含 pValue2 则返回 1
 * 相反 返回 0
 */
```

【示例】

（报表表样需求）

	信用	保证	抵押	质押
农业				
工业				
…				

实现类似的需求只要能做到判断担保类型归属哪个分类即可实现。

方法

```
SELECT SUBSTR(IndustryType,1,1) AS IndustryType,
    isLike(VouchType,'01') * Balance AS Balance1,
    isLike(VouchType,'02') * Balance AS Balance2,
    isLike(TermMonth,'03') * Balance AS Balance3,
    isLike(VouchType,'04') * Balance AS Balance4
FROM …
{WHERE …}
GROUP BY 1
ORDER BY 1;
```

5.3.5 isNotNull

【作用】

判断传入的字段值是否不为空，若是，返回 1，否则，返回 0。

【说明】

```
isNotNull("传入字段值");
```

【示例】