教程

1. 集算器说明

1.1 集算器功能简介

集算器能够完成批量数据计算，它具有以下的一些基本功能：

- ✧ 数据分析与结构化计算
- ✧ 自由访问数据库，在线数据分析

1.2 集算器安装与运行

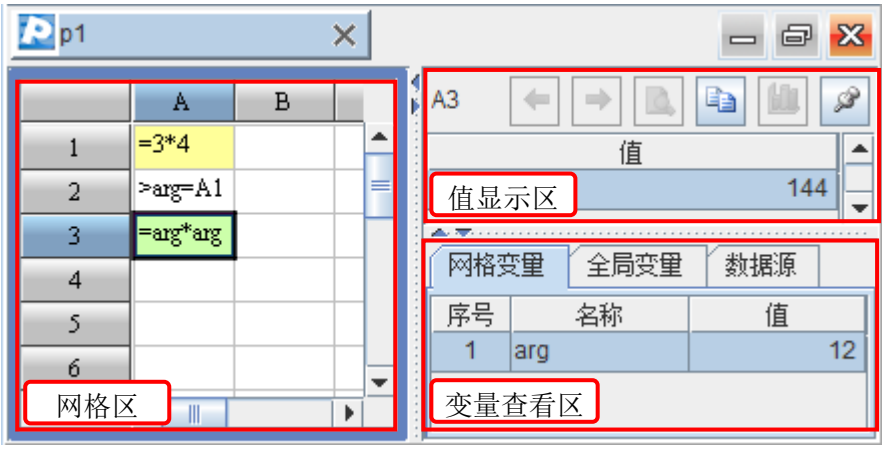
拿到安装包后执行安装程序，按提示逐步进行：

1) 运行安装程序
2) 按照提示，依次点击下一步，接受许可证协议
3) 选择安装路径，点击安装
4) 完成安装。

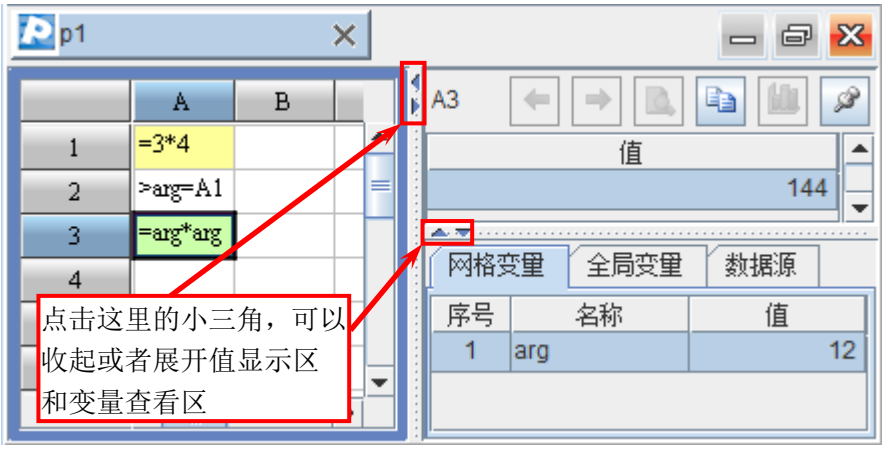
如果对 Java 的运行环境配置比较了解，而且本地已经安装了 JDK1.5 以上版本，也可以选择不自动安装 JDK 的集算器安装包来安装，在安装中需要填写本机 JDK 所在目录。

1.3 界面布局及网格文件的创建

运行集算器主程序打开**集算器标准版**，点击  按钮可以**创建网格文件**。



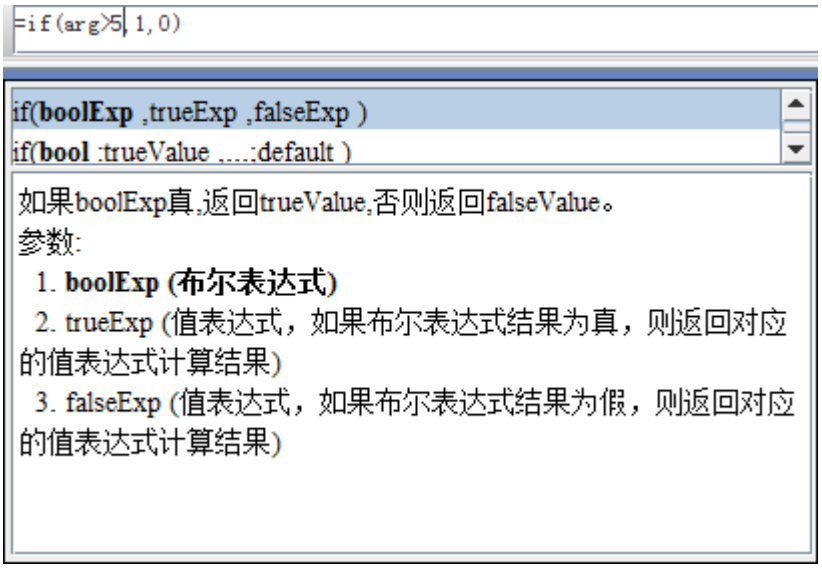
左边的**网格区**是当前活动的网格文件。右边，上部分区域是**值显示区**，下部分区域是**变量查看区**。值显示区和变量查看区可以收起和展开。



1.4 集算器中的函数编辑辅助功能

在集算器中，提供了大量函数。在单元格中书写表达式时，可以通过函数编辑辅助功能，更方便地进行书写。

在函数编辑辅助界面中，可以看到函数的语法以及函数说明等信息：



按住 Alt+↓可以打开/关闭函数编辑辅助功能。

2. 集算器基础操作

2.1 集算器中的数据类型

2.1.1 数据类型

在集算器中，有以下几种数据类型：

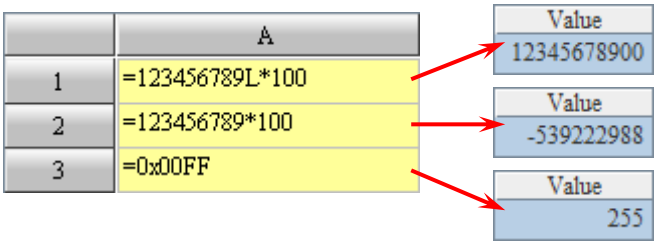
- **整数**

-2^{31}~2^{31}-1 之间的整数，即取值范围-2147483648~2147483647，可用类型转换函数 **int()** 将其它类型数据转换为整数。

- **长整数**

-2^{63}~2^{63}-1，比整数类型的取值范围更大，可用类型转换函数 **long()**将其它类型数据转换为长整数。

特别的，可以在整数后面添加大写的 **L** 表示长整数，长整数与整数相比有更大的取值范围；还可以用 **0x** 开头的串来表示十六进制的长整数：



由于普通整数的取值范围是-2^{31}~2^{31}-1，即-2147483648~2147483647，因此 A2 中的结果超出了整数的取值范围。而 A1 中，使用了长整数，取值范围增大为-2^{63}~2^{63}-1，就可以获得正确结果。可以注意到，在进行某一步运算时，当参与运算的操作数之一为长整数，结果就是长整数。

- **浮点数**

64 位浮点数，这是集算器中最常用的数据类型，涉及小数的运算基本都是用它来计算的，可用类型转换函数 **float()**将其它类型数据转换为浮点数。由于浮点数是用二进制规则存储数据的，因此在计算中有可能出现误差。

- **长实数**

长实数可以无误差地存储任何实数，但是使用长实数计算时，需要消耗更多的内存，且计算的效率较低。可用类型转换函数 **decimal()**将其它类型数据转换为长实数。

- **54、43.31、-4.45E13、3%**

- **实数**

实数包括整数、长整数、浮点数、长实数这四种类型，可用类型转换函数 **number()**将其它类型数据转换为实数。

- **布尔型**

包括 true/false

- **true、false**

- **字符串**

在使用表达式时，用双引号括起来，其中转义字符用\。而在直接定义字符串常量时，不用引号，还可以用函数 **string()**将其它类型的数据转换为字符串。在表达式中计算两个字符串 *x* 与 *y* 合并，可以在之间添加空格 ***x y***。

- **abcd**
- **="中国\t 美国"**
- **="中国" "北京"**　　将两个字符串合并，结果为：中国北京

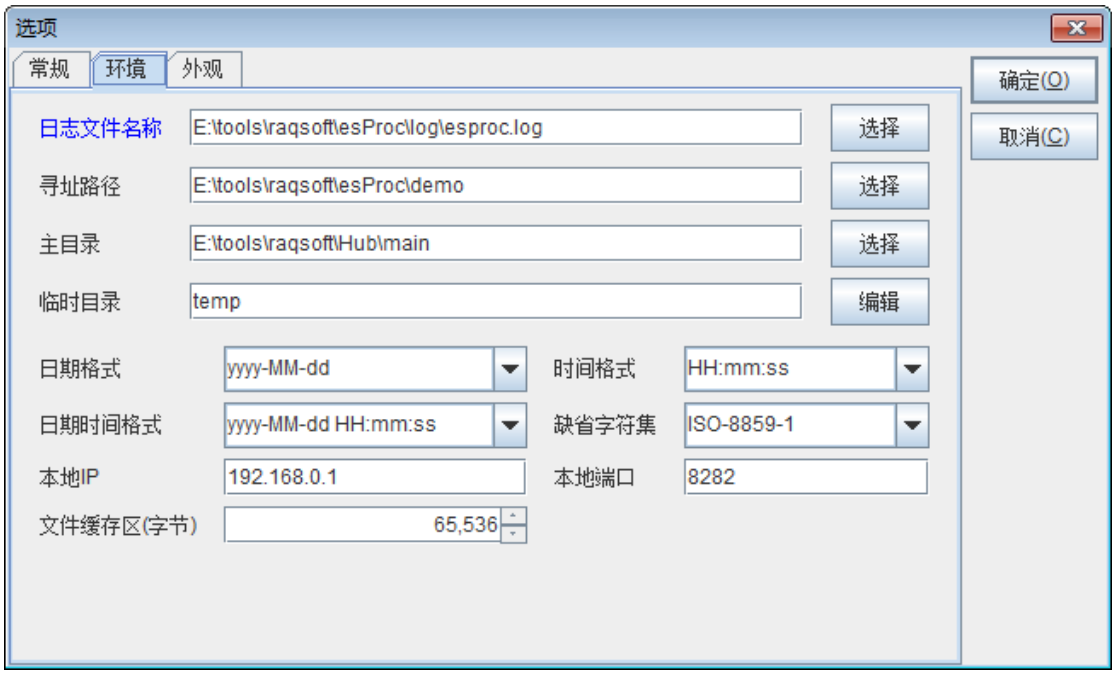
转义规则与 JAVA 相同，可参考使用手册。

- **日期/时间**

写成 **yyyy-mm-dd** 和 **hh:mm:ss** 形式。可以用类型转换函数 **date()**，**time()**和 **datetime()**将字符串或者长整数转换为日期，时间或者日期时间类型。

- **2010-1-3，23:04:23**

可以在菜单栏的**工具**选项中，点击**选项**按钮，在**环境**选项卡中，设定时间/日期数据的格式，以及字符编码等。



当把数据转换为其它类型时，有可能会丢失精度。

2.1.2 数据类型的判断

可以用下面几个函数判断数据的类型：

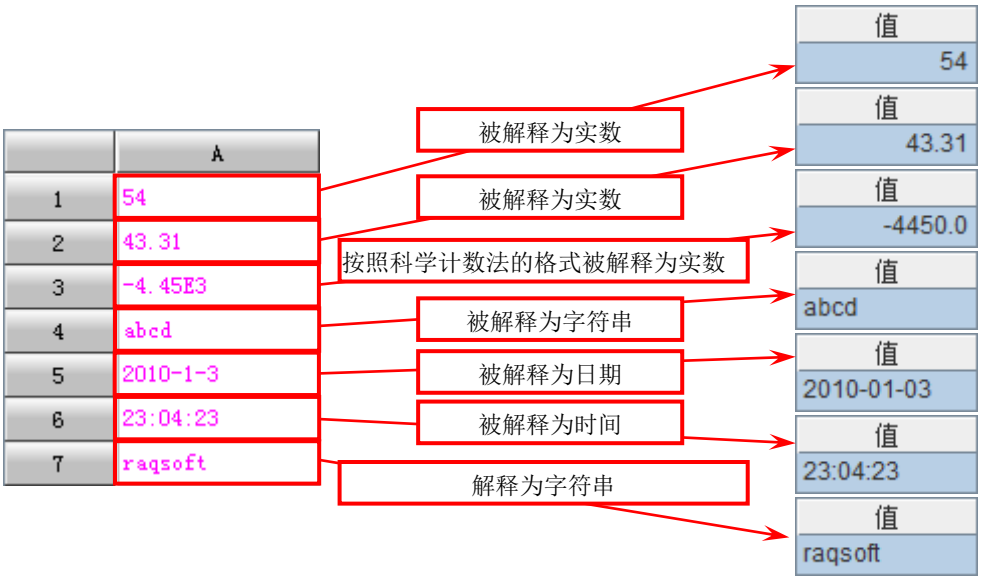
- **ifnumber(*x*)**
  判断 ***x*** 是否是实数
- **ifstring(*x*)**
  判断 *x* 是否是字符串
- **ifdate(*x*)**
  判断 *x* 是否是日期类型或日期时间类型
- **iftime(*x*)**
  判断 *x* 是否是时间类型

- **=ifnumber(3.5)**　　true
- **=ifstring(now())**　　false，**now()**计算当前日期时间

2.2 单元格类型

2.2.1 常数格

格串能被解释为常数的单元格称为**常数格**，其格值即为该常数。
常数格有如下几种情况：



常数格的文字缺省会显示成粉色。根据格子中的数据，会解析为各种数据类型，如果无法理解，会把单元格中的格串解释为字符串。

- 注意：**3%**的写法不能在表达式中使用，仅在常数格中有效。

在单元格中，还可以使用一些常数保留字，注意**大小写是敏感的**！

- **null**

空值，空格子的格值也是 null

- **true**

真

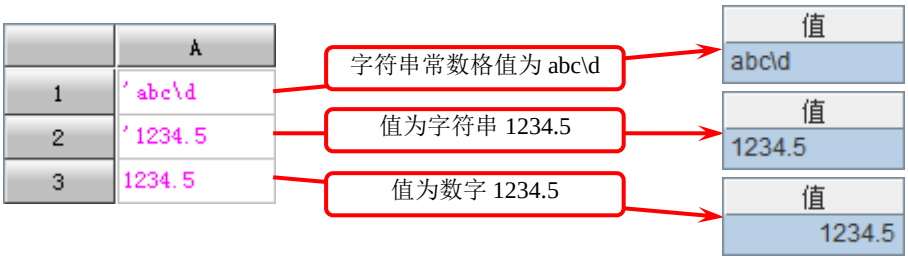
- **false**

假

2.2.2 字符串常数格

格串是以英文单引号'开头的单元格，表示是格值为字符串的常数格，格值即为单引号后字串所构成的字符串。此时，单引号后面的所有字符都会被解析为字符串，不必再添加引号、转义符等字符。

- **'abc\d** 字符串 abc\d。
- **'1234.5** 字符串 1234.5。



2.2.3 计算格

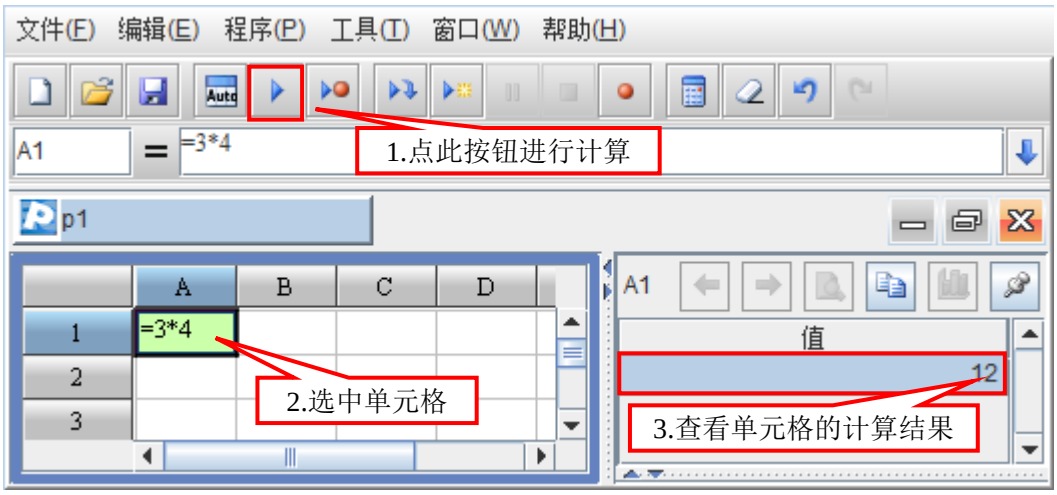
格串是以=开头的计算表达式的单元格称为计算格，其格值即为该表达式的计算结

果。表达式中可以引用其它单元格的格值。

- **=5*(3+2)**
- **=A1*3**
- **=A1/B2+B1**

计算格文本缺省显示成黑色。

点按  执行代码后，会看到计算格底色变成淡黄色，表明该格已经计算出格值。选中有格值的单元格，值显示区将显示其当前格值，即可查看计算结果。



在表达式中，可以使用各种基本操作符计算格值：

➢ ***a+b a-b a*b a/b***

加、减、乘、除运算。

- **=3*4** 12
- **=1+2*3** 7

➢ ***a%b***

求 *a* 除以 *b* 的余数。

- **=121%11** 0，说明 121 是 11 的倍数，可以被 11 整除
- **=100%7** 2

➢ ***a\b***

整数除法，保留 *a* 及 *b* 的整数部分，相除后返回结果的整数部分。

- **=11\4** 2，结果的小数部分舍去
- **=6.1\2.9** 3，相当于 6\2，结果为 3

➢ s_1+s_2

连接字符串 s_1 和 s_2。需要注意的是，如果字符串与实数相加，字符串将被忽略。

- **="Stephen"+" "+"Rolfe"** 字符串 Stephen Rolfe
- **="A"+3** 整数 3，字符串"A"被忽略
- **="A"+string(3)** 字符串 A3，如果确实需要连接字符串和实数，需要先将实数转换为字符串。

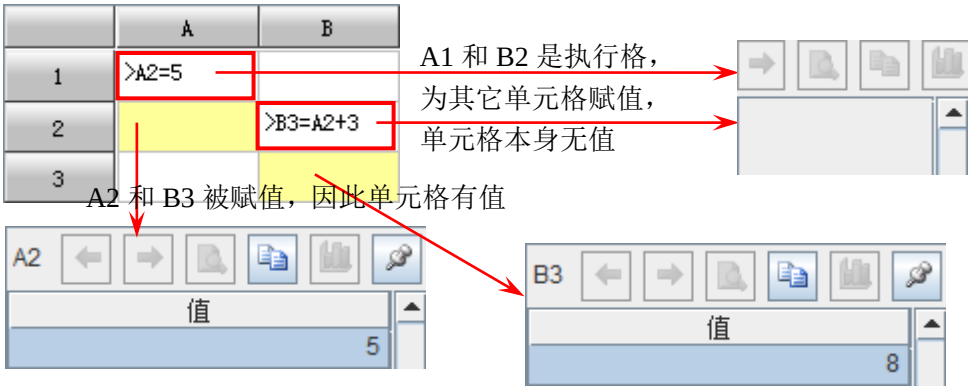
2.2.4 执行格

格串以>开头的单元格称为执行格，用于执行某种操作。执行格没有格值。当执行格以保留字作为开头时，如 if、for 等，>省略不写。

执行格的代码可以修改其它单元格的值。

- **>A1=5**
- **>B1=A1+3**

使用执行格可以将单元格用作变量。

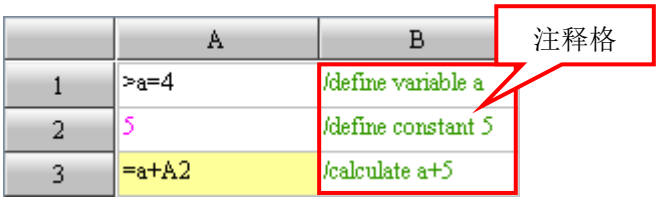


2.2.5 注释格

格串以/开头的单元格称为注释格，其格值为空。

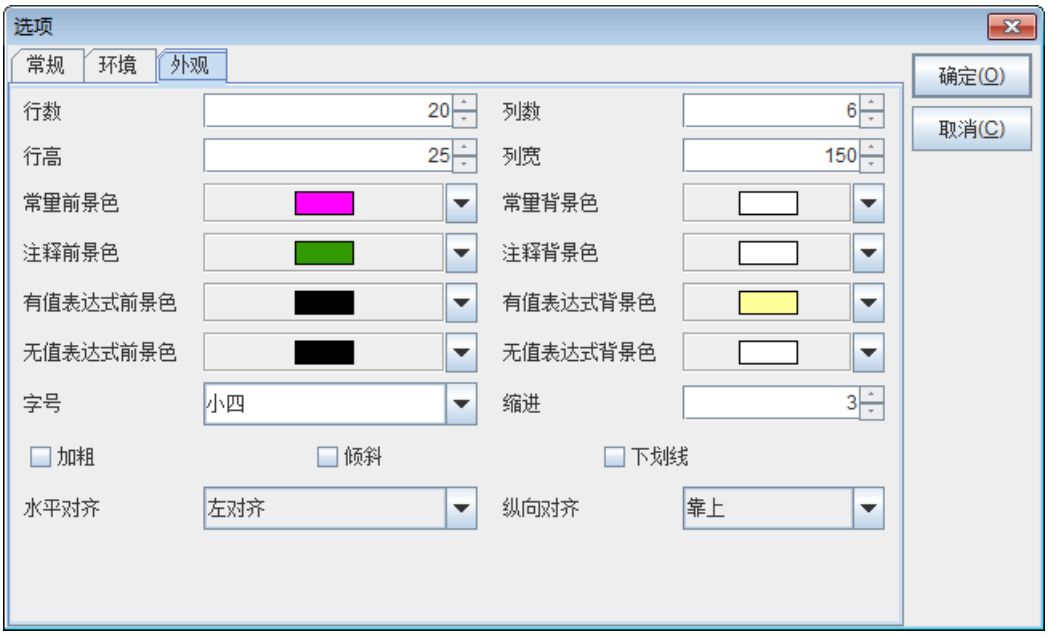
注释格在执行中将被忽略。

注释格文本缺省显示成绿色。



2.2.6 各种类型网格的颜色设定

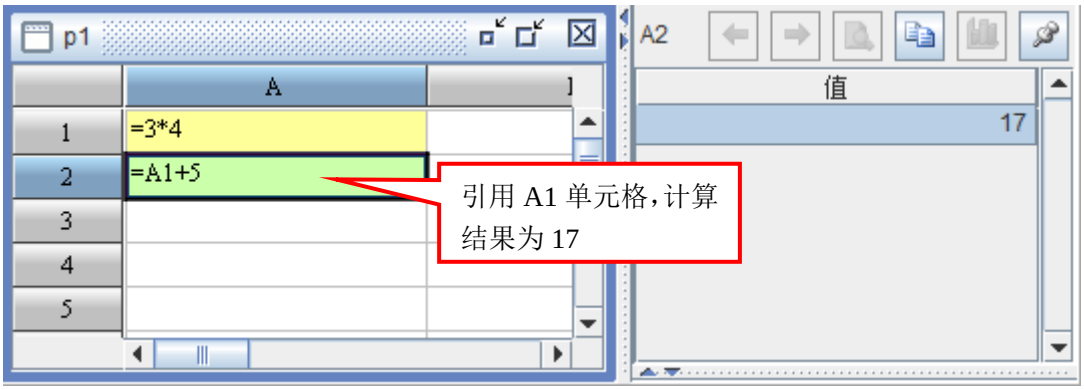
集算器提供选项由用户设置不同类型单元格的前景色和背景色，以及文字的字体。可以在菜单栏的**工具**选项中，点击**选项**按钮，在**外观**选项卡中，查看或修改设置。



2.3 网格中的变量和参数

2.3.1 使用单元格值作为变量

在 2.2.3 与 2.2.4 中的例子中，我们都看到了使用单元格值作为变量使用的情况。在集算器中，如果一个单元格有值，那么在计算格或者执行格中，就可以直接使用单元格名称引用单元格值。有值的单元格可以是常数格、计算格，也可以是被执行格赋值的其它单元格。单元格名称由列号字母和行号整数组成。



2.3.2 定义网格参数

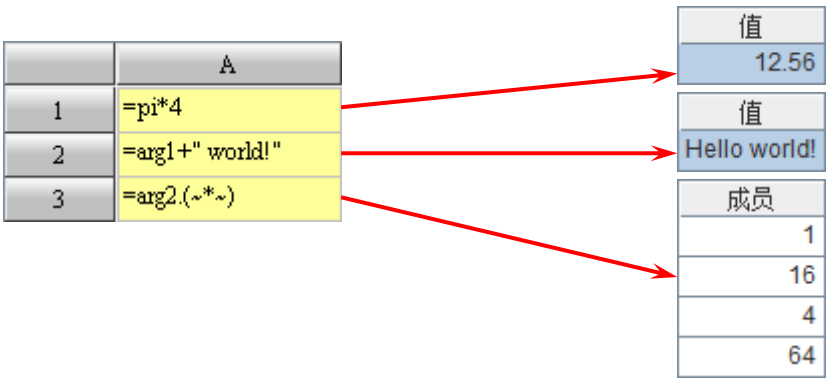
在网格文件中，还可以定义网格参数。

在菜单栏的**程序**菜单项中点击**网格参数**，可以查看网格参数设置：

在网格参数设定窗口中，可以设定网格文件中使用的参数。网格参数的参数名中不能包含空格或符号，且不能只含数字；参数值可以使用各种类型的常量，将根据设定值自动解析为相应的数据类型。需要注意的是，网格参数中无法使用表达式。

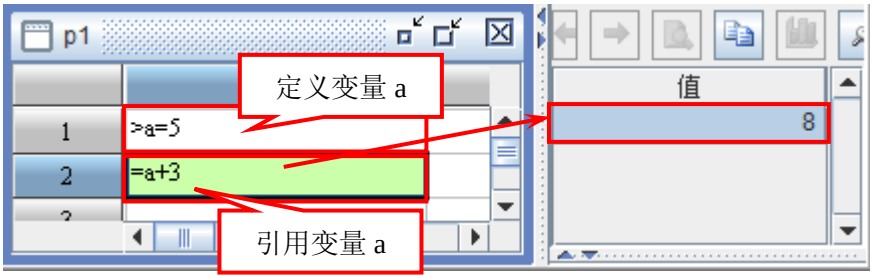
如果勾选**在运行前设定参数**选项，则会在运算网格之前，弹出网格参数设定窗口。

网格参数在使用时，直接用参数名调用：



2.3.3 网格变量

除了直接使用单元格名，还可以使用起了名字的变量，称为**网格变量**。



网格变量无须事先声明，赋值时即自动产生，在整个网格内均有效。但引用尚未赋值的变量将出错。

网格变量也没有数据类型，可以随意修改其值。

➢ ***a*?=*x***

a=*a*?*x* 的简写，其中“?”是运算符。除了用表达式 *a*=*x* 为变量 *a* 赋值以外，还可以使用简化的表达式 *a*?=*x* 改变变量的值。

- **a+=5**　　将变量 *a* 的值增加 5，相当于表达式 **a=a+5**。

➢ ***-a***

a 与*-a* 仅有符号不同，它们互为**相反数**。特别的，如果 *a* 的数据类型为日期时间，计算相反数时，取 *a* 关于 1970 年 1 月 1 日相对称的日期时间。

➢ **(x_1,x_2,…,x_k)**

分别计算表达式 x_1,x_2,…,x_k，并返回 x_k 的计算结果。

- **=(a=1,b=a+4,b*b)**　　结果为 25，同时将网格变量 a 的值设为 1，b 的值设为 5

2.3.4 变量的判断及删除

在集算器中，如果变量已经使用完毕，可以用 **rmv** 函数删除变量。

➢ **ifv(*v*)**

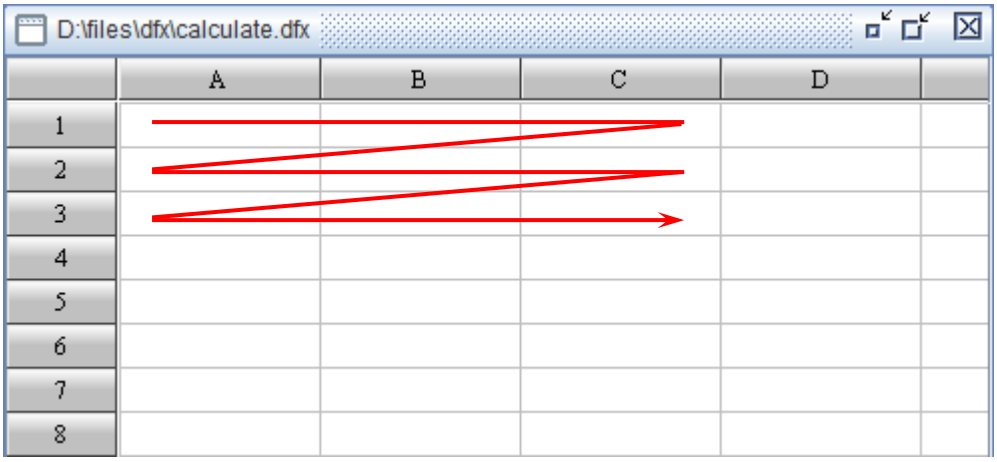
判断 *v* 是否是变量。

➢ **rmv(v_i,…)**

删除网格变量 v_i。删除变量后网格中不能再调用该变量。

2.4 网格的执行次序

网格程序从上到下、从左到右依次执行。



后计算的格子可以引用前面计算的格子，前面计算的格子可以引用后面的常量格子。常数数未被执行时格值也有效，但计算格必须执行后才有格值。

举例：

	A	B	C
1	=A3	=2+1	=B3
2	=B1		
3	3	=3	

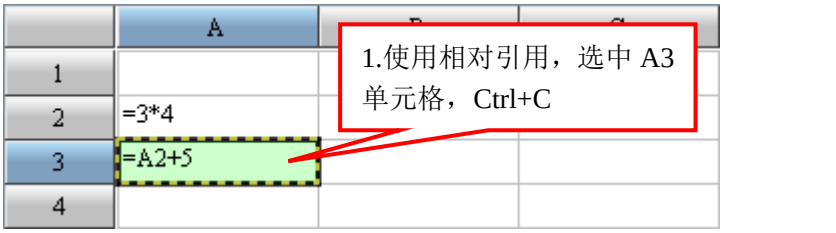
A2 在 **B1** 后计算，因此可以引用 **B1** 单元格；

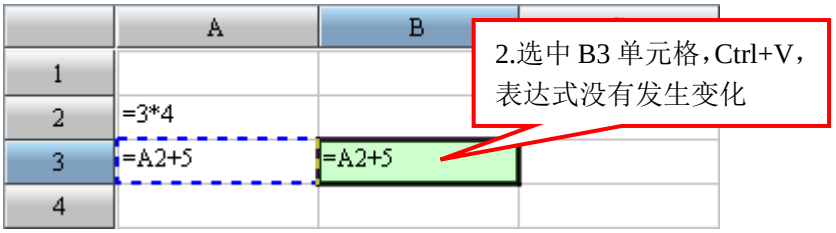
A3 是一个常量格子，**B3** 是表达式，格值为表达式计算的结果，因此 **A1** 单元格能引用 **A3**，但 **C1** 不能正确获得 **B3 的格值**；

A1,**A2**,**B1**,**B3** 计算结果都是 **3**，**C1** 格值为 null。

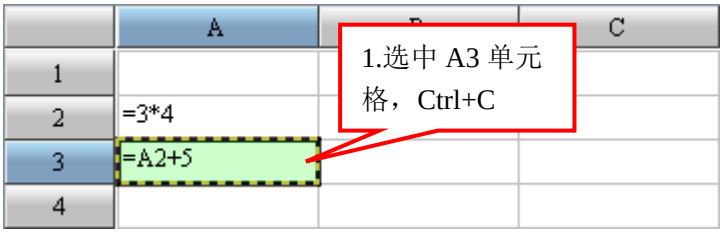
2.5 复制单元格

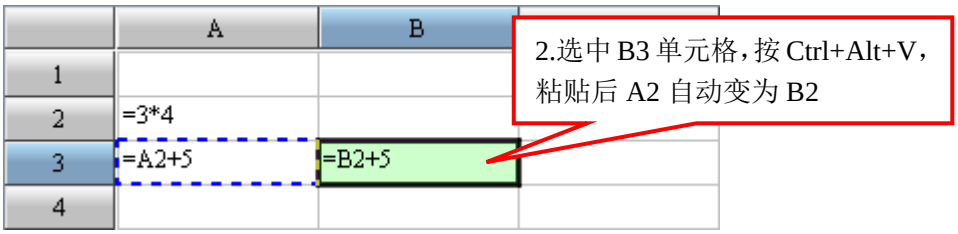
与 EXCEL 不同，在集算器中，直接使用 Ctrl+C 和 Ctrl+V，单元格复制粘贴时不会自动变迁表达式。



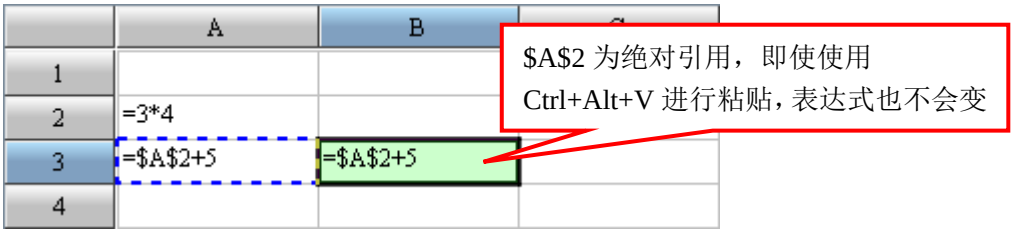


但使用 Ctrl-Alt-V 粘贴时将会像 EXCEL 一样变迁表达式。





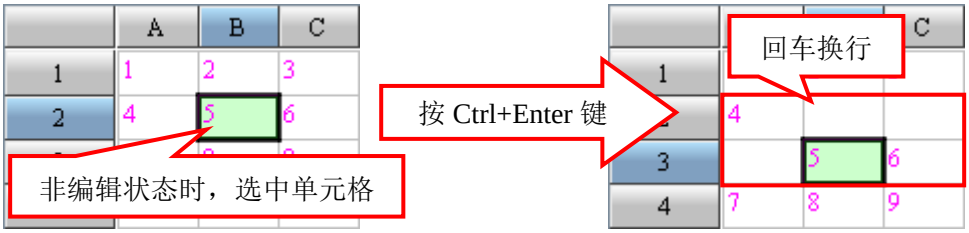
与 EXCEL 相同，行/列号冠以$的单元格在任何时候都不会变迁。



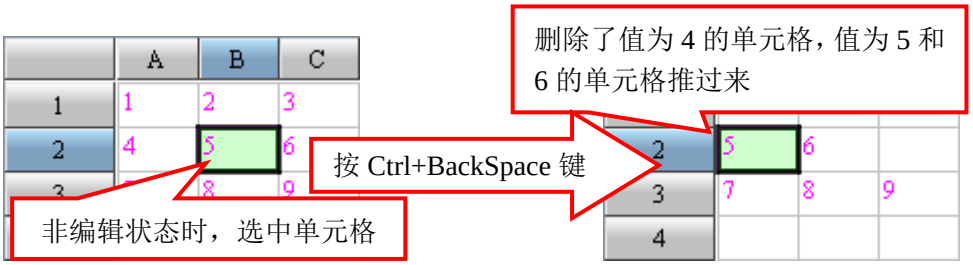
2.6 类似文本编辑的插入和删除

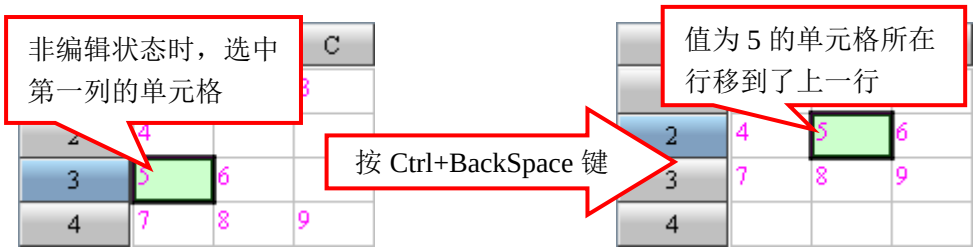
在单元格上按 Ctrl-Enter、Ctrl-BackSpace、Ctrl-Delete 会产生类似回车换行、退格和删除本格的效果，与文本编辑中按 Enter、Backspace、Delete 基本相当。按 Ctrl-Insert 将在本行中插入单元格。

1) Ctrl-Enter 类似回车换行

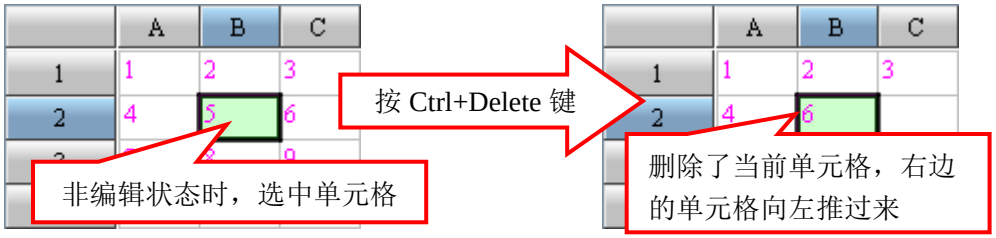


2) Ctrl-BackSpace 向前删除

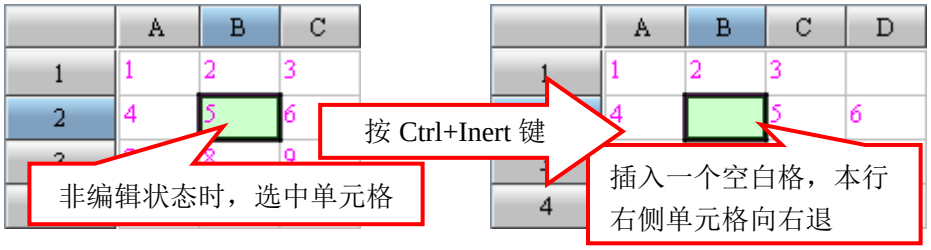




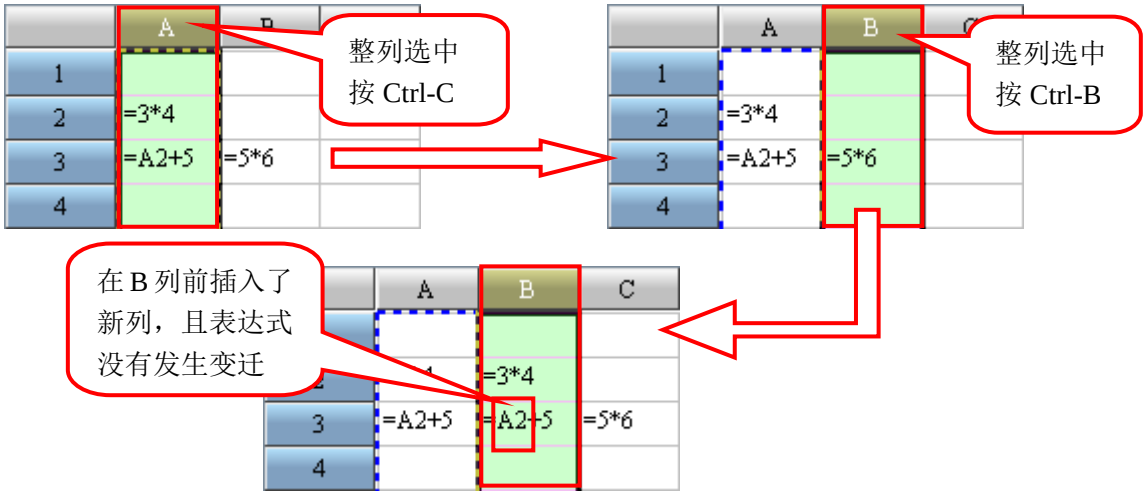
3) Ctrl-Delete 删除本格，后面补回



4) Ctrl-Insert 插入本格，后面右退



5) 当复制区域是整行/列时，使用 Ctrl-B 将会自动先插入行/列后再粘贴内容



如果使用 Cltr-Alt-B 则将同时变迁表达式（与 Ctrl-Alt-C 类似）

2.7 函数与函数选项

2.7.1 常规函数

在表达式里，会使用到各种函数。函数的基本格式是 *f*(…)，其中…是函数的参数。

- **if(*a*,*b*,*c*)**

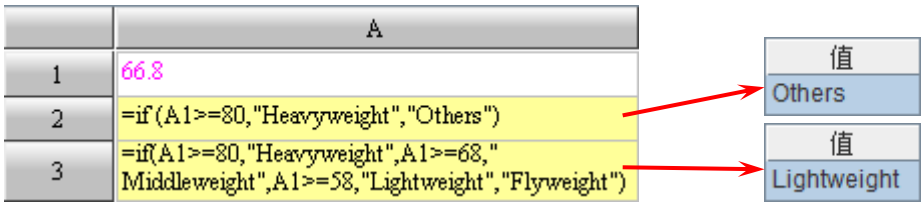
计算表达式 *a*，如果为 true 则计算 *b* 并返回结果，否则计算 *c* 并返回结果。如果 *b* 和 *c* 都省略，则直接返回 true 或 false。

- **=if(*a*>0,*a*,-*a*)**　　计算 *a* 的绝对值。

➢ **if(x_1:y_1,x_2:y_2,…,x_k:y_k;*y*)**

计算表达式 x_1，如果为 true 则计算 y_1 并返回结果；否则继续计算 x_2，如果为 true 则计算 y_2 并返回结果；……；如果所有 x_i 均为 false，则计算 *y* 返回结果。相当于 **if(x_1,y_1,if(x_2,y_2,…,if(x_k,y_k,*y*)))**，在使用时，也可以简单写为 **if(x_1,y_1,x_2,y_2,…,x_k,y_k,*y*)**。

- **=if(*a*>1000:*a**0.8,*a*>500:*a**0.9;*a*)**　　计算购物金额，500 以上打 9 折，1000 以上打 8 折。



➢ **case(*x*,x_1:y_1,x_2:y_2,…,x_k:y_k;*y*)**

计算表达式 *x*，如果值为 x_1 则计算 y_1 并返回结果；如果值为 x_2 则计算 y_2 并返回结果；……；如果不等于所有 x_i，则计算 *y* 返回结果。相当于 **if(*x*==x_1:y_1, *x*==x_2:y_2,..., *x*==x_k:y_k;*y*)**，在使用时，也可以简单写为 **case(*x*,x_1,y_1,x_2,y_2,…,x_k,y_k,*y*)**。

- **=case(*a*%3,1,"one",2,"two","zero")**　　输出 *a* 除以 3 的余数，如 *a* 为 995 时结果为 two。

在函数的参数中，会使用冒号 **：**逗号 **，**分号 **；**这三种分隔符，在处理时，它们的优先级依次降低。

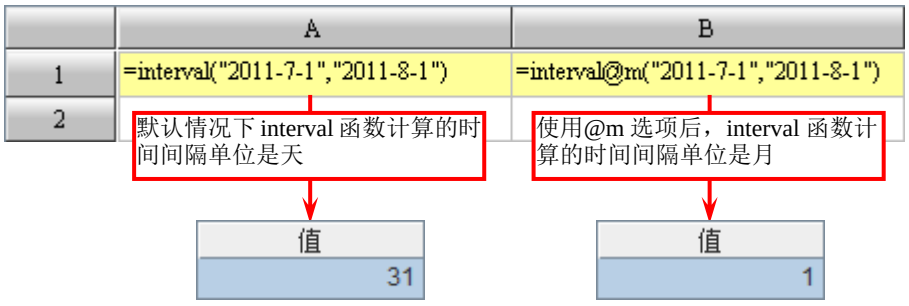
此外，集算器中还提供了大量面向对象的函数，其格式为 *o.f*(…)，*o* 是函数作用的对象，在集算器中，*o* 可以是序列、序表、排列、记录等，在后面会一一介绍。

2.7.2 函数选项

集算器中，很多函数都可以使用函数选项。通过函数选项，同一个函数可以有不同的工作方式。函数选项的基本格式是 *f*@*o*(…)，*o* 就是函数 *f* 的选项。

- **=interval@m("2011-7-1", "2011-8-1")**

这个表达式中，**@m** 就是 **interval** 函数的选项，有了这个选项后，函数计算两个时间的间隔时，将以月为单位。



在需要时，可以同时使用多个允许共用的选项，选项之间没有次序。

➢ **in(*x*,*a*:*b*)**

x 是否在区间[*a*,*b*]中，即是否满足 *a*<=*x*<=*b*，根据结果返回 true 或者 false。当 *a* 省略时，视为无穷小；当 *b* 省略时，视为无穷大。

- **@b** 根据结果返回整数：在数轴上，当 *x* 在区间内时返回 0，在区间左侧返回-1，在区间右侧返回 1。
- **@l** 左侧使用开区间，即判断表达式用 *a*<*x*<=*b*。
- **@r** 右侧使用开区间，即判断表达式用 *a*<=*x*<*b*。

关于各类函数的具体使用方法，可点击**帮助**>**函数参考**，阅读相应文档。

2.8 用文件对象读写数据

2.8.1 集算器中的文件对象

在集算器中，可以使用文件的路径来定义文件对象，并通过对文件对象的操作，来实现对文件的读写等各种操作。

- **file(*fn*: *cs*)**

打开文件名为 *fn* 的文件对象。*fn* 为加载的文件名，使用绝对路径。如果使用相对路径，则相对于当前路径（若无当前路径，则相对于系统当前路径）。*cs* 为字符集(可省略，默认为操作系统缺省值)

- **@s** 按照指定顺序搜索非绝对路径的文件名，如果当前路径存在则先搜索当前路径，再搜索路径列表 (集算器选项菜单里配置的**寻址路径**)，否则先搜索类路径再搜索寻址路径列表，返回结果为只读文件名称 *fn*。

2.8.2 使用文件对象存储/读取字串

使用文件对象，可以读取文件信息或者修改文件，如将集算器中的字串存储到文件，也可以将文件中的内容读取为字串等。

- ***f*.exists()**

文件是否存在。

- ***f*.size()**

文件的字节数。

- ***f*.date()**

文件最后修改的日期时间。

- **movefile(*fn*, *path*)**

将文件 *fn* 移动到指定的路径 *path*。*path* 省略时，删除文件；*path* 中只有文件名而没有路径时，修改文件 *fn* 的名称。

- **@c** 复制文件，而不是移动。
- **@y** 当目标路径下存在同名文件时，强行覆盖；无**@y** 则取消移动操作。

- ***f*.write(*s*)**

将字串 *s* 写出到文件对象 *f* 中。s 也可以是 Blob 数据。

- **@a** 追加写入。
- **@b** 写为二进制文件，使用二进制文件时效率更高，但无法作为文本编辑。

	A
1	=file("D:/files/txt/test.txt")
2	test
3	>A1.write(A2)

字串写入后的文件如下：

test

使用@a 选项，可以追加写入文件。

	A
1	=file("D:/files/txt/test.txt")
2	test2
3	>A1.write@a(A2)

@a 选项，字串被追加写入文件：

test
test2

➢ ***f*.read()**

将文件对象 *f* 读为字串。

- **@b** 读为二进制数据。
- **@n** 返回成串序列，每行作为一个成员。
- **@v** 根据相应的数据类型返回串序列，可结合**@n** 选项使用

	A
1	=file("D:/files/txt/test.txt")
2	=A1.read()

值
test test2

3. 序列

3.1 什么是序列

一些数据构成的有序集合即称为序列，构成序列的数据称为其成员。序列相当于高级语言中的数组，但其成员的数据类型不要求一致。

序号	1	2	3	4	5	…
成员	1 整数	true 布尔型	“abc” 字符型	1.1234 浮点数	2010-4-12 日期型	…

序列中的成员是**有序的**，根据序号，就可以找到某个成员。注意，序列的**序号是从 1 开始的**。

3.2 创建序列

3.2.1 常数序列

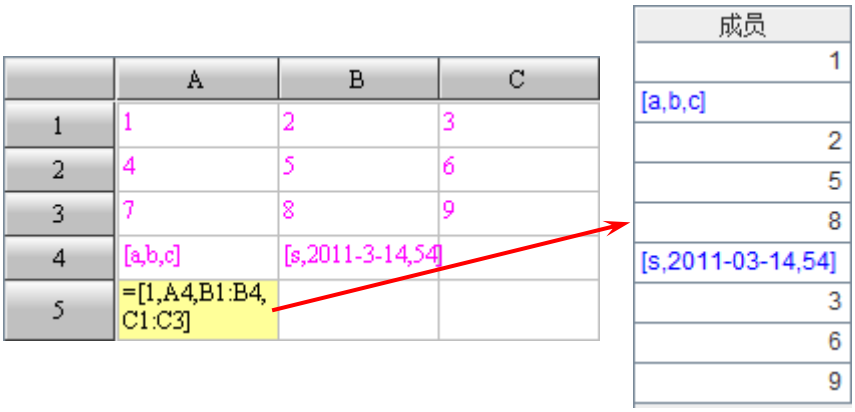
将成员用**[]**括起来即可表示常数序列。

- **[1,3,4]**
- **[a,b,c]**
- **[s,2011-3-14,54]**

3.2.2 用表达式定义序列

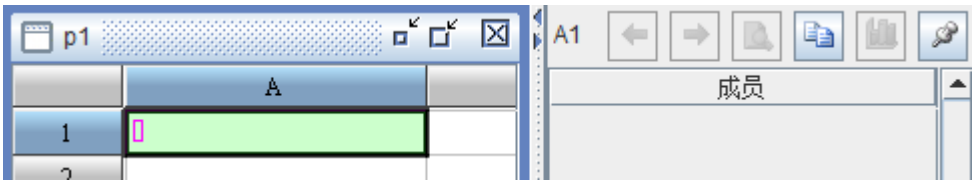
用表达式直接定义序列，格式与常数序列类似，表达式中可以引用单元格，并可用冒号表示一片单元格区域，取区域时单元格值先行后列排列。

- **=[1,A4,B1:B4,C1:C3]**



3.2.3 空列

没有成员的序列称作**空列**，可以直接用**[]**定义。



3.3 序列与成员

3.3.1 序列与成员

- **ifa(*x*)**

  x 是否是序列。

- ***A*.len()**

  序列 *A* 中成员的个数，称为 *A* 的**序列长度**。

- **空列、*n* 序列**

  长度为 0 的序列，就是空列；长度为 *n* 的序列，称为 ***n* 序列**。

- **单列、纯序列**

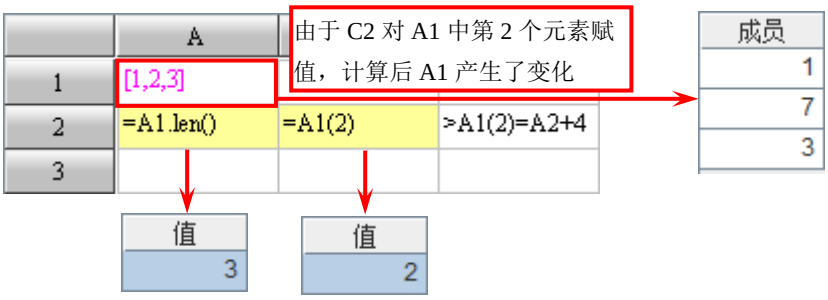
  没有重复成员的序列，称为单列；所有成员的数据类型相同的序列称为**纯序列**。

- ***A*(*i*)**

  取出序列 *A* 中的第 *i* 个成员。注意，在集算器中，获取序列成员用 *A*(*i*)而不是 *A*[*i*]。

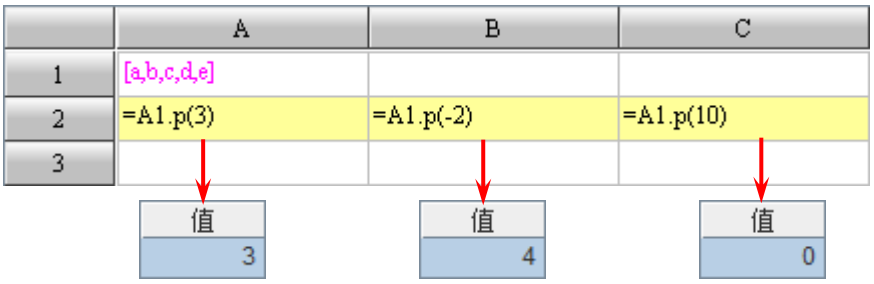
- ***A*(*i*)=*x***

对序列 *A* 中的第 *i* 个成员赋值，*i* 越界时报错。



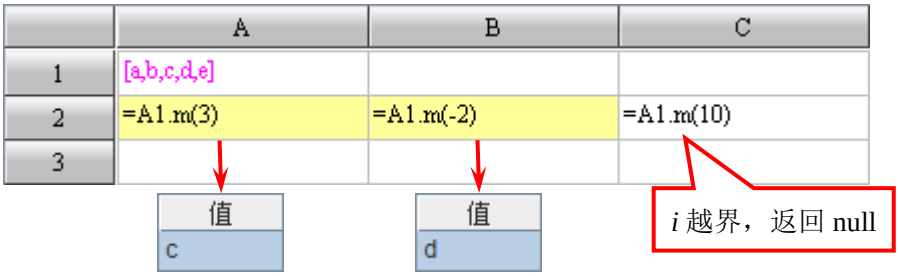
- ***A*.p(*i*)**

i>0 时，返回 *A* 中的第 *i* 个成员的序号；*i*<0 时，返回倒数第 *i* 个成员的序号。如果越界，返回的序号为 0。



- ***A*.m(*i*)**

i>0 时，返回 *A* 中的第 *i* 个成员；*i*<0 时，返回倒数第 *i* 个成员。如果越界，返回 null。



3.3.2 修改序列

- ***A*.insert(*k*,*x*)**

在序列 *A* 的第 *k* 个成员之前，插入新成员，当 *x* 是序列时插入 *x* 序列的所有成员作为新成员；当 *x* 不是序列时则插入一个成员 *x*。

- ***A*.delete(*k*)**

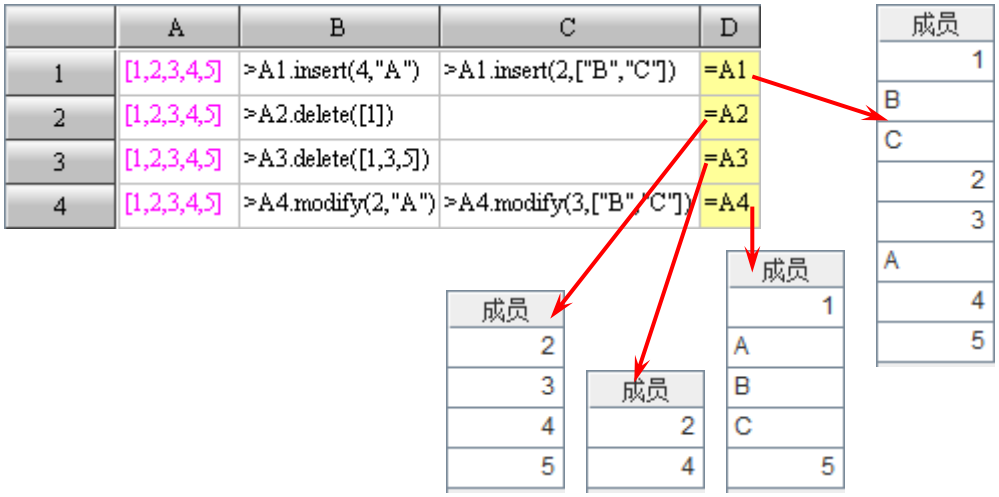
k 为整数，删除序列 *A* 的第 *k* 个成员。

- ***A*.delete(*p*)**

在序列 *A* 中删除序号数列 *p* 中的所有成员。

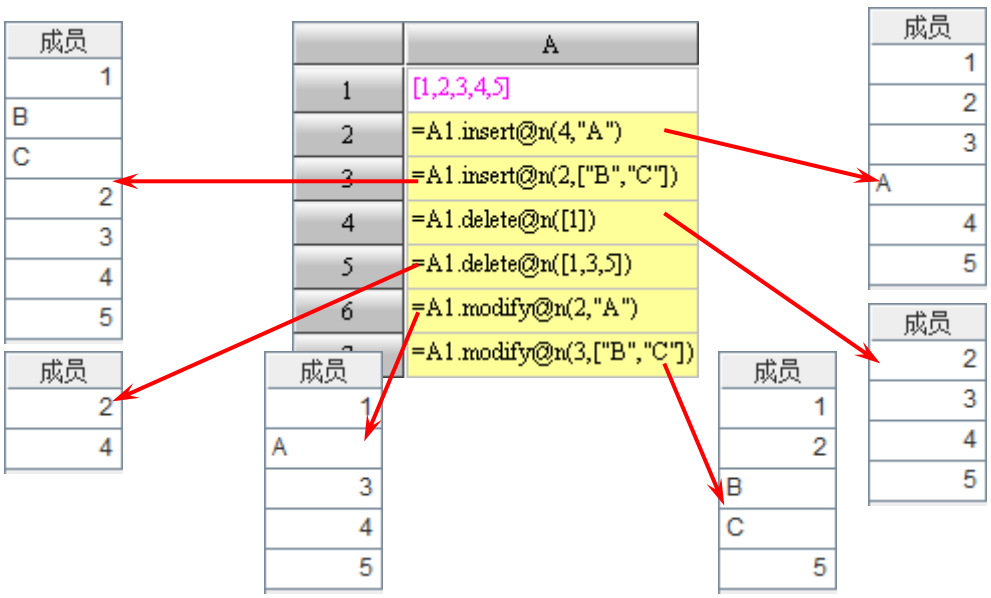
- ***A*.modify(*k*,*x*)**

从序列 *A* 的第 *k* 个成员起，修改 *A* 中成员的值，当 *x* 是序列时依次修改为 *x* 序列的所有成员；当 *x* 不是序列时仅修改第 *k* 个成员为 *x*。



在修改序列的各个函数中，均可以使用**@n** 选项：

❖ **@n**　执行时，不修改序列 *A*，而返回新序列。



3.4 数列

成员都是整数的序列，称为**数列**。

3.4.1 数列相关的概念

➢ ***n* 置换**

由 1,2,…,*n* 这 *n* 个数所构成，顺序不定的数列，称为 *n* 置换。

- **[1,4,3,2,5]**、**[1,2,3,4,5]**和**[5,4,3,2,1]**，这些数列都是 5 置换。

3.4.2 to 函数

➢ **to(*a*,*b*)**

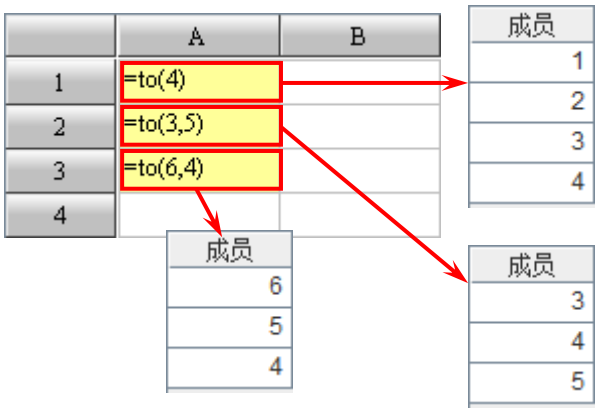
产生从整数 *a* 到整数 *b* 的所有整数构成的数列，包括 *a* 和 *b* 本身，*a*>*b* 或者 *a*<*b* 均可。

❖ **@s**　　*b* 改为指定结果中整数个数，*b*>0 时数列递增 1，*b*<0 时数列递减 1。

- **=to@s(10,3)**，结果为数列[10,11,12]
- **=to@s(10,-3)**，结果为数列[10,9,8]

➢ **to(*n*)**

to(1,*n*)的简写形式，*n* 小于 1 时返回空列。

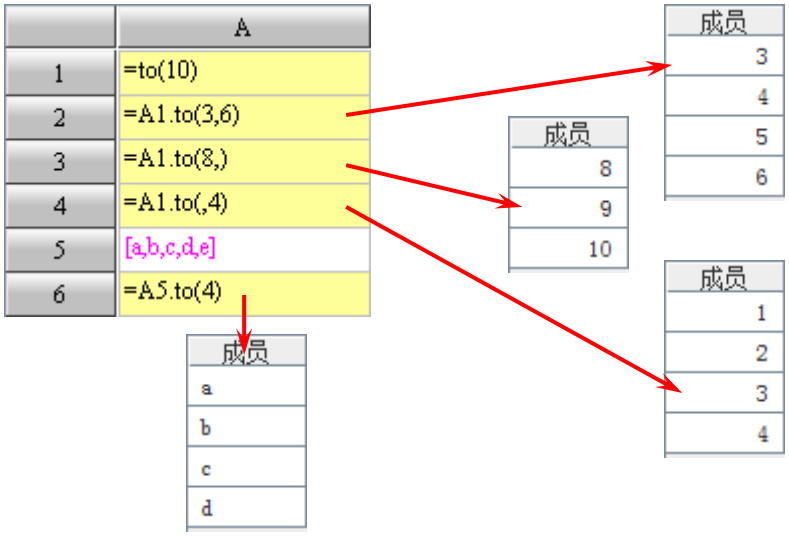


➢ ***A*.to(*a*,*b*)**

序列 *A* 中，从第 *a* 个起到第 *b* 个成员所构成的序列，即 ***A*(to(*a*, *b*))**的简写形式。当 *a* 省略时，默认为 1；当 *b* 省略时，默认为 ***A*.len()**，注意此时逗号不能省略。

➢ ***A*.to(*a*)**

序列 *A* 中，从 1 个起到第 *a* 个成员所构成的序列，相当于 ***A*.to(1, *a*)**。



3.5 产生列

由序列 *A* 的成员构成的序列称为 *A* 的**产生列**。

3.5.1 使用函数生成产生列

➢ ***A*(*p*)**

[*A*(*p*(1)),*A*(*p*(2)),…]，用位置数列 *p* 获得 *A* 的产生列。

➢ ***A*.p(*p*)**

[*A*.p(*p*(1)),*A*.p (*p*(2)),…]，获得位置数列 *p* 在 *A* 中对应的实际位置，*p* 中可以有负数。

- ***A*.m(*p*)**

A(*A*.p(*p*))，用位置数列 *p* 获得 *A* 的产生列，*p* 中可以有负数。

- ***A*.dup()**

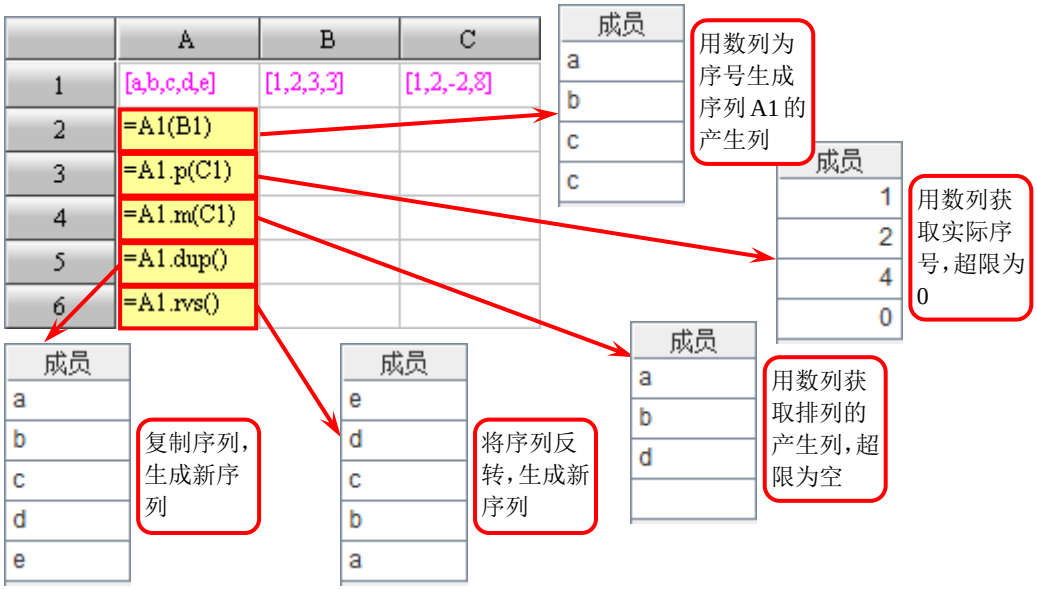
A(to(*A*.len()))，即复制序列 *A*。

- ***A*.rvs()**

A(to(*A*.len(),1))，即反转序列 *A*。

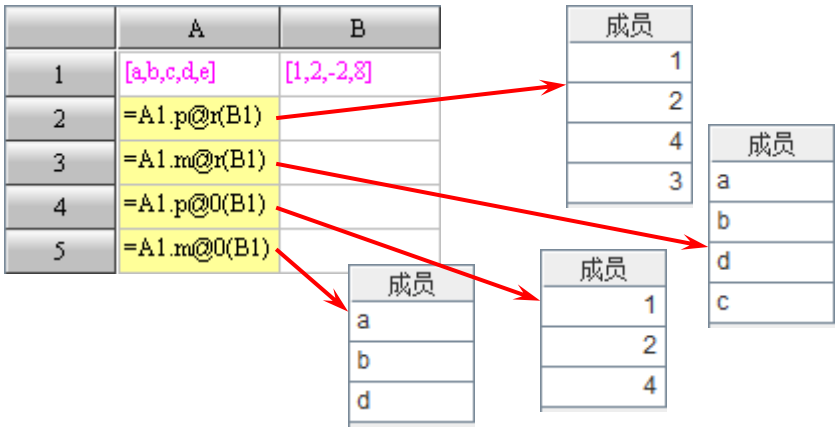
- ***A*.step(*m*,*k*)**

A([*k*,*k*+*m*,*k*+2**m*,…])，即返回 *A* 的产生列，其位置数列从 *k* 开始，步进值为 *m*。



3.5.2 在 *A*.p(*p*)与 *A*.m(*p*)中使用选项

在 ***A*.p(*p*)**与 ***A*.m(*p*)**中，可以使用选项**@r** 和**@0**。在使用**@r** 选项时，对于数列中越界的序号，在获取成员或者实际序号时，将进行回转操作，循环读取。在使用**@0** 选项时，数列中越界的序号将被忽略，0 或者空值不会出现在结果序列中。



3.5.3 产生列相关函数

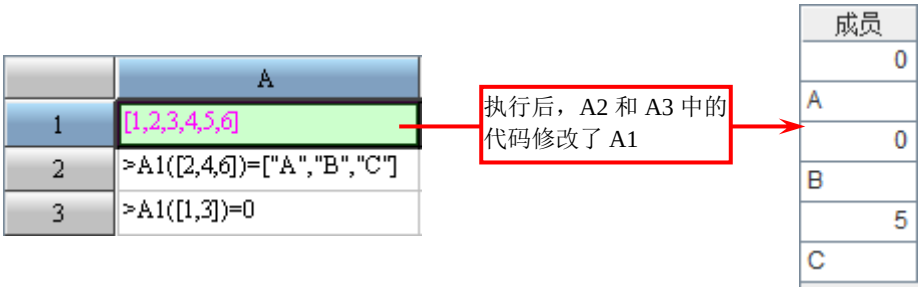
- ***A*(*p*)=*X***

将 *A* 的产生列 *A*(*p*)依次用序列 *X* 的成员赋值，即 *A*(*p*(1))=*X*(1), *A*(*p*(2))=*X*(2),…*X* 的序列

长度要大于或等于 *p* 的序列长度。

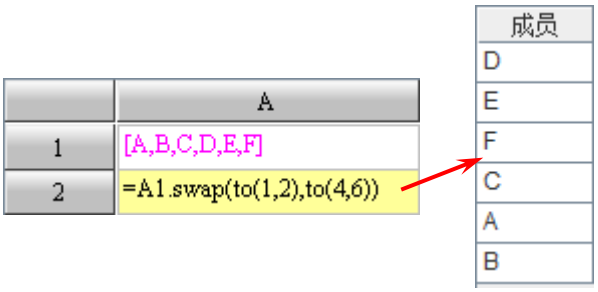
- ***A*(*p*)=*x***

将 *A* 的产生列 *A*(*p*)中的成员全部赋值为单值 *x*，即 *A*(*p*(1))=*x*, *A*(*p*(2))=*x*,…。



- ***A*.swap(*p*,*q*)**

p 与 *q* 均为连续的数列区间，且无交集，将 *A* 中序号在两个区间内的成员交换位置，构成新序列返回。



swap 函数并不改变原序列。

3.6 序列与字串

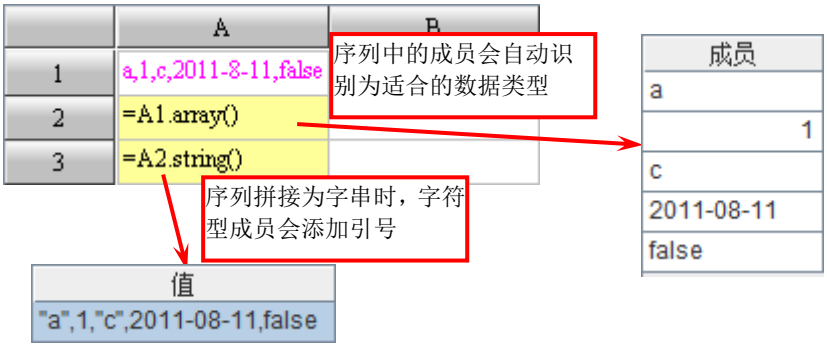
通过 ***s*.array()**和 ***A*.string()**两个函数，序列和字串可以很方便地相互转化。

- ***s*.array(*d*)**

将字串 *s* 以分隔符 *d* 拆分成序列，自动识别数据类型；*d* 缺省为逗号。

- ***A*.string(*d*)**

将序列 *A* 用分隔符 *d* 连接，拼接成字串，自动处理数据类型；*d* 缺省为逗号。



- ***s*.regex(*rs*)**

用正则表达式 *rs* 匹配字串 *s*，返回匹配结果的序列；如果无法匹配，则返回 null。

❖ **@i**　大小写字母不敏感。

❖ **@u** 使用 unicode 匹配。

用 **regex** 函数，最简单的用法就是判定某个字符串与指定正则表达式的匹配结果，下面先来看一下与数字字符串的匹配用法：

	A	B	C
1	'4	'12	'546201.05
2	=A1.regex("([0-9])")	=A1.regex("(\\d)")	=A1.regex("([0-3])")
3	=B1.regex("([0-9][0-9])")	=B1.regex("([0-9]+)")	=B1.regex("([0-9]*)")
4	=C1.regex("([0-9]*)")	=C1.regex("(\\d*)")	=C1.regex("(\\d*.\\d*)")

由于 *s*.regex(*rs*)作用于字符串 *s*，因此，第 1 行的数前面都用一个单引号 ' 表示是字符串常数。正则表达式中，可以用一些符号来与数字匹配，如 A2 中，[0-9]表示 1 个数字，可以匹配 A1 中的字符串，在外面加小括号()后可以返回匹配结果：

Member
4

可以看到 **regex** 函数返回的是一个匹配结果的序列。B2 中，\d 同样用来表示 1 个数字，由于在字符串中\是转义符，因此需要写为\\d，匹配结果和 A2 相同。C2 中，[0-3]表示 1 个 0 至 3 之间的数字，无法匹配"4"，因此结果为 null。

正则表达式中，还可以用+或者*表示多个字符。如 A3 中[0-9][0-9]表示 2 个任意数字，B3 中[0-9]+表示 1 个或更多的任意数字，C3 中[0-9]*表示 0 个或更多的任意数字。A3、B3 和 C3 都可以和 B1 中的字符串匹配，返回结果如下：

Member
12

A4 与 B4 中的正则表达式，均表示 0 个或多个任意数字，而 C1 的字符串中间包含小数点，因此匹配结果只取出了小数点之前的数字，A4 与 B4 中结果如下：

Member
546201

C4 的正则表达式中，\\d*.\\d*，表示匹配多个数字，后面跟 1 个小数点，后面再跟着多个数字的模式，匹配结果如下：

Member
546201.05

用 **regex** 函数，还可以与普通的字符串做匹配，如：

	A	B	C
1	x	Max	Adam Stuart
2	=A1.regex("([a-z])")	=A1.regex("([^0-9])")	=A1.regex@u("(\\u0078)")
3	=B1.regex("([a-z]{2})")	=B1.regex("([a-z]*)")	=B1.regex@i("([a-z]{2})")
4	=C1.regex("([ASadmrtu]*)")	=C1.regex("(\\S* \\S*)")	=C1.regex("([A|S][a-z]* [A|S][a-z]*)")

A2 中，[a-z]表示匹配一个 a 到 z 之间的字符；B2 中[^0-9]表示匹配 1 个不在 0 到 9 之间的字符，即 1 个非数字字符；C2 中，用 unicode 值\u0078 即 x 匹配。A2,B2 和 C2 中的匹配结果如下：

Member
x

A3 中，{2}表示匹配 2 个字符，即试图匹配 2 个 a 到 z 之间的字符，由于 M 是大写字母不在此范围中，因此匹配结果为 null。

B3 中，用*表示 0 到多个 a 到 z 之间的字符，即使第一个字符 M 就不满足条件，但是用*时是允许 0 个匹配字符的，因此还是能获得匹配结果空字符，而并非 null：

Member

C3 中，加了@i 选项，不分大小写匹配 2 个字母，匹配结果如下：

Member
Ma

A4 中，匹配多个由 ASadmrtu 这些字母及空格组成的串。B4 中，\S 表示 1 个非空白字符，即非空格制表符换行符换页符回车符等，整个正则表达式表示多个非空字符，后面跟一个空格，后面再跟多个非空字符的模式。C4 中，[A|S]表示 A 或 S，整个正则表达式表示 A 或 S 开头的单词，后面跟一个空格，后面再跟 A 或 S 开头的单词的模式。A4,B4 和 C4 的匹配结果是相同的，结果如下：

Member
Adam Stuart

上面我们已经对正则表达式的匹配用法有了初步的了解，实际上，**regex** 函数在集算器中更多地是用来把字符串按指定规则拆分为序列的，如：

	A	B	C
1	Max	Adam Stuart	Adam,Bob,Charlotte,Doug
2	=A1.regex@i("([a-z])([a-z])([a-z])")		
3	=B1.regex("(\S*) (\S*)")		
4	=C1.regex@i("([a-z]*),([a-z]*),([a-z]*),([a-z]*)")		

A2 将 A1 中的字符串拆分为字母组成的序列，拆分时忽略大小写，结果如下：

Member
M
a
x

A3 将 B1 中的字符串拆分，以空格分隔，拆分为 2 个由连续的非空白字符组成的部分，结果如下：

Member
Adam
Stuart

A4 将 C4 中的字符串以逗号分隔，拆分为 4 个单词，忽略大小写，结果如下：

Member
Adam
Bob
Charlotte
Doug

可以看到，在使用 **regex** 函数将字符串拆分为序列时，需要用多个小括号()中的表达式依次匹配结果序列中的各个成员。其中 A3 和 A4 中的表达式比较常用，可以将一个字符串

拆分为多个单词组成的序列。需要注意的是，一旦其中某个成员无法匹配，函数返回的结果即为 null。

3.7 序列的基本运算

3.7.1 序列的双目运算

- ***A|B***

和列，简单将两个序列连接起来，其中 *B* 的成员添加在 *A* 成员的后面。当 *A*、*B* 其中之一或者两者都是单值，而不是序列时，作为单值的序列来处理。

- ***A&B***

并列，获得新序列，在 *A* 的成员之后再加入 *B* 中成员，加入时去除 *B* 中已在 *A* 中出现的成员。当 *A*、*B* 其中之一或者两者都是单值，而不是序列时，作为单值的序列来处理。

- ***A^B***

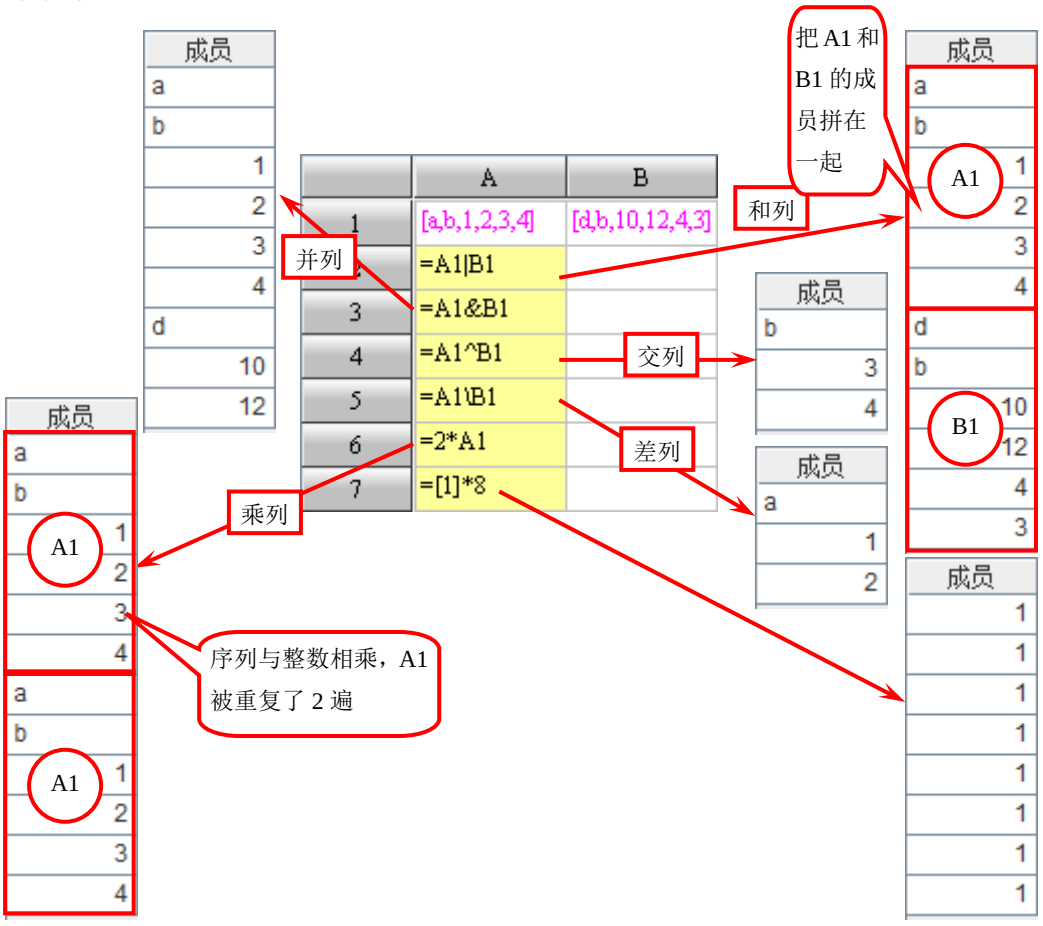
交列，*A* 与 *B* 的交集，从 *A* 中依次获得同时出现在 *B* 中的成员所组成的序列。

- ***A\B***

差列，在 *A* 中但未在 *B* 中出现的成员，当 *B* 不是序列时，作为单值序列处理。

- ***k*A***

A|*A*|…|*A*，*k* 个 *A* 做和列，即将 *A* 复制 *k* 遍，表达式中 *k* 和 *A* 的位置可以互换。



3.7.2 序列的对位运算

两个长度相同的，数值构成的序列可按成员进行对位计算，返回序列。

- ***A++B***

  [*A*(1)+*B*(1),*A*(2)+*B*(2),…]

- ***A--B***

  [*A*(1)-*B*(1),*A*(2)-*B*(2),…]

- ***A**B***

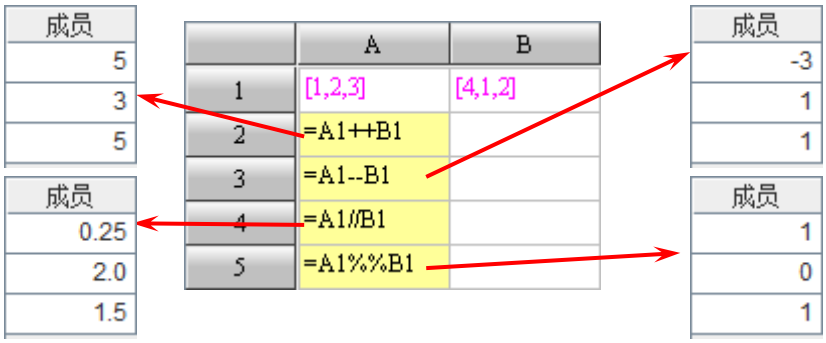
  [*A*(1)**B*(1),*A*(2)**B*(2),…]

- ***A//B***

  [*A*(1)/*B*(1),*A*(2)/*B*(2),…]

- ***A%%B***

  [*A*(1)%*B*(1),*A*(2)%*B*(2),…]，这里%是求余运算。

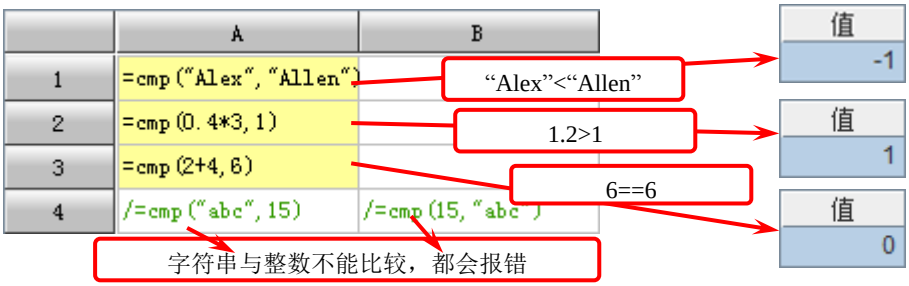


3.7.3 序列的比较

用函数 cmp(*x*,*y*)可以比较两个表达式 *x* 和 *y* 计算结果的大小，用函数 cmp(*A*,*B*)可以比较两个序列 *A* 与 *B* 的大小。

- **cmp(*x*,*y*)**

比较表达式 *x* 和 *y* 计算值的大小，*x*>*y* 时返回 1，*x*<*y* 时返回-1，*x*=*y* 则返回 0。如果 *x* 与 *y* 无法比较，则报错。



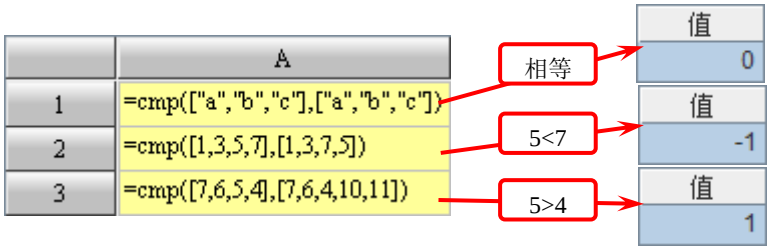
在集算器中，用下面的方式表示 *a* 与 *b* 之间的大小关系：

- ***a>b*** *a* 大于 *b*，即 cmp(*a*,*b*)>0
- ***a<b*** *a* 小于 *b*，即 cmp(*a*,*b*)<0
- ***a==b*** *a* 等于 *b*，即 cmp(*a*,*b*)==0，为了与赋值区分使用两个等号
- ***a!=b*** *a* 不等于 *b*，即 cmp(*a*,*b*)!=0
- ***a>=b*** *a* 大于或等于 *b*，即 cmp(*a*,*b*)>=0
- ***a<=b*** *a* 小于或等于 *b*，即 cmp(*a*,*b*)<=0

如果序列 *A* 中，任何一个成员都不小于前一个成员，即 *A*(*i*)>=*A*(*i*-1)；或者任何一个成员都不大于前一个成员，即 *A*(*i*)<=*A*(*i*-1)，两种情况均称序列 *A* 为**有序列**；其中前一种序列称为**递增列**。

- **cmp(*A*,*B*)**

比较序列大小时，对位比较每个成员的值，遇到第一个不等成员时根据大小分别返回 1 或-1，*A* 与 *B* 全等则返回 0。特别的，**cmp(*A*)**或者 **cmp(*A*,0)**，表示 *A* 和与之等长且成员均为 0 的数列比较，即 **cmp(*A*,[0,0,...,0])**。



为了与普通数值的比较相区分，两个序列的比较不能简写。

3.7.4 序列的聚合运算

在集算器中，对于 *n* 序列 *A*，可以使用函数进行各种聚合运算。

- ***A*.count(*x*)**

计算 *A* 中使得表达式 *x* 的结果中非空且非 false 的成员数目。省略 *x* 时对 *A* 中成员计数，和 *A*.len()不同。

- ***A*.ifn()**

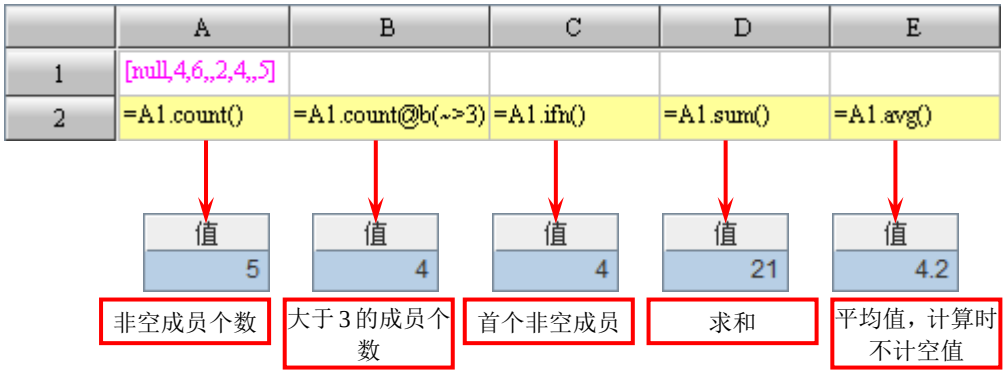
获取 *A* 中的第一个非空成员。

- ***A*.sum(*x*)**

对 *A* 中所有成员计算表达式 *x*，对结果求和。如果 *x* 省略直接对 *A* 的成员求和。

- ***A*.avg(*x*)**

对 *A* 中所有成员计算表达式 *x*，求结果的平均值。如果 *x* 省略，计算 *A* 中成员的平均值，计算时不计空值成员。



- ***A*.min(*x*)**

对 *A* 中所有成员计算表达式 *x*，求结果的最小值。如果 *x* 省略，求 *A* 成员的最小值。

- ***A*.max(*x*)**

对 *A* 中所有成员计算表达式 *x*，求结果的最大值。如果 *x* 省略，求 *A* 成员的最大值。

➢ ***A*.variance()**

求方差。计算方差时，不计空值成员。

➢ ***A*.ranki(*y*)**

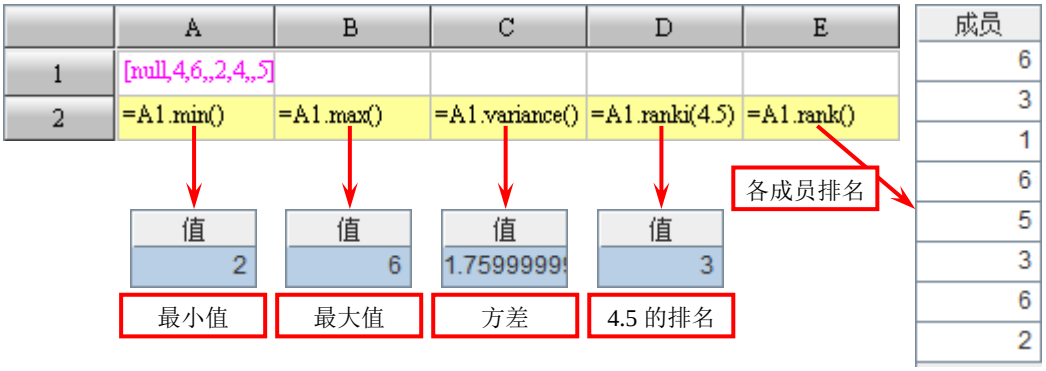
求 *y* 在序列中值的排名，不要求 *y* 是序列中成员的值。

❖ **@z** 从小到大排

➢ ***A*.rank()**

求序列中每个成员的排名。

❖ **@z** 从小到大排



➢ ***A*.conj()**

求序列 *A* 中所有成员的和列，如果 *A* 的某个成员不是序列，则作为 1 序列处理。

➢ ***A*.union()**

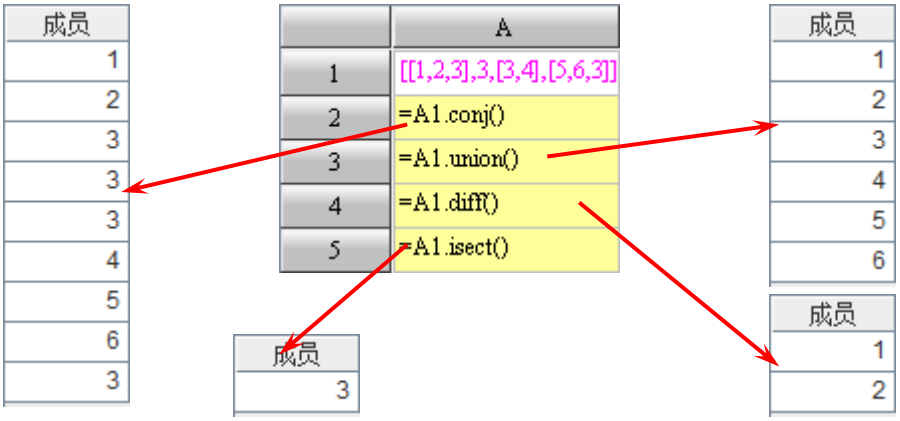
求序列 *A* 中所有成员的并列，如果 *A* 的某个成员不是序列，则作为 1 序列处理。(类似 SQL 中的 **UNION** 运算符的效果，结果序列中不会出现重复成员)

➢ ***A*.diff()**

依次将序列*A*中的成员做差列计算，如果*A*的某个成员不是序列，则作为1序列处理。

➢ ***A*.isect()**

求序列 *A* 中所有成员的交列，如果 *A* 的某个成员不是序列，则作为 1 序列处理。

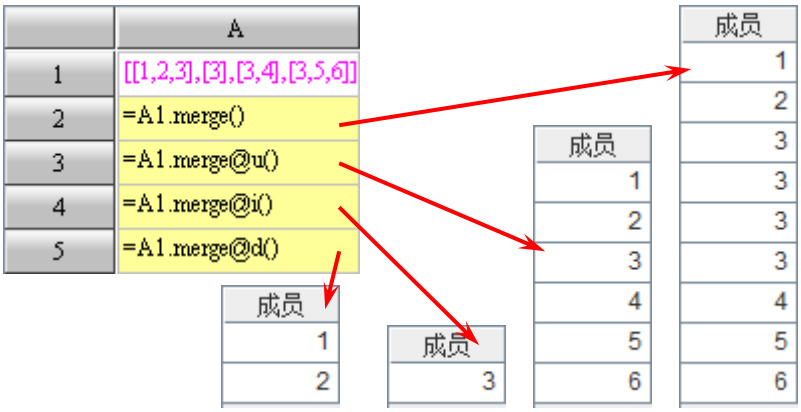


➢ ***A*.merge(x_i,…)**

归并计算。*A* 是序列的序列，且每个序列 *A*(*i*)均对于[x_i,…]有序，归并合并所有 *A* 中的

成员，合并后获得的序列对于[x_i,…]仍然有序。x_i 省略时，要求 $A(i)$中成员有序，合并后获得成员有序的序列。

- **@u** 归并时求并列
- **@i** 归并时求交列
- **@d** 归并时求差列



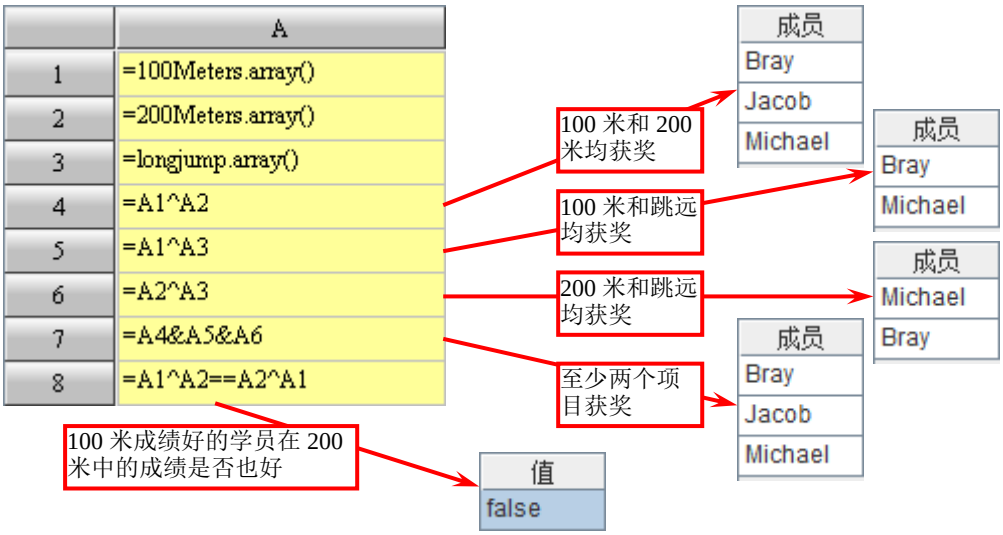
注意，**merge** 函数要求 *A* 的所有成员均为序列。

3.8 序列的使用

3.8.1 名单统计

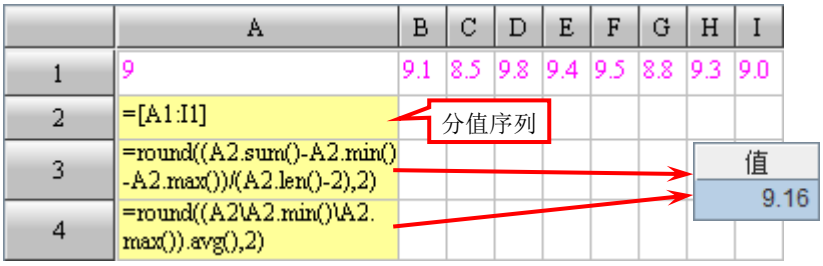
运动会中 100 米、200 米、跳远的优胜者（前 8 名）作为参数（名单是用逗号分隔的姓名字串），计算：

- 有两项及以上获奖的运动员
- 100 米跑得快的优胜者是否 200 米也跑得更快



3.8.2 裁判打分（1）

体操比赛中裁判打分，去掉最高分和最低分后计算平均分。



3.8.3 裁判打分（2）

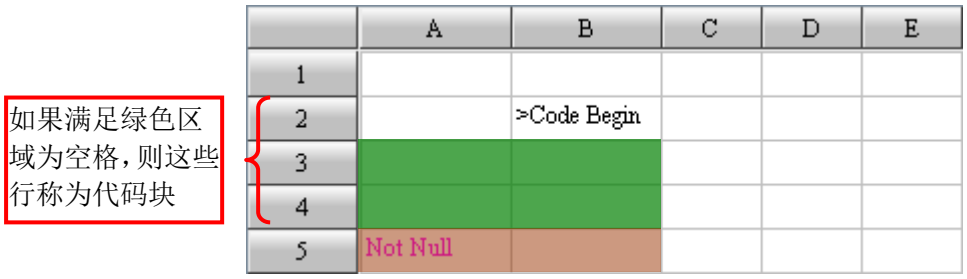
裁判级别不同，有不同的加权指数，计算加权平均分。

4. 代码块与流程控制

4.1 代码块

网格中一片缩进的单元格称为**代码块**，其起始格称为代码块的**主格**。

如果满足：主格所在行下面的各行中，主格所在列及该列左侧的所有单元格均为空格，则这些行及主格所在行一起构成主格的代码块；直到在某一行中，主格所在列或其左侧的任一个单元格非空，则自该行起的各行均不再属于代码块。



如上图中，如果绿色区域均为空格，则第 2 至第 4 行即为主格 B2 的代码块；而红色区域中的 A5 或 B5 中的任一个格子为代码格、计算格、执行格、常数格甚至注释格，都意味着从第 5 行起不再是 B2 的代码块。

代码块的主格一般都是**语句格**，即格串是以保留字起头的语句代码。

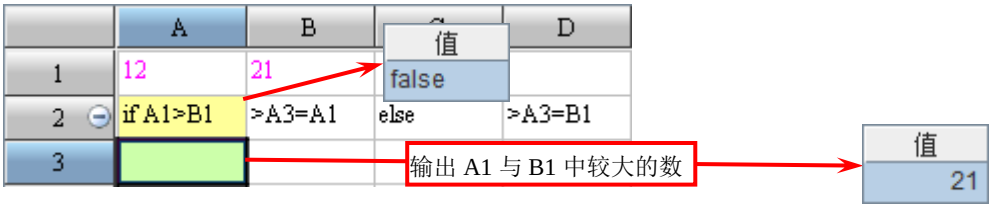
与 JAVA 等高级语言不同，集算器利用直接的代码块格式决定语句的作用范围，而不使用符号(如{})或保留字（BEGIN/END）将作用范围括起来。

4.2 if/else 判断

分支语句 if/else 有如下几种格式：

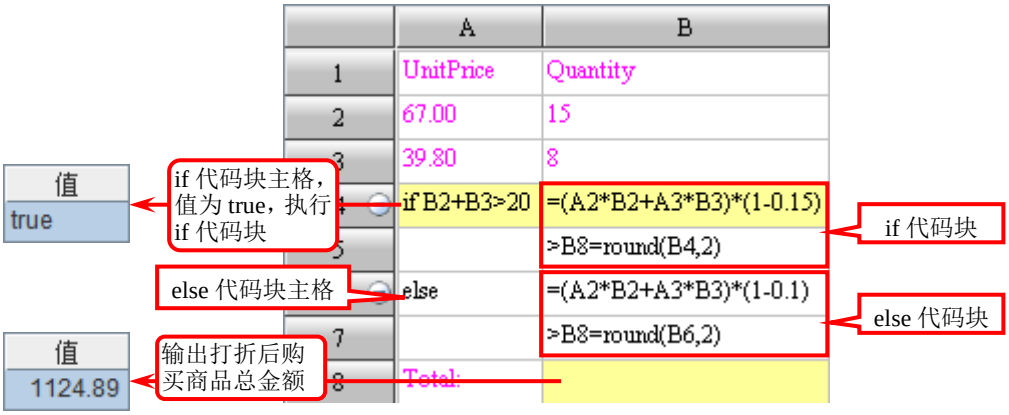
- **单行的 if *x* … else …**

x 成立时执行其后语句，否则执行 **else** 后的语句，**else 部分可省略**。**else** 和 **if** 必须写在同一行上。执行后，**if** 所在格的格值是 *x* 的计算结果。



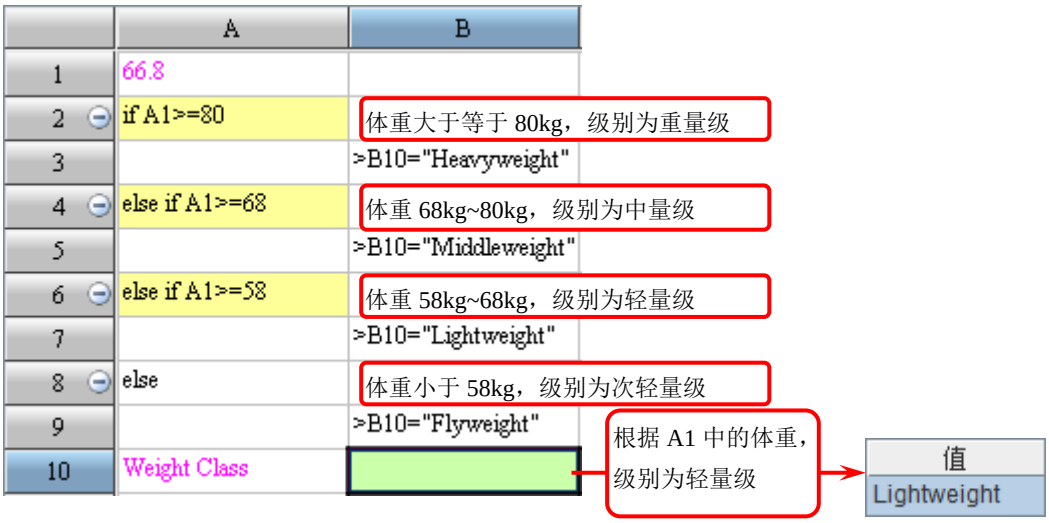
- **代码块的 if *x* … else …**

x 成立时执行 **if** 的代码块，否则执行 **else** 的代码块，**else** 部分可省略。**else** 和 **if** 必须写在同一列上。



- **多块的 if *x* … else if *y* …**

可一直重复写下去，**else if** 必须写在同一格内。再次强调，没有对应的 **end if** 语句，集算器利用代码块范围决定 if 语句何时结束。



在判断语句中，可以使用下面的一些逻辑连接符：

- ***a&&b***

a 与 *b*，当且仅当条件 *a* 与 *b* 同时成立时，结果才为 true。

- ***a||b***

a 或 *b*，当且仅当条件 *a* 与 *b* 中至少一个成立时，结果即为 true。

- ***!a***

非 *a*，当且仅当条件 *a* 不成立时，结果才为 true。

4.3 for 循环与 next，break

循环语句 for 将重复执行以其为主格的代码格，有如下几种格式：

- **for**

死循环，格值依次为当前循环计数

- **for *n***

循环 *n* 遍，格值依次为当前循环计数

- **for *A***

循环序列 *A* 的成员，格值依次为当前 *A* 的成员

- **for *x***

当 *x* 为真时循环，格值为 *x* 计算值

在循环体内：

- ***#C***

当前循环读数，*C* 是循环体的主格，即 **for** 所在单元格

- **next**

跳过当前最内层循环体剩下的代码，直接进入下一轮循环

- **next *C***

进入下一轮以 *C* 为主格的循环体

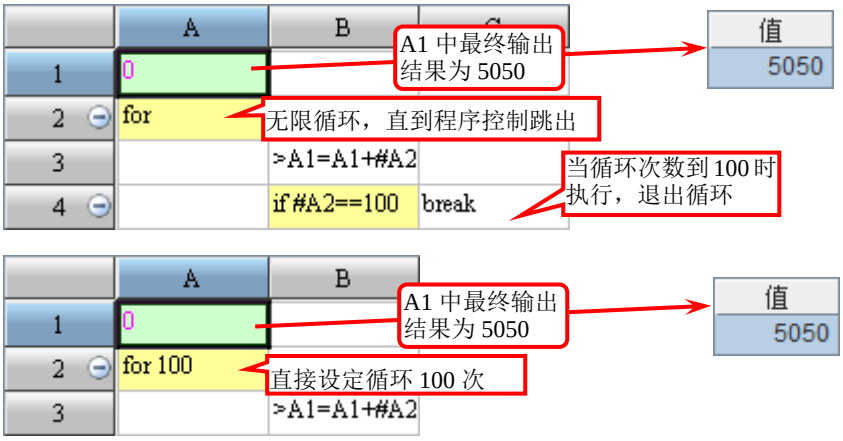
➢ **break**

跳出当前最内层循环体到代码块后面的单元格去执行

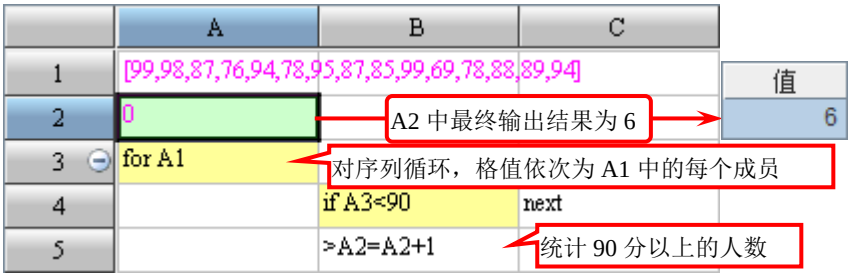
➢ **break *C***

跳出以 *C* 为主格的循环体

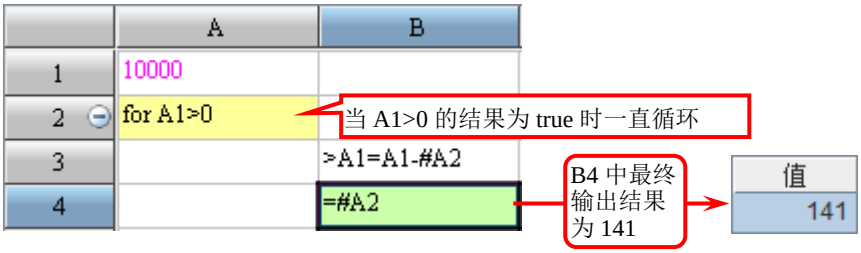
- 计算从 1 累加到 100：



- 统计分数在 90 分以上的人数：



- 计算从 1 累加到多少时刚好大于 10000：



4.4 func C,arg/func/return 子程序

➢ **func**

定义子程序，子程序代码为 **func** 为主格的代码块。调用子程序时的参数抄在 **func** 所在单元格及其右侧单元格中。

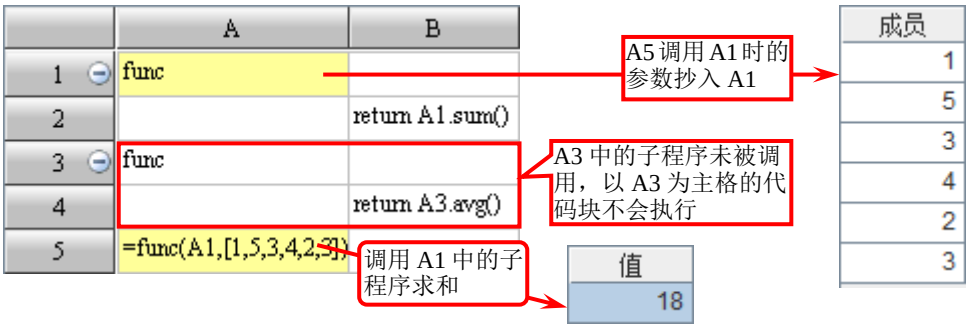
➢ **return *x***

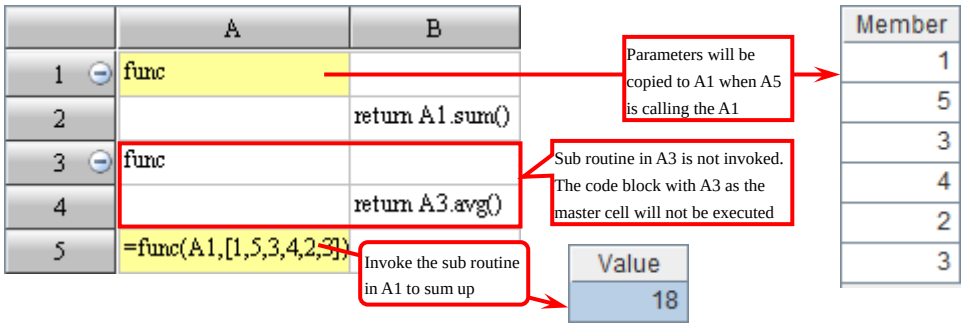
从子程序返回结果 *x* 并结束子程序执行，当无返回值时则返回 null。

➢ **func(*C*, x_i,…)**

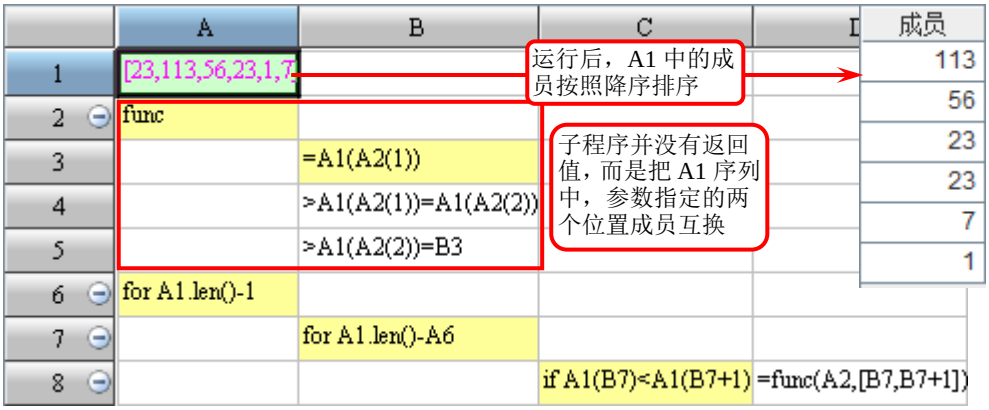
以参数 x_i,…调用起始于单元格 *C* 的子程序，返回值填入本格；也可以写为函数模式 func(*C*, x_i,…)。调用子程序时，多个参数会从 *C* 开始依次向右填充。

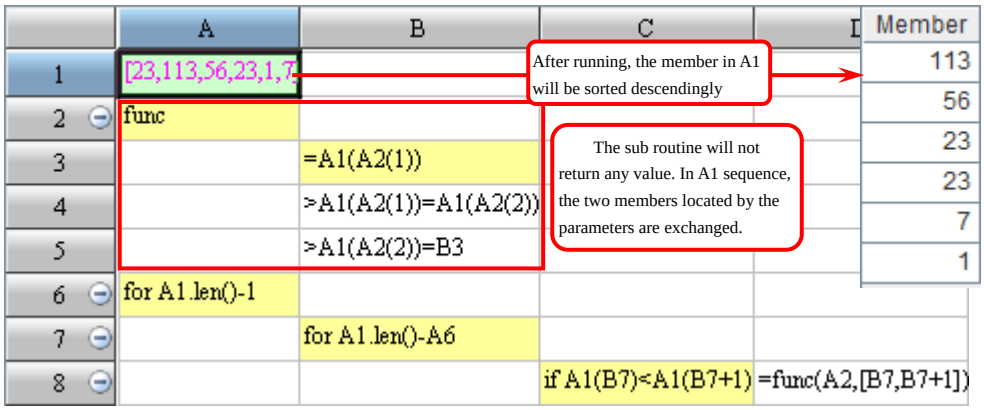
在网格运行时，只有当子程序被调用时，子程序所在的代码块才会被执行：



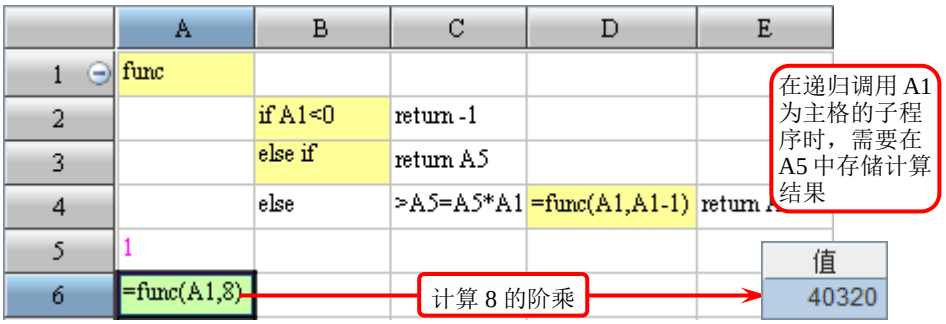


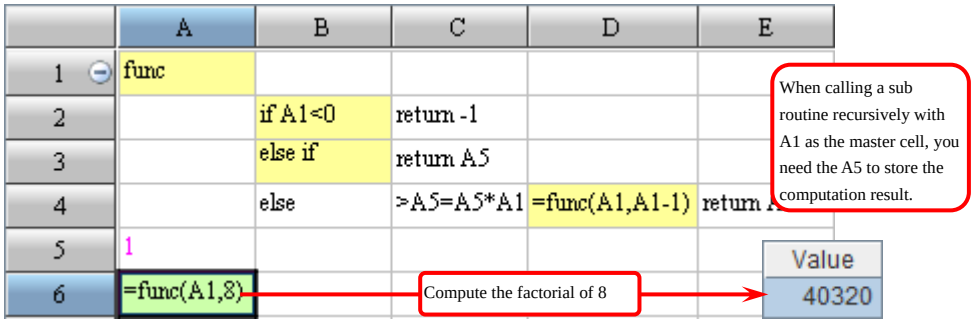
子程序不一定有返回值，当子程序代码块中没有 **return** 命令时，调用时会顺次执行到代码块结束为止。





集算器没有局部变量，所有的变量在整个网格内都是可以调用的，因此原则上无法写出真正的递归程序，但并未主动禁止递归调用。



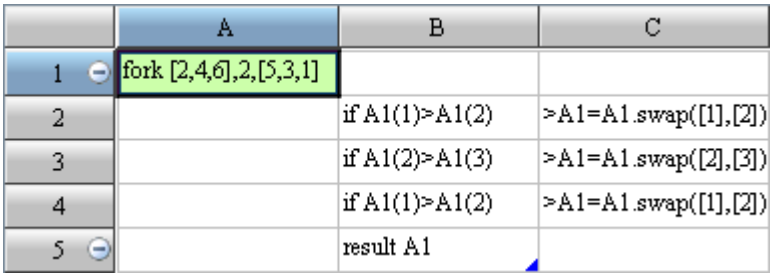


4.5 fork

- **fork A_i,…**

用多线程执行本格的代码块，其中 A_i,…为序列参数。各个参数的序列长度应该是相等的，如果其中包含单值参数，则将被视为成员都相同的序列来使用。执行时，当前的网格状态将被复制到各个线程中分别执行，参数会被拆分后分别设入各个线程，代码块中用 result 返回的结果将被拼成序列后返回到主线程。

与一般的 for 循环不同，fork 循环中，各个循环互不相关，互不影响，类似于多线程调用子程序，如：



A1 中的命令，相当于分为 3 个线程计算，依次使用 2,2,5；4,2,3；6,2,1 作为参数，代码块中将序列从小到大排序后返回，A1 中结果如下：

Member
[2,2,5]
[2,3,4]
[1,2,6]

4.6 网格程序的执行与调试

在集算器中，可以使用菜单栏下按钮栏中的按钮，来控制运行以及调试网格程序：

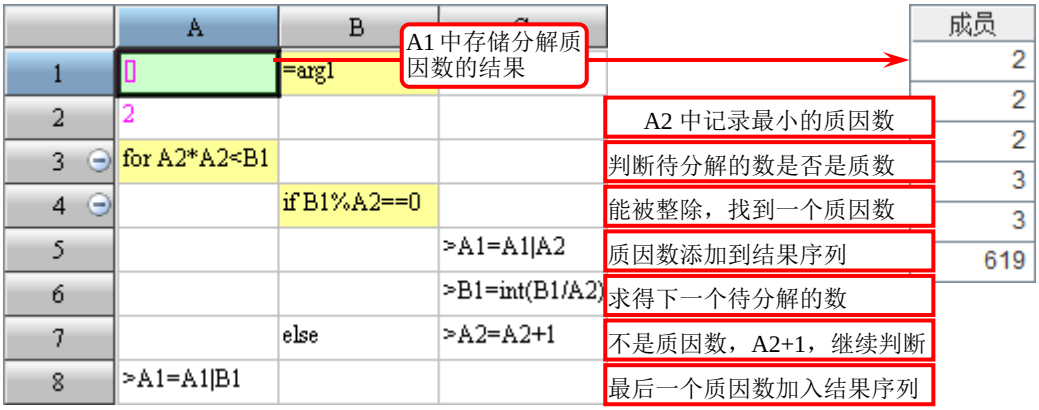
- ✧ 执行  执行整个网格中的程序
- ✧ 调试执行  执行程序到下一个设定的断点前
- ✧ 单步执行  执行到下一个存在表达式的单元格前
- ✧ 执行到光标  执行到光标所在的单元格前
- ✧ 暂停执行  暂停当前的执行状态，暂停后，单击执行按钮可以继续执行
- ✧ 停止执行  停止当前的执行状态，停止后，单击执行按钮会重新执行
- ✧ 设置/取消断点  在光标所在的单元格中，设置或者取消断点

func 语句可以被单步执行进去。

4.7 判断/循环等函数的使用

4.7.1 分解质因数

以自然数为输入参数，分解质因数后写入某个序列中。



5. 排列与序表

5.1 序表与记录

集算器继承了关系数据库中的数据表概念，称为**序表**。与关系数据库的概念一致，每个序表也有其自身的**数据结构**，由若干**字段**构成。序表的成员也称为**记录**。

序表与关系数据表的关键不同点：

- ✧ 序表同时是一个序列，因此其**成员之间有明确的次序**，故称序表
- ✧ 序表的字段没有数据类型，不同记录的同一字段取值**数据类型可以不同**，如果所有记录的某个字段数据类型均相同，则这个字段称为序表的一个**纯字段**。
- ✧ 序表的字段不必须有名字，可以**用序号访问**

与关系数据表有根本的不同，**序表的记录可以被提出来作为数据对象引用**，而序表可以看作是由记录构成的序列，可以象普通序列一样单独访问其成员。

- ***T*(1)** 序表 *T* 的第 1 条记录。
- ***T*(to(3))** 序表 *T* 的前 3 条记录构成的序列。

但是，为保证数据结构的一致性，序表成员**不能像普通序列那样赋值**。

- ***T*(1)= *x*** 出错，不允许执行。

序表中的每一条记录，都会存储所在序表的引用，共用这个序表的数据结构。

序表成员修改需要使用特殊函数，将在 **6.3 修改数据**中详细讲述。

5.2 排列

把序表记录取出后构成的序列称为**排列**（如前面的 ***T*(to(3))**），作为一个序列，排列的成员可以用前面讲的办法赋值。

排列的成员**不一定来自同一个序表**，由同一序表中记录构成的排列称为**纯排列**。

排列和序表看起来都是记录构成的序列，但有着根本的不同：

- ✧ 序表实际保存了记录的值，任何记录必须属于且仅属于某个序表，记录不可以脱离序表单独存在
- ✧ 排列保存的是记录的引用而非实际值，同一记录可以从属于多个排列，也可以在同一排列内重复出现

排列是个为叙述方便而引入的概念，集算器中并没有专门针对排列的计算，也不检查排列的合法性，比如，可以将排列的某个成员赋值成不是记录的值。

5.3 序表的创建

5.3.1 从文件中读写序表

在读写序表时，可以使用多种常见的文件格式，如 txt 文件，xml 文件，csv 文件等。

➢ ***f*.import(*F*$_i$:*type*$_i$,…;*s*,*b*:*e*)**

从文件 *f* 中读出序表返回，参数缺省时，import 函数要求文件由换行符（\n）分隔行，Tab（\t）分隔列，每行对应一条记录，每列对应一个字段。可以用 *b*:*e* 来定义读取时的数据范围，为从第 *b* 个字节至第 *e* 个字节。可以用 *F*$_i$,…指定需要读出的字段，并可以指定该字段的数据类型为 *type*$_i$，没有 *F*$_i$,…参数则将读出 *A* 的所有字段；也可以用 *s* 定义列分隔符。当 *type* 省略时，优先使用第一行的数据类型。

import 缺省将返回无字段名的序表，可用选项得到有字段名的序表：

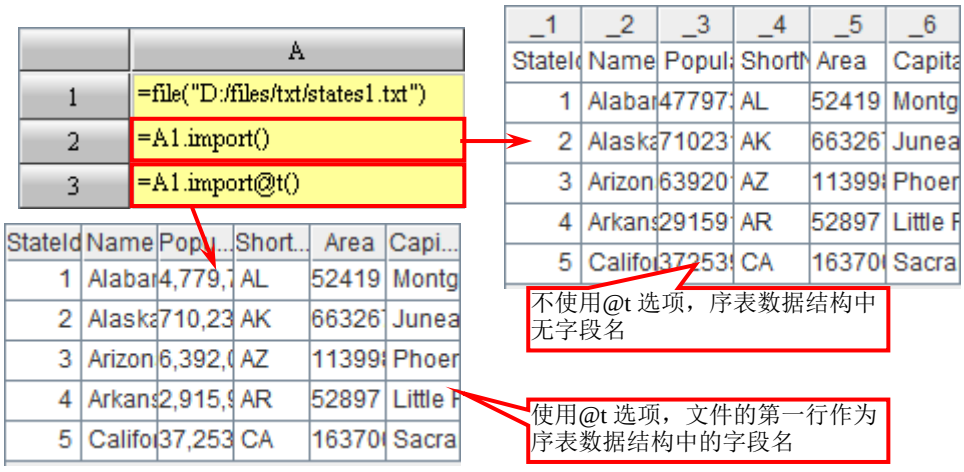
❖ **@t** 将文件的第一行作为返回序表的字段名。
❖ **@b** 从导出二进制文件中读取序表，此时**@t** 选项无效，且忽略 *type*,*s* 参数。
❖ **@z** 分块从文件中读取数据，此时 *b*,*e* 分别为分块序号和总块数。
❖ **@s** 不拆分字段，读取为单字段的字符串构成的序表，此时将忽略参数。

➢ ***f*.write(*A*)**

将字串序列 *A* 写入到文件 *f* 中，每个成员占一行。

❖ **@a** 追加写。

StateId	Name	Population	ShortName	Area	Capital
1	Alabama	4779736	AL	52419	Montgomery
2	Alaska	710231	AK	663267	Juneau
3	Arizona	6392017	AZ	113998	Phoenix
4	Arkansas	2915918	AR	52897	Little Rock
5	California	37253956	CA	163700	Sacramento



和读出序表类似，集算器还可以将序表输出到指定文件。

➢ ***f*.export(*A*, *x*:*F*,…;*s*)**

根据 *A* 计算表达式 *x* 作为字段 *F*,…写出到文件 *f* 中。没有 *x*,…参数则将写出 *A* 的所有字段，格式仍为 Enter 分隔行，Tab 分隔列。*f*.**export** 函数缺省导出 txt 文件。*s* 为列分隔符，缺省情况下为 Tab。

❖ **@t** 以第一条记录的字段名作为标题写入文件。
❖ ***@a*** 追加写，如果省略则是覆盖原文件中的内容，与**@t** 互斥。
❖ **@b** 导出二进制压缩文件，导出二进制文件时效率更高，与**@t@a** 选项互斥。
❖ **@bg** 导出二进制文件时，使用按组分段规则。

和 **import** 函数略有不同，**export** 函数只处理那些可以转化成字串的字段，不能字串化的字段值（如记录、排列等）将被忽略。

5.3.2 直接创建序表

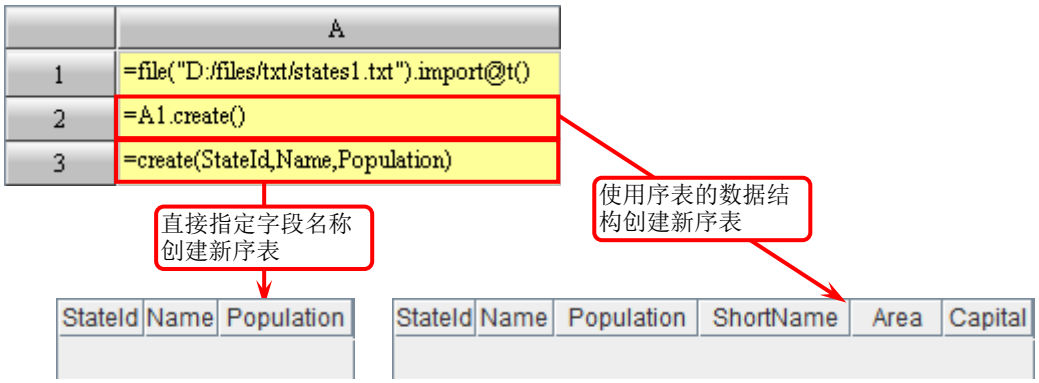
所谓空序表，是指只有数据结构定义而没有记录的序表。可以使用函数直接创建空序表。

➢ **create(*F*$_i$,…)**

创建一个以 *F*$_i$,…为字段的空序表。

➢ ***T*.create()**

复制序表 *T* 的数据结构创建空序表。

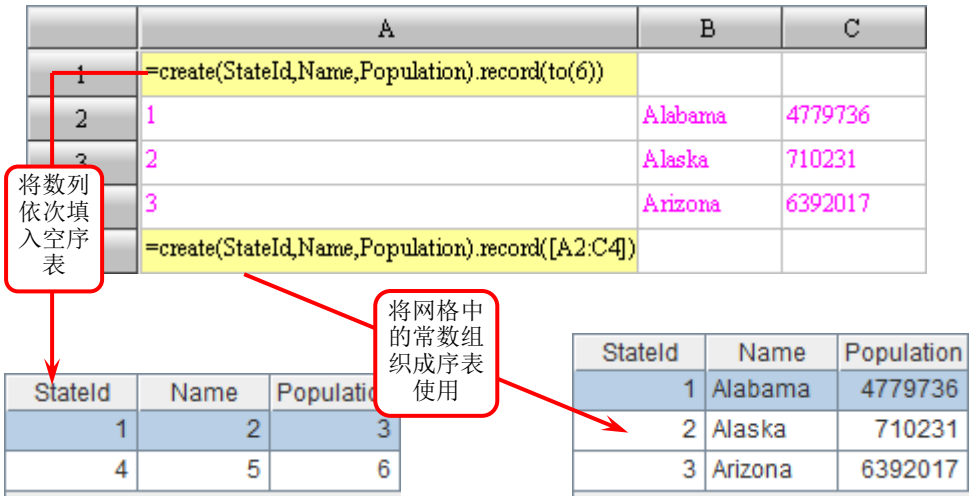


➢ ***T*.record(*A*,*k*)**

将序列 *A* 的成员依次填进序表的各个字段形成记录。当 *k* 缺省或 *k* ==0 时，在序表所有记录之后增添记录；当 *k*!=0 时，从指定位置的记录开始修改 *T* 中的数据。

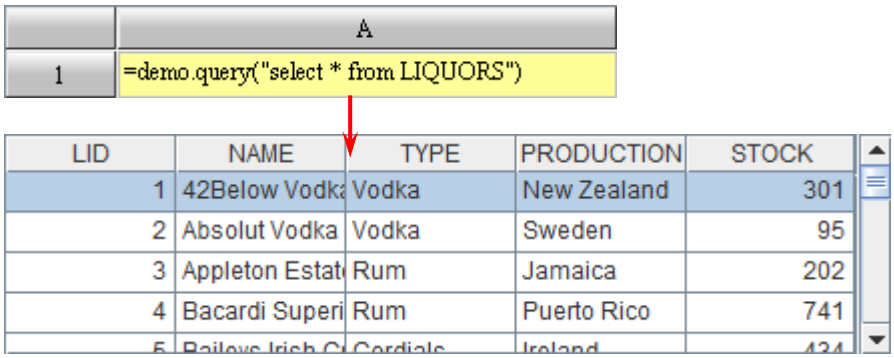
❖ **@i** 当 *k*!=0 时，在指定记录前插入新数据而不是修改。

create 函数配合 **record** 函数可以将网格上的常数组织成序表使用。



5.3.3 从数据库中读出序表

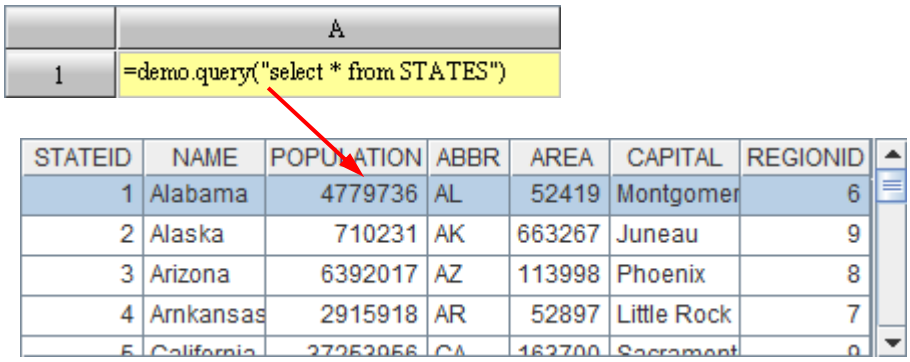
由于序表和数据库中的数据表结构相似，因此，可以用数据库中读出的数据构成序表。



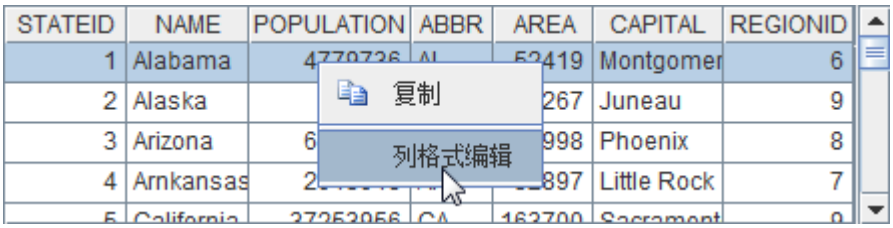
从数据库中读出序表将在 **8.2.1 用 SQL 读取序表**中详细叙述。

5.4 如何查看序表或排列

点击序表所在的单元格，可在值显示区查看序表的值：(本例中，A1 单元格中的序表)



在查看序表时，在记录选定的字段上按下右键，还可以**设置序表中该字段的显示格式**。



在弹出的显示格式设定窗口中进行设置：

设置后，序表将按照设定的显示格式显示：

STATEID	NAME	POPULATION	ABBR	AREA	CAPITAL	REGIONID
1	Alabama	4,779,736	AL	52419	Montgomer	6
2	Alaska	710,231	AK	663267	Juneau	9
3	Arizona	6,392,017	AZ	113998	Phoenix	8
4	Arnkansas	2,915,918	AR	52897	Little Rock	7

排列的查看，与查看序表类似。但是，在排列中，各个记录的数据结构不尽相同，在展现时，会按照第一条记录的数据结构来显示。数据结构不同的记录，只会显示其中同名字段的值。在查看排列时，是**不能**设定字段的显示格式的。

STATEID	NAME	POPULATION	ABBR	AREA	CAPITAL	REGIONID
1	Alabama	4779736	AL	52419	Montgome	6
2	Alaska	710231	AK	663267	Juneau	9
3	Arizona	6392017	AZ	113998	Phoenix	8
32	New York	8084316				
5	Los Angel	3798981				
13	Chicago	2886251				

5.5 记录与序表的访问

5.5.1 记录的访问

- ***T(i)***

序表或者排列是由记录所构成的序列，所以可以直接按位置访问记录

5.5.2 记录字段的访问与赋值

与大多数结构化程序设计语言一样，字段的访问采用.操作符。

- ***r.F***

返回记录 *r* 的字段 *F* 的值。

- ***r.F=x***

将记录 *r* 的字段 *F* 赋值为 *x*。

字段还可以用序号访问，显然，没有名字的字段只能这种方法访问：

- ***r.#i***

返回记录 *r* 的第 *i* 个字段的值。

- ***r.#i=x***

将记录 *r* 的第 *i* 个字段赋值为 *x*。

- **r.field(*i*,*x*)**

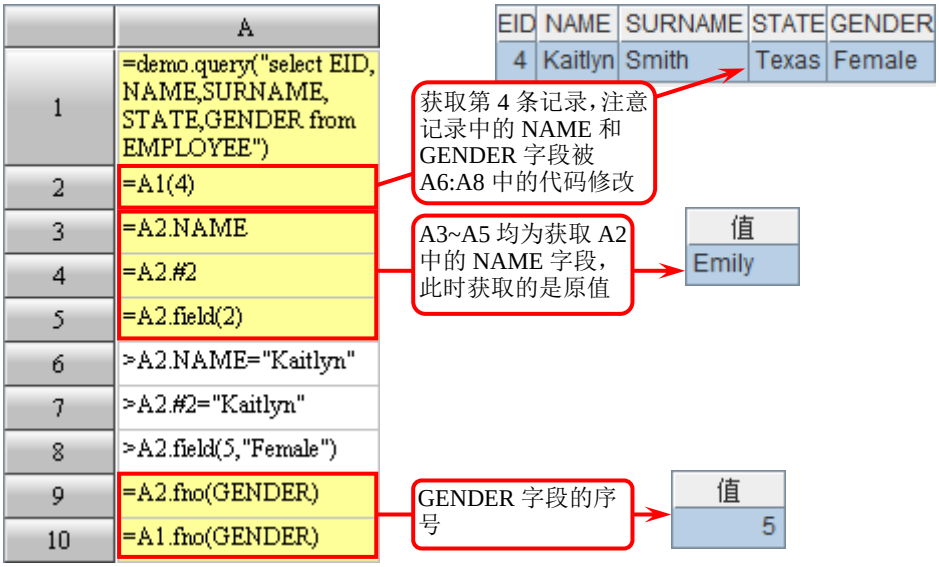
将记录 *r* 的第 *i* 个字段赋值为 *x*。

- ***r*.fno(*F*)，*T*.fno(*F*)**

记录 *r* 中或者序表 *T*，字段 *F* 的序号。当 *F* 省略时，返回序表的字段数量。

- **r.field(*i*)**

记录 *r* 中，第 *i* 个字段的值。

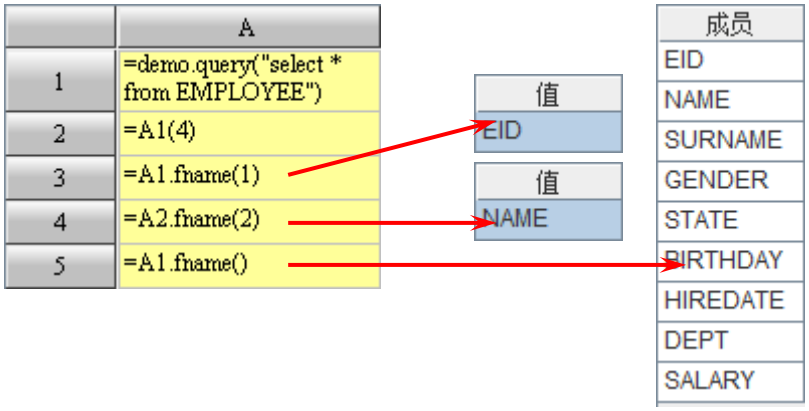


➢ ***T*.fname(*i*)**

序表 *T* 中，第 *i* 个字段的名称，返回为字符串。

➢ ***r*.fname(*i*)**

记录 *r* 的数据结构中，第 *i* 个字段的名称，返回为字符串。



5.5.3 记录/序表的类型判断

➢ **ifr(*x*)**

判断 *x* 是否记录，返回 true/false。

➢ **ift(*x*)**

判断 *x* 是否序表，返回 true/false。

- **ifr(A1)** A1 中的计算结果是否是记录。
- **ift(A3)** A3 中的计算结果是否是序表。

5.6 循环

5.6.1 序列循环中表达式的约定

循环函数参数中可以引用序列成员，约定规则：

➢ ~

当前序列成员，也称为**基成员**

➢ #

当前成员的序号

- *A*.sum(~*~)　　计算 *A* 中成员的平方和。
- *A*.max(if(#%2==0,~,0))　　找出偶数位置成员的最大值。

➢ ***A*[*i*]、~[*i*]**

相对引用，返回循环当前成员之后序号相差 *i* 的成员，*i* 可以是负数。

- ***A*.(~-~[-1])**　　计算 *A* 中每个成员与前一个成员的差所构成的序列。

5.6.2 序列的循环函数

针对序列的每个成员做某种计算的函数称为**循环函数**，一般形式为 ***A*.*f*(*x*)**，实际上，序列的聚会运算也是循环函数，如 **3.7.4 序列的聚合运算**中的函数如 ***A*.sum()**等。

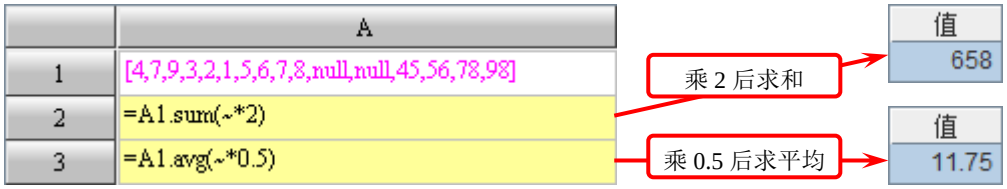
循环函数的计算行为由函数名决定，如 **sum** 表示求和、**avg** 表示平均、…。

➢ ***A*.sum(*x*)**

对 *A* 每个成员计算表达式 *x*，计算结果的和。

➢ ***A*.avg(*x*)**

对 *A* 每个成员计算表达式 *x*，计算结果的平均值。



返回相关序列：

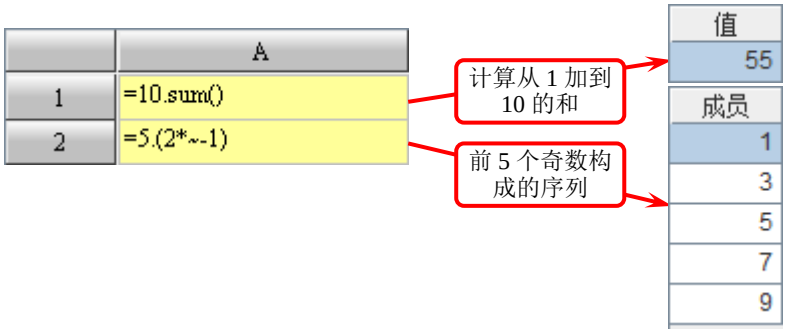
➢ ***A*.(*x*)**

返回针对 *A* 中每个成员计算表达式 *x* 后构成的序列。当 *x* 省略时，直接返回 *A*。

整数循环：

➢ ***n*.*f*(*x*)**

to(*n*).*f*(*x*)的简写形式。



5.6.3 排列循环计算的表达式约定

➢ ***F***

在表达式中引用字段，当在排列中引用字段时，排列中的所有记录都必须拥有该字段。

➢ ***r.F, ~.F, A.F***

在表达式中引用指定记录的字段，基成员的字段，或者是指定排列的字段。

特别强调：如果内存中有与字段同名的变量，则不可以在循环函数中使用该字段的省略写法，必须写全~.*F* 或 *A.F*，仅写 *F* 将被解释为同名的变量。

使用 *A.F* 时，如果未循环计算 *A*，则取 *A*(1).*F*。

循环函数还提供相邻成员引用方案：

➢ ***A[i], ~[i]***

在表达式中引用当前记录后的第 *i* 条记录，其中 *i* 可以是负值，当引用的成员越界时返回 null。

➢ ***A{a:b}, ~{a:b}***

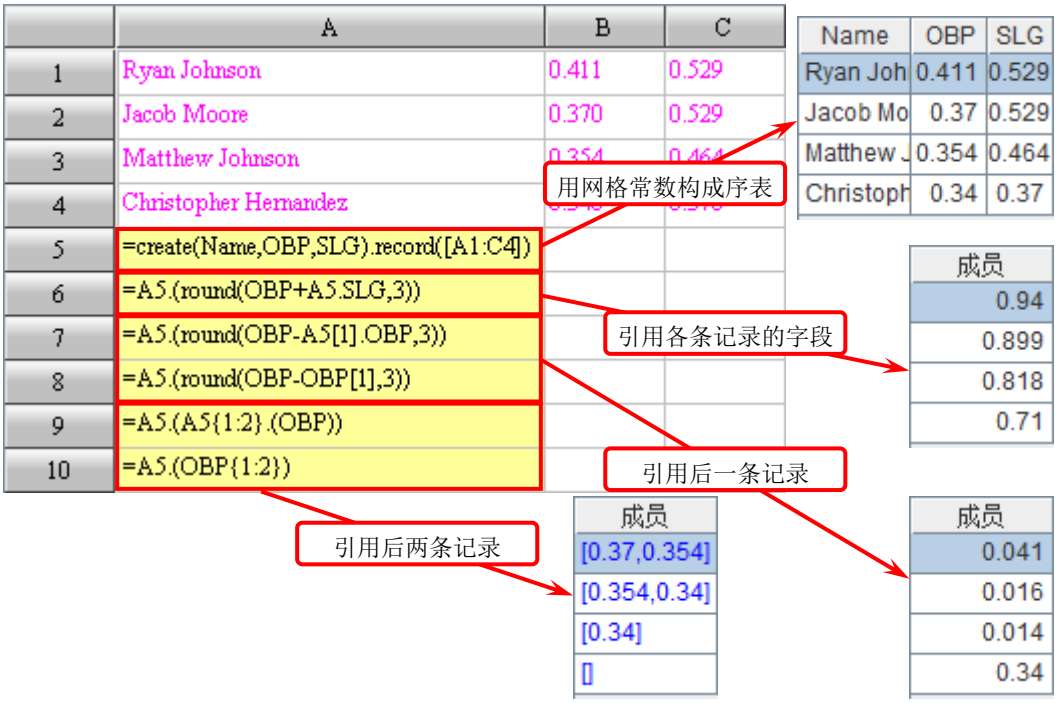
在表达式中引用当前记录后的第 *a* 条到第 *b* 条记录组成的排列，其中 *a* 或 *b* 可以是负值。*a* 缺省时，将从 *A* 中的第 1 条记录开始；*b* 缺省时，将至 *A* 中的最后一条记录为止。

➢ ***F[i]***

相当于 *A*[*i*].*F*，在表达式中引用当前记录后的第 *i* 条记录的 *F* 字段

➢ ***F{…}***

相当于 *A*(…).(*F*)，先根据表达式在 *A* 中获取记录后，再计算 *F* 字段组成的序列。

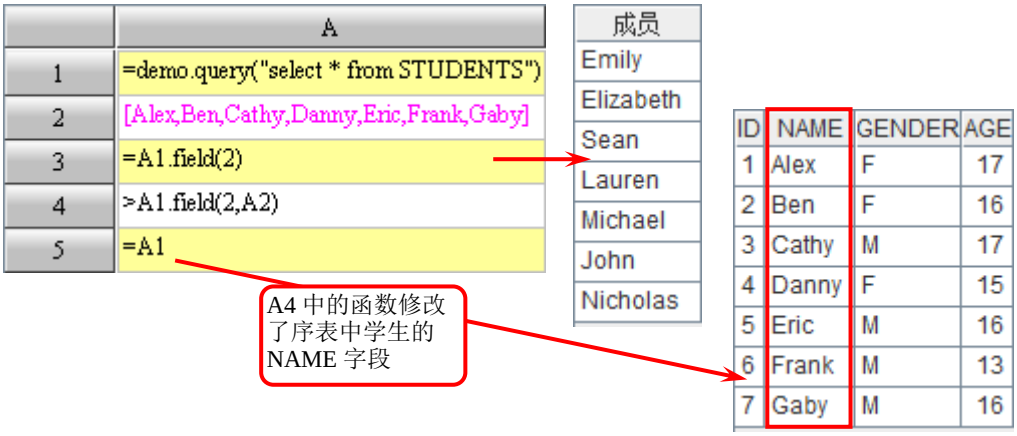


➢ ***A.field(i)***

返回排列 *A* 中，所有记录第 *i* 个字段所构成的序列。

➢ ***A*.field(*i*,*x*)**

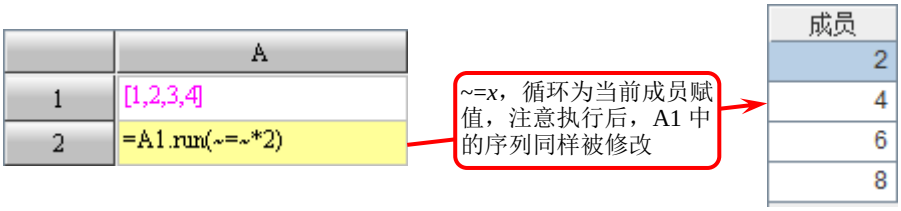
表达式 *x* 的计算结果通常为序列，将该序列中的值依次赋值给排列 *A* 中每条记录的第 *i* 个字段。



5.6.4 run 函数

➢ ***A*.run(*x*)**

循环函数计算 *x*，但仍返回序列 *A*。



针对记录也可以使用 **run** 函数，在多次引用字段时简化书写。

➢ ***r*.run(*x*)**

针对记录 *r* 计算 *x*，返回 *r*

- *r*.**run(金额=单价*数量)** 相当于 ***r*.金额=*r*.单价**r*.数量**

➢ ***r*.(*x*)**

针对记录 *r* 计算 *x*，返回 *x*

- ***r*.(单价*数量)** 相当于 *r*.单价**r*.数量

run 函数虽然可以用于普通序列，但是它一般用于对排列中的记录做赋值操作。在 run 函数用于排列中的记录赋值时，集算器还提供了更清晰地写法：

➢ ***A*.run(x_i:F_i,…)**

将字段 F_i 赋值为 x_i，相当于 ***A*.run(F_i = x_i,…)**。

- ***A*.run(年龄+1:年龄)** 相当于 ***A*.run(年龄=年龄+1)**

这种形式的 run 函数也可以同时对多个字段赋值，依次写下去即可：

- ***A*.run(年龄+1:年龄,年龄-入学年龄:学龄)**

	A	B
1	42-Inch 1080p LCD TV - Black	1110.99
2	50-Inch 1080p Plasma HDTV	1099.95
3	42-Inch 1080p Plasma HDTV	799.95
4	51-Inch 1080p 600Hz 3D Plasma HDTV	1299.99
5	=create(TV,Original,Off,Price).record([A1:D4])	
6	/>A5.run(Off=round(Original*0.15,2), Price=Original-Off)	>A5.run(round(Original*0.15,2):Off, Original-Off:Price)

TV	Original	Off	Price
42-Inch	1110.99	166.65	944.34
50-Inch	1099.95	164.99	934.96
42-Inch	799.95	119.99	679.96
51-Inch	1299.99	195.0	1104.99

A6 与 B6 中代码是等价的，使用 run 函数计算出降价 15%后，减价金额及折后价

5.6.5 迭代

循环函数可以迭代使用，即在计算表达式中再使用循环计算。

➢ ***A*1.*f*1(*A*2.*f*2(*x*))**

在循环计算时，*x* 中的符号将优先解释为从属于 A_2。

- **10.(#.sum(~*~))** 从 1 到 10 累计平方和序列

在嵌套的循环函数中，~、#将解释为里层序列的当前成员和序号，而引用外层序列时需冠以序列名称，写作 ***A*.~**、***A*.#**。

- ***A*.max(*B*.count@b(*A*.~==~))** *A* 中成员在 *B* 中出现最多的次数

	A	B	C	D
1	Nicholas Johnson	47.8	Light	48
2	Jacob Williams	55.8	Bantamweight	54
3	Michael Harris	56.1	Featherweight	57
4	Tyler Anderson	68		
5	=create(Name,Weight).record([A1:B4])			
6	=create(Class,Weight,Count).record([C1:E3])			
7	>A6.run(Count=A5.count(Weight<=A6.Weight && (Weight>A6[-1].Weight)))			

A7 的表达式中，Weight 如果不写明 *A.F*，则优先解释为从属于 A5，A7 功能为计算各级别运动员总数，A6 中的运算结果如下：

Class	Weight	Count
Light flyweight	48	1
Bantamweight	54	0
Featherweight	57	2

5.6.6 汇总循环

➢ **~~**

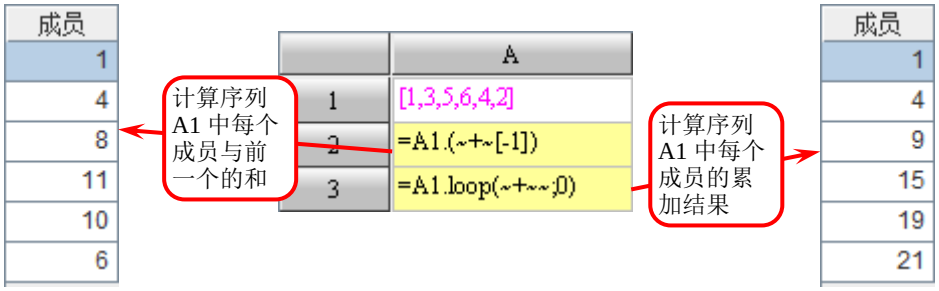
在汇总循环语句中调用，返回上一次运算表达式 *x* 的结果。

➢ ***A*.loop(*x*;*a*)**

汇总循环，对序列 *A* 进行循环，计算表达式 *x* 的结果构成序列。表达式 *x* 中可以使用~~来引用 *x* 上一次的计算结果，~~的初始值为 *a*。使用 **loop** 函数可以完成累加及汇总等循环计算。

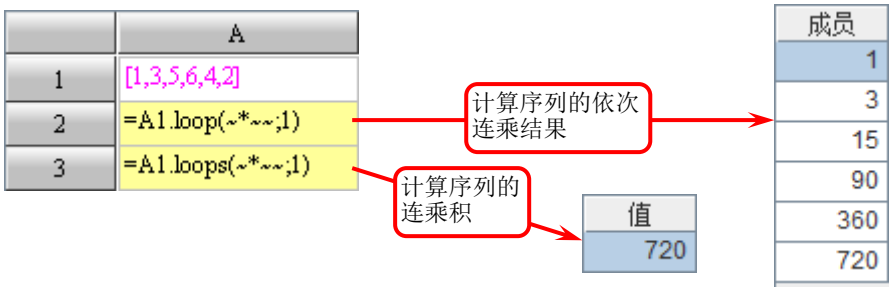
使用~~进行引用，返回的是表达式 *x* 上一次的计算结果，这和相对引用是不同的，~[-1]

返回的是序列 *A* 中的前一个成员。



➢ ***A*.loops(*x*;*a*)**

计算 ***A*.loop(*x*;*a*)**，返回最后一次表达式 *x* 的运算结果。



使用 ***A*.loops(*x*;*a*)**，可以计算序列汇总循环的最终结果。

5.7 排列的基本计算

5.7.1 定位

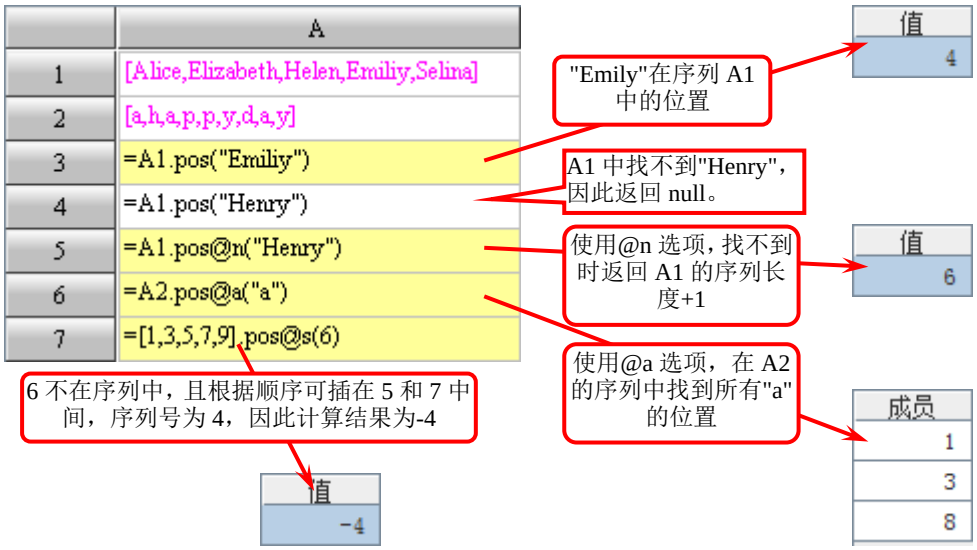
定位函数，可以从序列或排列中，找出指定成员或者是满足条件成员的位置，定位函数有下面两个通用选项：

- **@a** 返回所有满足条件的成员位置序列，否则只返回其中的第一个位置。
- **@z** 在查找时从后向前进行。

➢ ***A*.pos(*x*)**

返回序列 *A* 中成员 *x* 的序号，其中 *x* 不是序列，找不到 *x* 时返回 null。

- **@b** *A* 中成员有序，查找时使用二分法。
- **@s** *A* 中成员有序，在用二分法查找时，当 *x* 为 *A* 成员时返回其位置；当 *x* 非 *A* 成员时，返回 *x* 按顺序可插入位置序号的相反数。
- **@n** 当找不到 *x* 时，返回 ***A*.len()+1**，而不是 null，与**@a** 选项互斥。



➢ ***A*.pmin(*x*)**

返回 *A* 中使得表达式 *x* 为最小值的成员序号。

- ***A*.pmin(单价*数量)** 总价最少的商品序号。

➢ ***A*.pmax(*x*)**

返回 *A* 中使得表达式 *x* 为最大值的成员序号。

- ***A*.pmax@a(胜利*3+平局)** 计算足球联赛积分最高的所有球队序号。

➢ ***A*.pselect(*x*)**

返回 *A* 中第一个满足条件 *x* 的成员序号。当 *x* 缺省时，返回 *A* 中所有成员的序号构成的序列。

❖ **@b** *A* 中成员对于表达式 *x* 有序，返回 *A* 中第一个满足 *x*==0 的成员序号，查找时使用二分法。

❖ **@s** *A* 中成员对于表达式 *x* 有序，在使用二分法查找时，如果 *A* 中的任何成员都无法使表达式 *x* 的计算结果为 0，那么返回满足条件的数值可插入位置的相反数。

❖ **@n** 当 *A* 中没有成员满足条件 *x* 时，返回 ***A*.len()+1**，与**@a** 选项互斥。

- ***A*.pselect(~>5)**
- **[1,2,3,4,5].pselect@bs(cmp(~,3.5))** 结果为-4，使用**@bs** 选项时，如果序列中所有成员都不能满足表达式，会返回负值。

➢ ***A*.pselect(x_k:y_k,…)**

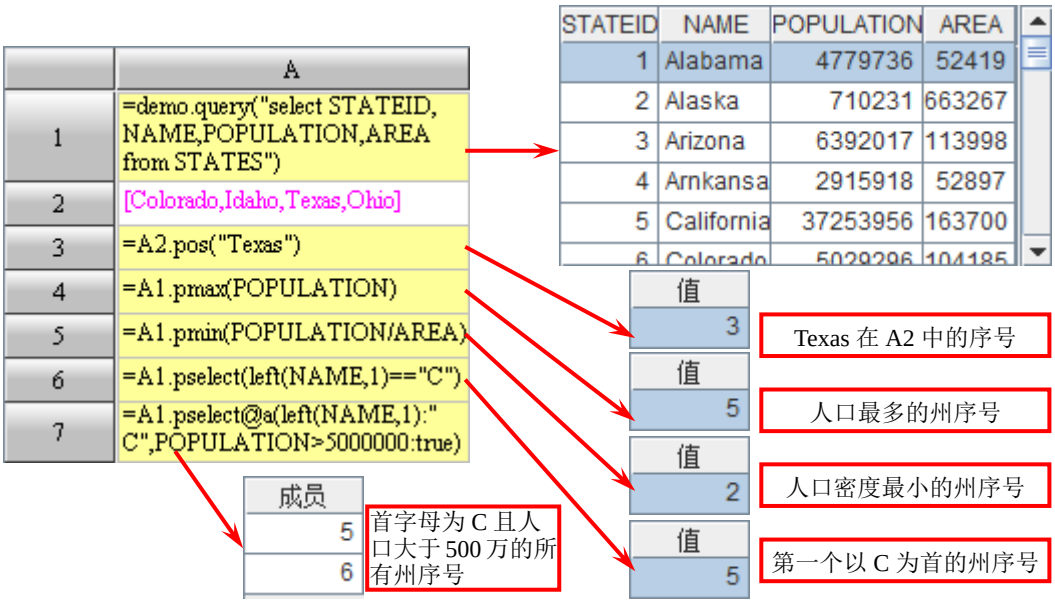
返回 *A* 中满足条件 x_k==y_k,…的第一个成员的序号。

❖ **@b** *A* 中成员对于表达式 x_k 全部递增或全部递减，查找时使用二分法。

❖ **@s** *A* 中成员对于表达式 x_k 全部递增或全部递减，使用二分法时，如果未找到满足条件的成员，则返回满足条件数值的可插入位置的相反数。

❖ **@n** 当 *A* 中没有成员满足条件时，返回 ***A*.len()+1**，与**@a** 选项互斥。

- ***A*.pselect(Gender:"M",left(Name,1):"C")** 找出姓名以 C 为首的首个男性员工序号。



在使用上面这些定位函数时，可以指定查找的起始位置：

➢ ***A. f(x,k)***

从第 *k* 个成员开始查找，当 *x* 缺省时，需要写为 ***A.f(,k)***。

- ***A.pos(1,5)*** 从第 5 个成员开始，查找 1 所在位置。

同时返回多个成员的位置，也可以使用 pos 函数，但是参数为序列。

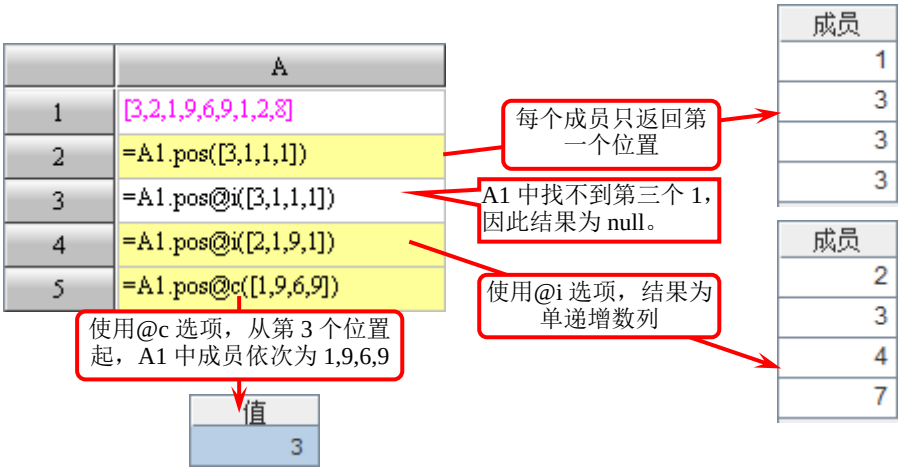
➢ ***A.pos(x)***

其中 *x* 为序列，返回数列 *p*，使得 *A*(*p*)==*x*，查找时，*x* 的每个成员只找到第一个位置。

❖ **@i** 返回单递增数列 *p*，使得 *A*(*p*)==*x*。

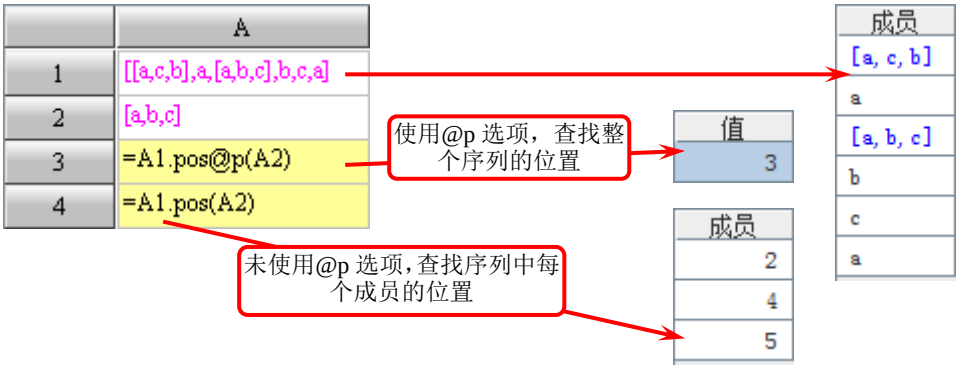
❖ **@b** *A* 有序，查找时使用二分法。

❖ **@c** 找到 *x* 的成员**依次**在 *A* 中出现的第一个位置。



特别的，当 *x* 为序列时，也可以查找序列成员 *x* 在 *A* 中的位置，此时 *A* 通常为序列的序列。

❖ ***A*.pos@p(*x*)** 返回序列成员 *p* 在中的位置。



➢ ***A*.in(*B*)**

序列 *A* 的所有成员都存在于序列 *B* 中，即 ***B*.posi@p(*A*)!=null**。

下面介绍一些组合函数，它们都会先根据条件搜索到所需位置，然后根据位置在指定序列中计算。

➢ ***A*.lookup(A_i:x_i,…)**

寻找第一个满足条件的 *k*，它使得序列 A_i,…中的第 *k* 个成员分别为 x_i,…，返回序列 *A* 中的第 *k* 个成员。

❖ **@a** 根据所有满足条件的位置，返回 *A* 的产生列。

➢ ***A*.sumif(A_i:x_i,…)**

寻找所有满足条件的 *k*，使得序列 A_i,…中的第 *k* 个成员分别为 x_i,…；返回序列 *A* 中这些位置成员值的和。

➢ ***A*.countif(A_i:x_i,…)**

寻找所有满足条件的 *k*，使得序列 A_i,…中的第 *k* 个成员分别为 x_i,…；对序列 *A* 中这些位置的成员计数后返回结果。

➢ ***A*.avgif(A_i:x_i,…)**

寻找所有满足条件的 *k*，使得序列 A_i,…中的第 *k* 个成员分别为 x_i,…；返回序列 *A* 中这些位置成员值的平均值。

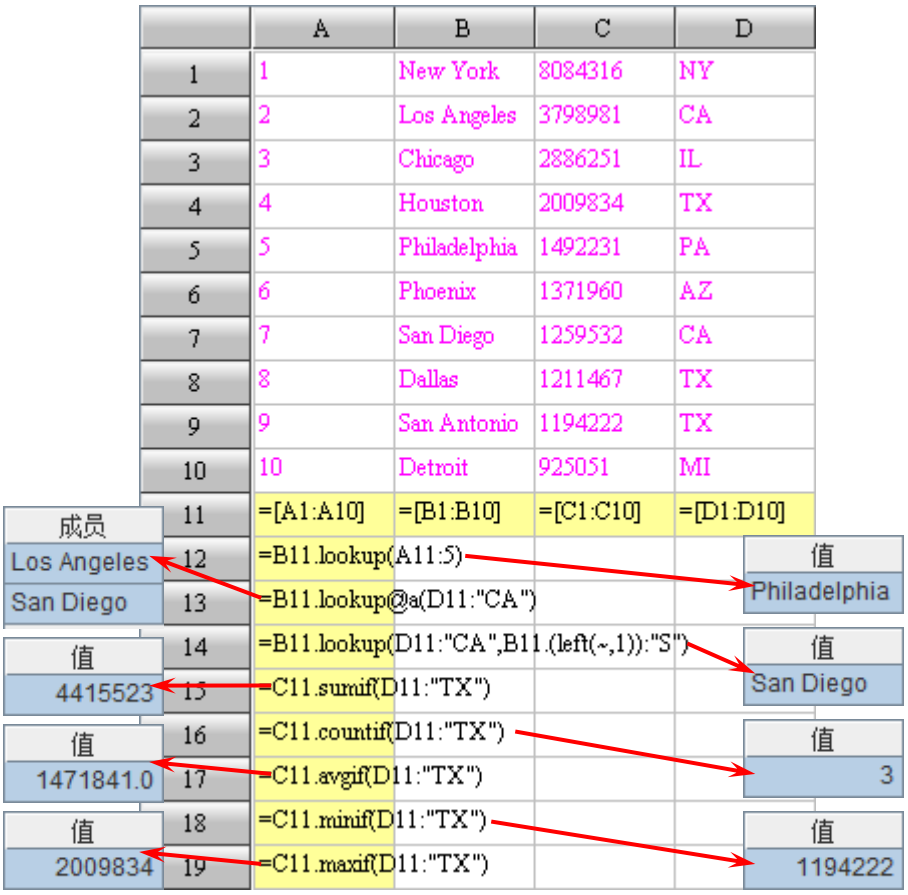
➢ ***A*.minif(A_i:x_i,…)**

寻找所有满足条件的 *k*，使得序列 A_i,…中的第 *k* 个成员分别为 x_i,…；返回序列 *A* 中这些位置成员值的最小值。

➢ ***A*.maxif(A_i:x_i,…)**

寻找所有满足条件的 *k*，使得序列 A_i,…中的第 *k* 个成员分别为 x_i,…；返回序列 *A* 中这些位置成员值的最大值。

这些组合函数通常用在数据直接存储在网格中的情况下，可以通过指定条件查找到所需数据。如：



网格中的数据是美国人口数量排名前 10 位的城市信息。A12 中找到排名第 5 的城市名称；A13 中找到其中加利福尼亚州的所有城市名称；A14 中找到加利福尼亚州且名称首字母为S的城市名；A15计算数据中德克萨斯州城市的人口总数；A16计算其中德克萨斯州的城市总数；A17~A19 分别计算数据中德克萨斯州城市人口的平均值、最小值和最大值。

在例子中，计算使用的序列由网格直接构成，在使用时也可以由序表计算获得。

5.7.2 选出

选出函数，可以从序列或排列中，选出指定位置或者是满足指定条件的成员或者记录，选出函数有下面四个通用选项：

- **@1** 返回第一个满足条件的成员。注意，这里的**@1** 为数字 1 而不是字母 l，实际上，集算器中不存在字母 l 的选项。
- **@a** 返回所有满足条件的成员序列。
- **@z** 在查找时从后向前进行。

➢ ***A*.minp(*x*)**

返回 *A* 中使得表达式 *x* 为最小值的成员。该函数默认选项为**@1**，如果需要返回满足条件的所有成员，可以使用**@a** 选项。

- ***A*.minp(单价*数量)** 总价最少的商品。

➢ ***A*.maxp(*x*)**

返回 *A* 中使得表达式 *x* 为最大值的成员。该函数默认选项为**@1**，如果需要返回满足条件的所有成员，可以使用**@a** 选项。

- ***A*.maxp@a(胜利*3+平局)** 计算足球联赛积分最高的所有球队。

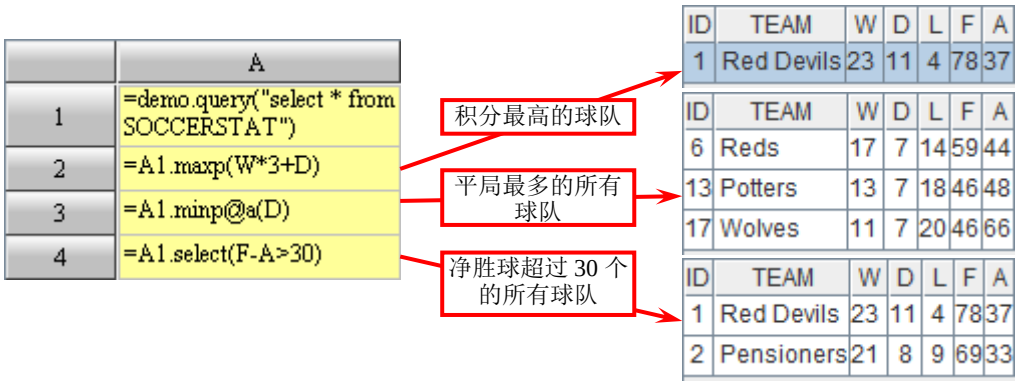
➢ ***A*.select(*x*)**

返回 *A* 中满足条件 *x* 的所有成员构成的序列，*x* 省略时返回 *A* 中所有成员。

❖ **@b** 查找时使用二分法，此时选出条件为 *x*==0，且要求 *A* 对于 *x* 有序。

- ***A*.select(~>5)** 选出序列 *A* 中大于 5 的成员。

与 **pselect** 不同，**select** 函数缺省情况下返回所有满足条件的成员，而 **pselect** 缺省只返回第 1 个。***A*.select(*x*)**可以定义为***A*(*A*.pselect@a(*x*))**。如果只需返回第一个满足条件的成员，可以使用**@1** 选项。



5.7.3 排序

➢ ***A*.psort(*x*)**

将 *A* 中成员排序，使得表达式 *x* 的结果递增，返回各个成员在 *A* 中的序号序列；*x* 省略时，将 *A* 中成员进行排序，并返回序号序列。

➢ ***A*.psort(x_i:d_i,…)**

将 *A* 中成员依次以表达式 x_i 的结果排序，返回各个成员在 *A* 中的序号序列。排序时，d_i 为每次排序的方向，1 或省略为递增，-1 为递减，0 为原序。

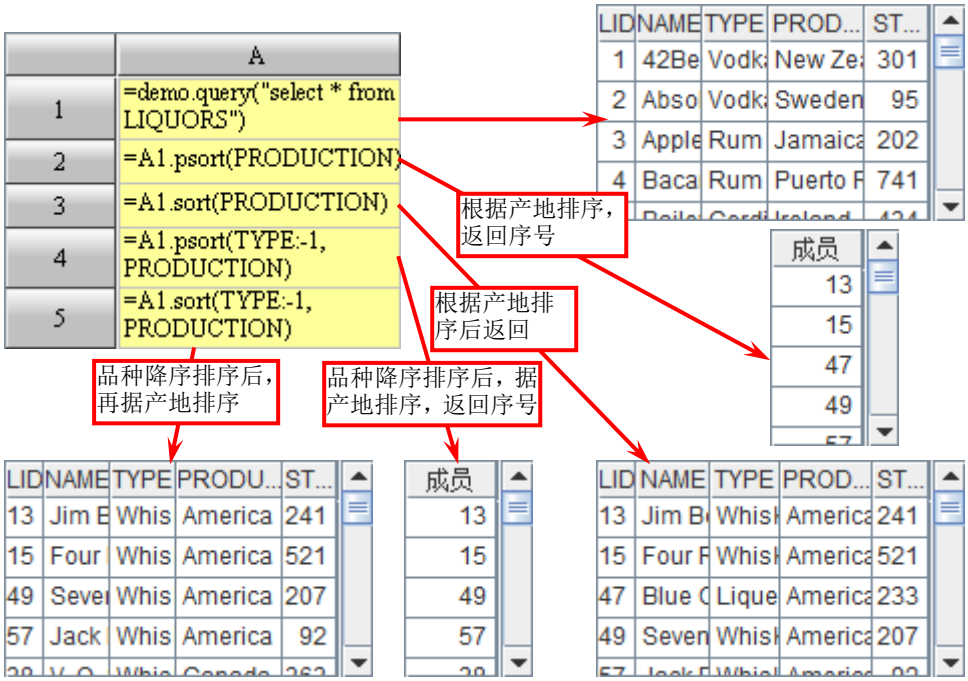
❖ **@i** 返回 ***A*.psort(...).inv()**。逆数列的相关内容请看 **5.8.2 逆数列与逆序列**了解。

➢ ***A*.sort(*x*)**

将 *A* 中成员排序后返回，使得表达式 *x* 的结果递增。当 *x* 省略时，将 *A* 中的成员进行排序。

➢ ***A*.sort(x_i:d_i,…)**

将 *A* 中成员依次以表达式 x_i 的结果排序后返回。排序时，d_i 为每次排序的方向，1 或省略为递增，-1 为递减，0 为原序。



➢ ***A*.sort(…;*loc*)**

用指定的本地语言排序，*loc* 为语言名。

- ***A*.sort(;"zh")** 用中文对 *A* 排序，除了语言以外，*loc* 中还可以指定地区，如"en"，"zh_TW"，"en_GB"等

➢ ***A*.ptop(*x*…;*n*)**

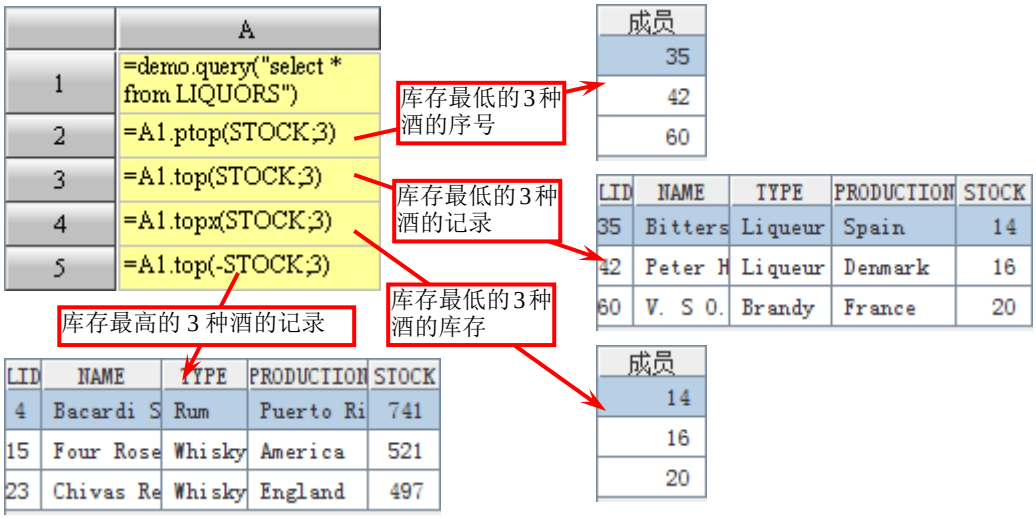
将 *A* 中成员排序，返回使得表达式 *x* 的结果最小的 *n* 个成员在 *A* 中的序号序列。

➢ ***A*.top(*x*…;*n*)**

将 *A* 中成员排序，返回使得表达式 *x* 的结果最小的 *n* 个成员。

➢ ***A*.topx(*x*…;*n*)**

将 *A* 中成员排序，返回表达式 *x* 的结果最小的 *n* 个结果。



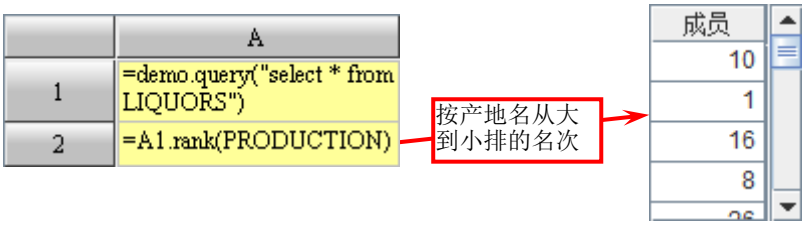
5.7.4 排名

在序列中可以使用聚合运算中的 ***A*.ranki(*y*)**以及 ***A*.rank()**函数计算排名，在排列中，也可以进行类似计算。

- ***A*.rank(*x*)**

对排列 *A* 中每个成员计算表达式 *x* 的排名，即 ***A*.(*x*).rank()**。

❖ **@z** 从小到大排



5.8 排列的其它计算

5.8.1 定位计算

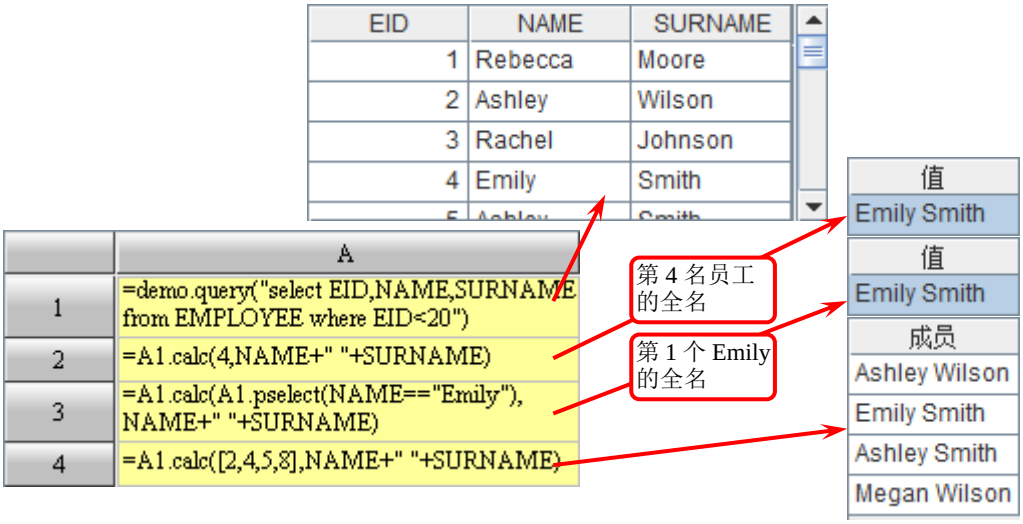
定位计算函数可以在序列指定位置的成员处计算表达式，并返回结果。定位计算函数虽然要求在序列上进行，但是通常是用来完成基于排列的计算的。

- ***A*.calc(*k*,*x*)**

在排列 *A* 中的第 *k* 条记录上，计算表达式 *x* 并返回结果。

- ***A*.calc(*p*,*x*)**

对于排列 *A*，在序号在数列 *p* 中的各条记录上，计算表达式 *x*，并返回结果所构成的序列。



5.8.2 逆数列与逆序列

一个长度为 *k* 的数列 *p*'，初始值均为 0，对于数列 *p* 循环，对 *p*'进行赋值，如果 *p* 中第 *i* 个位置的成员为 *p*(*i*)，则将 *p*'中，第 *p*(*i*)个位置的成员赋值为 *i*。最终获得的 *p*'即为 *p* 的**逆数列**。

在序列 *A* 中，以数列 *p* 等长的逆数列 *p*'为位置数列，所获得的 *A* 的产生列，称为 *A* 关

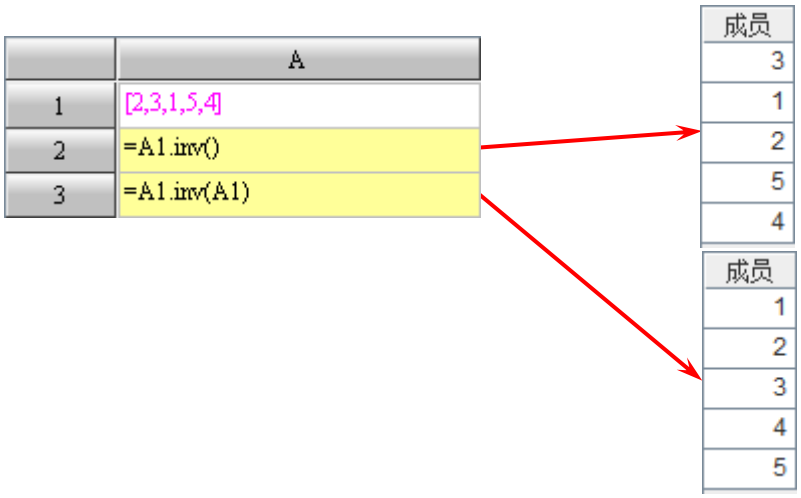
于 *p* 的逆序列。

- ***p*.inv(*k*)**

  返回长度为 *k* 的，数列 *p* 的逆数列；*k* 省略时，返回与 *p* 等长的逆数列。

- ***A*.inv(*p*)**

  返回 *A* 关于 *p* 的逆序列。值得注意的是，如果数列 *p* 为 *n* 置换，那么 *p* 与 *p*.inv()互为逆数列，且 *p*.inv(*p*)会得到[1,2,3,…,*n*]。



5.9 排列的使用

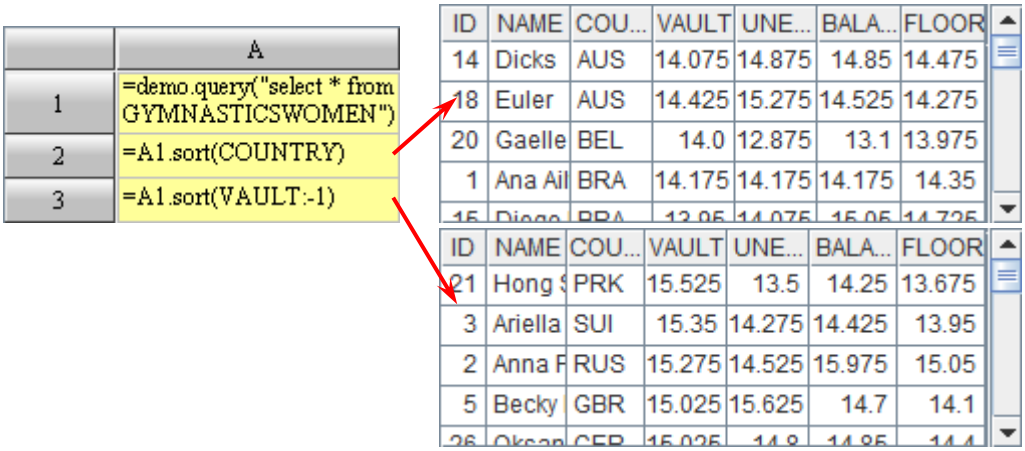
某次运动会中，女子体操全能的成绩存储在名为 GYMNASTICSWOMEN 的数据库表中：

ID	NAME	COUNTRY	VAULT	UNEVENBARS	BALANCEBEAM	FLOOR
1	Ana Ailva	BRA	14.175	14.175	14.175	14.35
2	Anna Pavlor	RUS	15.275	14.525	15.975	15.05
3	Ariella Kask	SUI	15.35	14.275	14.425	13.95
4	Barbara Keesling	SUI	14.525	14.125	15.075	14.3
5	Becky Down	GBR	15.025	15.625	14.7	14.1
6	Beth Tweddle	GBR	14.4	15.675	14.8	15.1

下面将使用集算器进行成绩统计。

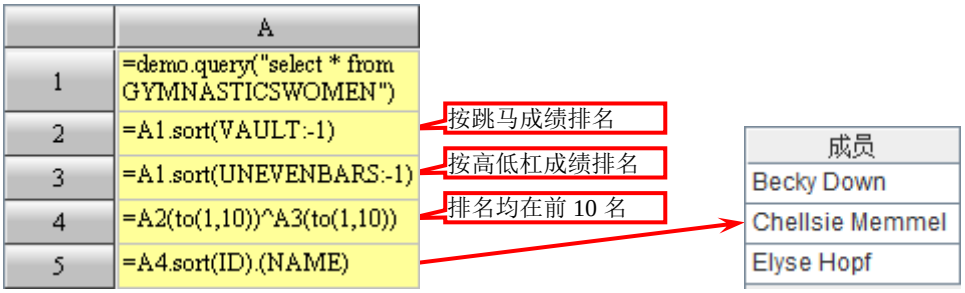
5.9.1 成绩统计 1

将运动员的成绩分别按照国籍和跳马的成绩进行排序：



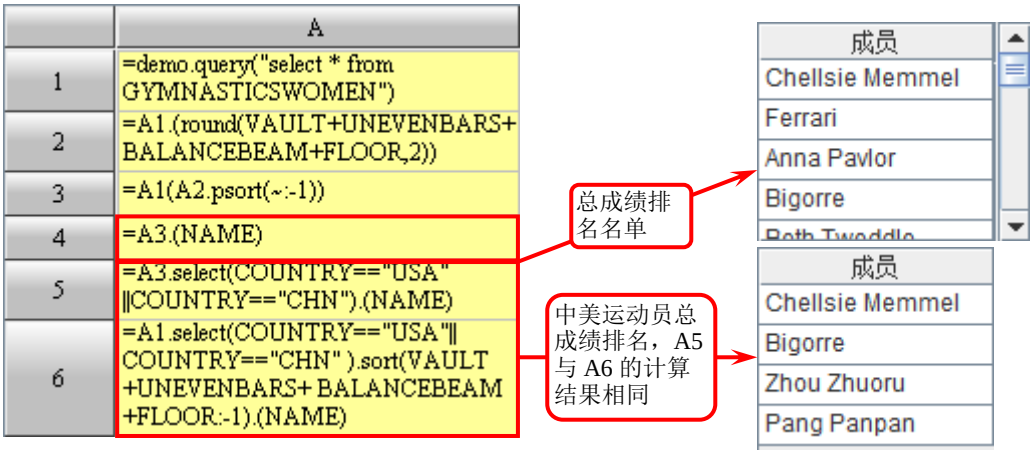
5.9.2 成绩统计 2

取得跳马和高低杠成绩都在前 10 名的运动员名单：



5.9.3 成绩统计 3

计算各运动员的总成绩，并获得总成绩排名名单，以及美国和中国运动员的总成绩排名名单：



6. 序表的维护与修改

6.1 计算字段

可以在序表或排列上添加计算字段，生成新序表后返回。

- ***A*.derive(x_i:F_i,…)**

为排列 *A* 添加字段 F_i,…生成新序表后返回，取值为 x_i,…。其中，*x* 是与 *A* 相关的表达

式，可以使用 *A* 的字段，省略时添加字段取值为 null，*F* 省略时将添加无名字段。

- ***A*.derive(单价*数量:金额)**　　在序表 *A* 上添加了"金额"字段后，生成新序表返回

	A
1	=demo.query("select * from RECEIPT")
2	=A1.derive(round(UNITPRICE*QUANTITY,2):Total)

derive 函数不改变原序表，在格子中生成新序表

NAME	UNITPRICE	UNIT	QUANTITY	Total
Apple	1.69	LB	1.2	2.03
Banana	0.69	LB	3.23	2.23
Cucur	0.77	EACH	3.0	2.31
Onion	0.99	LB	1.33	1.32
Orange	4.99	BAG	1.0	4.99
Red Grape	0.99	LB	2.87	2.84

6.2 记录引用

6.2.1 记录引用

序表中记录的字段没有数据类型，可任意取值，如果将其值赋为另一条记录，则可以方便地实现外键引用。

设 *A* 是部门表，*B* 是员工表，执行：

- ***A*.run(经理=*B*.select@1(编号:*A*.经理))**

序表 *A* 的字段经理将变成序表中的记录，此时可用：

- ***A*.select(经理.性别:"女")**　　列出经理为女性的部门。

这样能避免使用 SQL 中的多表连接，使书写更清晰且计算更快。

	A
1	=demo.query("select CID,NAME,POPULATION,STATEID as STATE from CITIES")
2	=demo.query("select STATEID,NAME,ABBR,CAPITAL from STATES")
3	=A1.run(STATE=A2.select@1(STATEID==STATE))

将城市信息中的 STATE 字段值设定为STATES 表中的记录引用

CID	NAME	POPULATION	STATE
1	New York	8084316	32
2	Los Angeles	3798981	5
3	Chicago	2886251	13
4	Houston	2009834	43

STATE 字段值为记录引用，双击可查看

STATEID	NAME	ABBR	CAPITAL
13	Illinois	IL	Springfield

6.2.2 排列引用

利用字段取值的任意性，还可将字段赋值成一个排列。

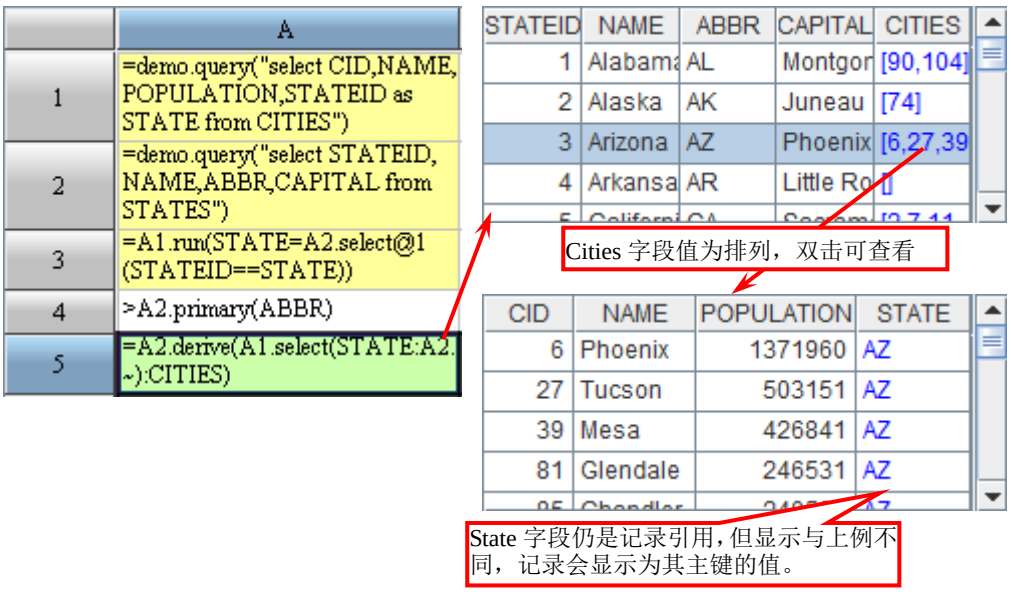
设有部门表 *D* 和员工表 *E*，执行：

- ***D*.derive(*E*.select(部门号:*D*.编号):员工)**

由 *D* 生成新序表，添加一个"员工"字段，其值为员工表记录构成的排列。

- ***D*.select@1(部门:"销售部").员工.avg(年龄)**

即可计算出销售部员工的平均年龄。



和记录引用类似，排列引用也可以避免掉 **SQL** 中连接运算，达到书写清晰和计算迅速的作用。不同的是，记录引用一般是在子表引用主表记录，而相反地，排列引用则一般是在主表引用子表记录。

6.3 修改数据

6.3.1 添加、删除记录

➢ ***T*.insert(*k*,x_i:F_i,…)**

在序表 *T* 的第 *k* 条记录前插入记录，字段 *F* 赋值为 *x*，*k* 为 0 表示在最后插入，*F* 省略表示依次赋值；*x* 和 *F* 均省略表示插入空记录。

- ***T*.insert(0,"David":Name,28:Age,"M":Gender)**
- ***T*.insert(2,"Julia","F",25)**

➢ ***T*.delete(*k*)**

删除在序表 *T* 的第 *k* 条记录。

➢ ***T*.delete(*p*)**

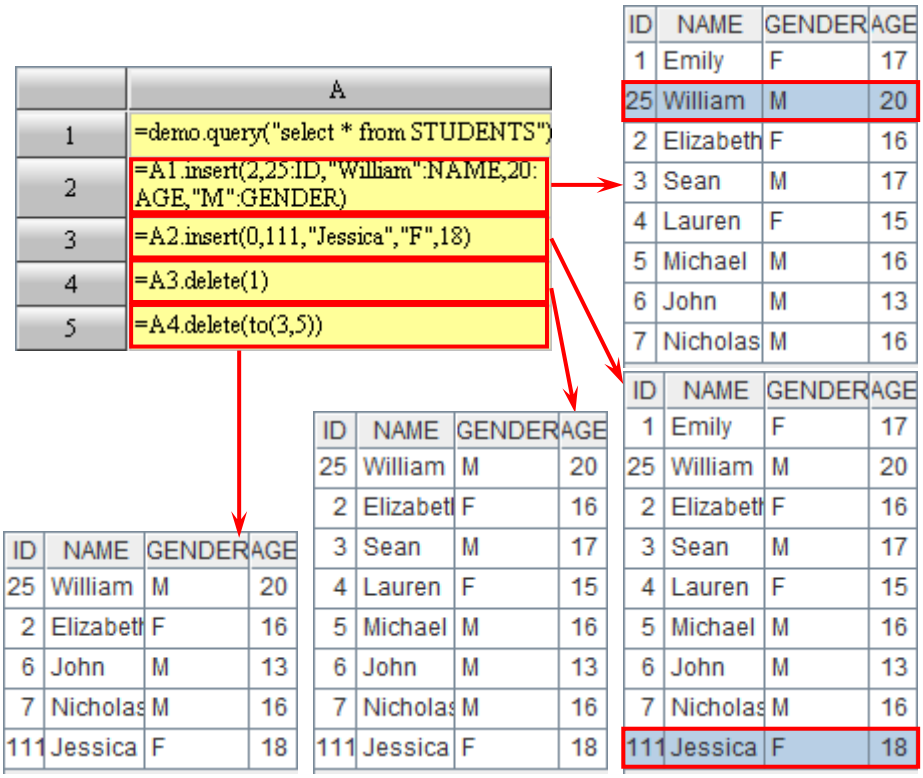
删除在序表 *T* 中序号在数列 *p* 中的记录。

➢ ***T*.delete(*A*)**

在序表 *T* 中删除排列 *A* 中的记录。

- ***T*.delete(3)** 删除第 3 条记录
- ***T*.delete(to(2,4))** 删除第 2 到 4 条记录

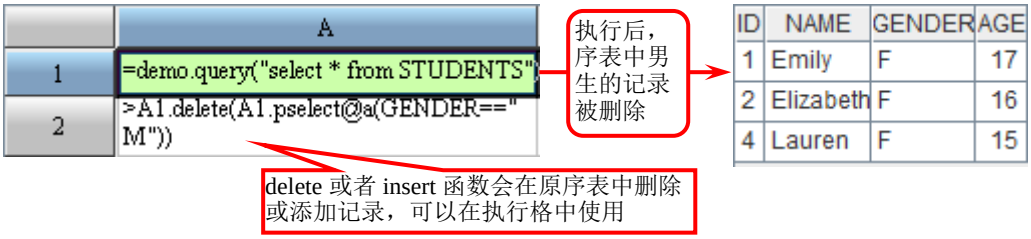
在下面的例子中，为了看到每一步中添加删除记录的执行结果，连续点击  按钮，使用调试功能中的分步执行，依次查看每一个单元格的执行结果：



对于序表只能且必须使用 **insert** 和 **delete** 函数增加和删除其中的成员，而不能像普通序列那样重新赋值改变。

集算器没有提供类似 **SQL** 的按条件删除功能，可以用 **pselete** 组合出来：

- ***T*.delete(*T*.pselect@a(Age>=30))**



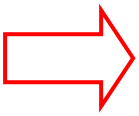
需要批量插入记录时，可以使用下面的方法：

- ***T*.insert(*k*:*A*,*x*$_i$:*F*$_i$,…)**

针对 *A* 的每个成员循环执行，在序表 *T* 的第 *k* 条记录前插入多条记录，字段 *F* 赋值为 *x*，*k* 为 0 表示在最后插入，*F* 省略表示依次赋值。*A* 可以用整数 *n* 表示序列 to(*n*)。

	A
1	=demo.query("select * from STUDENTS")
2	=A1.insert(2:3,25+#:ID,"William"+string(#):NAME,20:AGE,"M":GENDER)

ID	NAME	GENDER	AGE
1	Emily	F	17
2	Elizabeth	F	16
3	Sean	M	17
4	Lauren	F	15
5	Michael	M	16
6	John	M	13
7	Nicholas	M	16



ID	NAME	GENDER	AGE
1	Emily	F	17
26	William1	M	20
27	William2	M	20
28	William3	M	20
2	Elizabeth	F	16
3	Sean	M	17
4	Lauren	F	15
5	Michael	M	16
6	John	M	13
7	Nicholas	M	16

6.3.2 修改记录

- ***r*.modify(*x*$_i$:*F*$_i$,…)**

  修改记录 *r*，字段 *F*$_i$ 赋值为 *x*$_i$，*F* 省略则依次赋值。

- ***T*.modify(*k*,*x*$_i$:*F*$_i$,…)**

  ***T*(*k*).modify(*x*$_i$:*F*$_i$,...)**，修改 *T* 的第 *k* 条记录。

- ***T*.modify(1," Tyler":Name,28:Age,"M":Gender)**
- ***T*.modify(2," Alexis","F":Gender,25)**

我们仍然使用 **6.3.1 修改数据结构**中的 STUDENTS 学生表来看一下如何修改记录：

	A
1	=demo.query("select * from STUDENTS")
2	>A1.modify(1,11:ID,"Matthew":NAME,"M":GENDER)
3	>A1(4).modify(22,"Alyssa","F",18)

ID	NAME	GENDER	AGE
11	Matthew	M	17
2	Elizabeth	F	16
3	Sean	M	17
22	Alyssa	F	18
5	Michael	M	16
6	John	M	13
7	Nicholas	M	16

再次强调，不能对序表的成员进行替换性赋值，如 ***T*(2)=*r*** 是非法的。事实上，没有整体改变序表成员的办法，只能对其字段重新赋值。

- ***T*.modify(*k*:*A*,*x*$_i$:*F*$_i$,…)**

  针对 *A* 的每一个成员循环执行，批量修改 *T* 的第 *k* 条起的多条记录。*A* 可以用整数 *n* 表示序列 to(*n*)。

	A
1	=demo.query("select * from STUDENTS")
2	>A1.modify(2:3,ID+10:ID)

ID	NAME	GENDER	AGE
1	Emily	F	17
12	Elizabeth	F	16
13	Sean	M	17
14	Lauren	F	15
5	Michael	M	16
6	John	M	13
7	Nicholas	M	16

6.3.3 重置序表

➢ ***T*.reset()**

保留数据结构清空序表 *T*。

6.3.4 粘贴

➢ ***r*.paste(*r*')**

将记录 *r* 中的字段值按位置修改为记录 *r*'的字段值。

❖ **@n** 修改记录时，只修改 *r* 中与 *r*'同名的字段值。在判断字段是否同名时，对大小写字母敏感。

STUDENT

ID	NAME	GENDER	AGE
1	Emily	F	17
2	Elizabeth	F	16
3	Sean	M	17
4	Lauren	F	15
5	Michael	M	16
6	John	M	13
7	Nicholas	M	16

SELLERS

GROUPID	ID	NAME	GENDER
1	1	Ashley Williams	F
1	2	Andrew Miller	M
2	1	Sarah Johnson	F
2	2	Grace Johnson	F
2	3	Christina Williams	F

	A
1	=demo.query("select * from STUDENTS")
2	=demo.query("select * from SELLERS")
3	=A1(1).paste(A2(4))
4	=A1(3).paste@n(A2(4))

红框内字段值改变

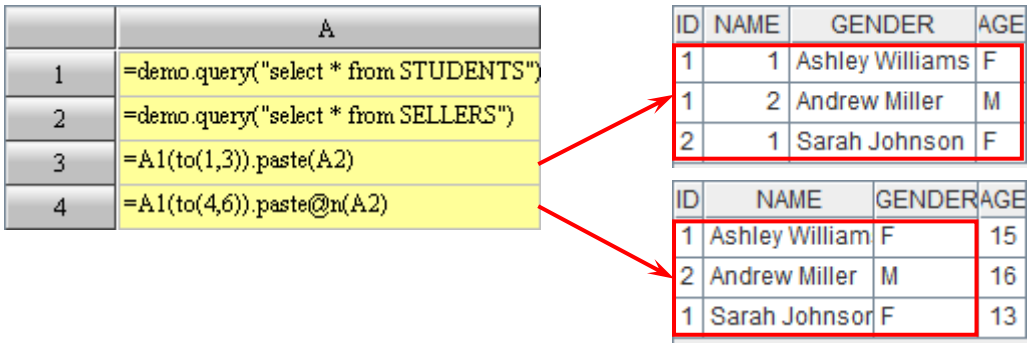
ID	NAME	GENDER	AGE
2	2	Grace Johnson	F

ID	NAME	GENDER	AGE
2	Grace Johnsor	F	17

➢ ***P*.paste(*A*)**

用序列 *A* 的成员，依次对排列 *P* 中的记录进行 paste。一般情况下，*A* 是排列，但是并不强制要求，*A* 的成员也可以是普通序列。

❖ **@n** 修改 *P* 中每一条记录时，使用**@n** 选项。



6.4 主键

6.4.1 主键的设定

在序表中，可以指定某个或某些字段作为主键，基于主键的查找有专门的函数以简化书写。原则上，序表**假定**主键值在记录中是**唯一**的，但并没有硬性检查，主键值重复了也**不会报错**。主键可以不进行设定，不设定时，按照第一个字段为主键处理。在用 *T*.create() 生成新序表时，*T* 的主键设定也同样会被复制。

- ***T*.primary(F_i,…)**

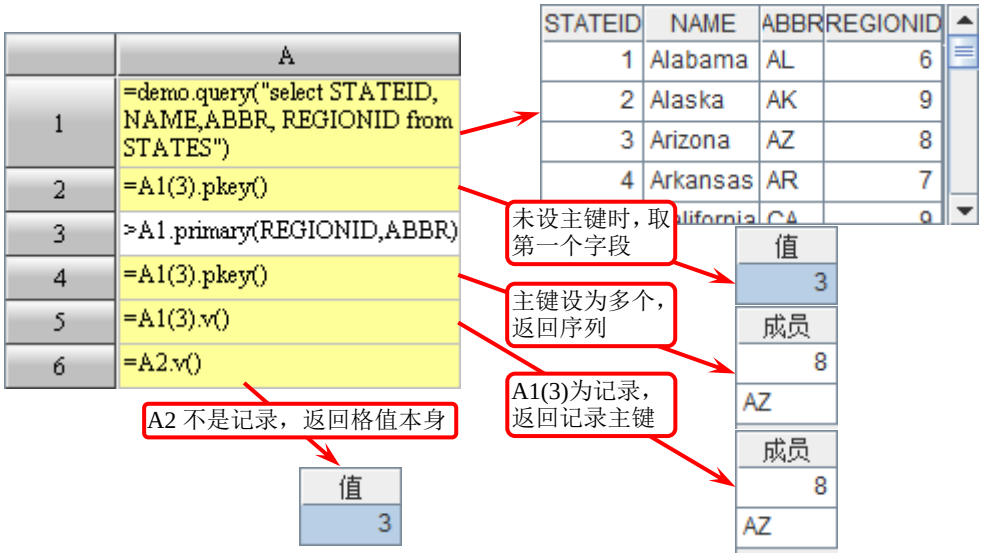
设置序表的主键为 F_i,…。使用 ***T*.create()**创建空序表时，会复制 *T* 的主键信息。

- ***r*.pkey()**

返回记录中所有主键字段成员组成的序列。当某个主键字段成员仍为记录时，则用该记录继续执行 **pkey()**的结果插入序列。当仅有一个主键时，只返回字段成员，不返回序列。

- ***v*.v()**

当 *v* 为记录时，返回记录主键字段值，或多主键字段值组成的序列；否则返回 *v* 本身。



- ***T*.index(*n*)**

为序表 *T* 的主键建立长度为 *n* 的索引表，*n* 省略时自动选择。如果需要多次根据主键来查找数据，在建立了索引表之后可以提高效率。

6.4.2 主键的使用

可以使用主键值来对应记录进行定位、选出等操作。

➢ ***A*.pfind(*v*)**

返回序列 *A* 中主键值为 *v* 的记录成员或者值为 *v* 的其他类型成员在序列中的位置。当主键有多个时，*v* 是序列。

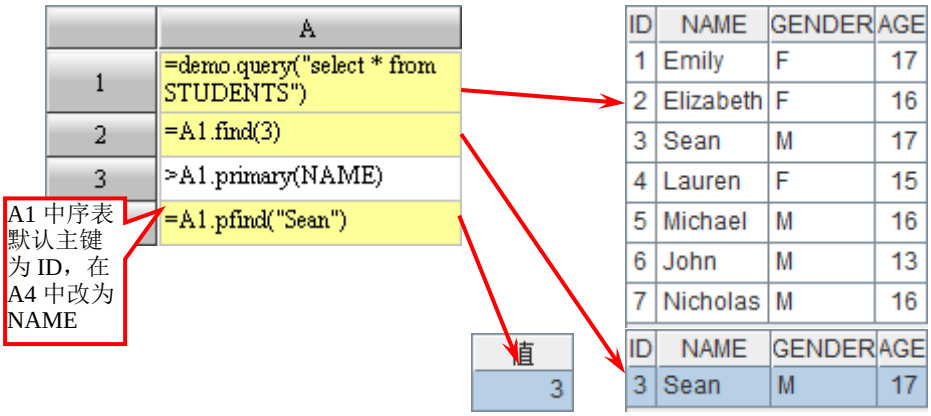
❖ **@b** 如果 *A* 中成员对其主键有序，查找时使用二分法。

❖ **@bs** 如果 *A* 中成员对其主键有序，在用二分法查找时，当能找到 *v* 时返回其位置；当 *v* 非 *A* 成员的主键值或者成员值时，返回 *v* 按顺序可插入位置序号的相反数。

➢ ***A*.find(*v*)**

返回序列 *A* 中主键值为 *v* 的记录成员或者值为 *v* 的非记录成员。当主键有多个时，*v* 是排列。如果 *A* 有索引表，则在查找时使用。

❖ **@b** 如果 *A* 中成员对其主键有序，查找时使用二分法，此时忽略 *A* 的索引表。



使用主键，还会影响一些函数的效果，如 **append@p**，具体情况将在 **7.1.3 追加记录**中说明。

6.4.3 记录转换与 switch 函数

使用 **switch** 函数，可以对记录 *r* 做变换，将 *r* 中一个或几个字段的值，转换为以其为主键的排列 *P* 中的记录引用。

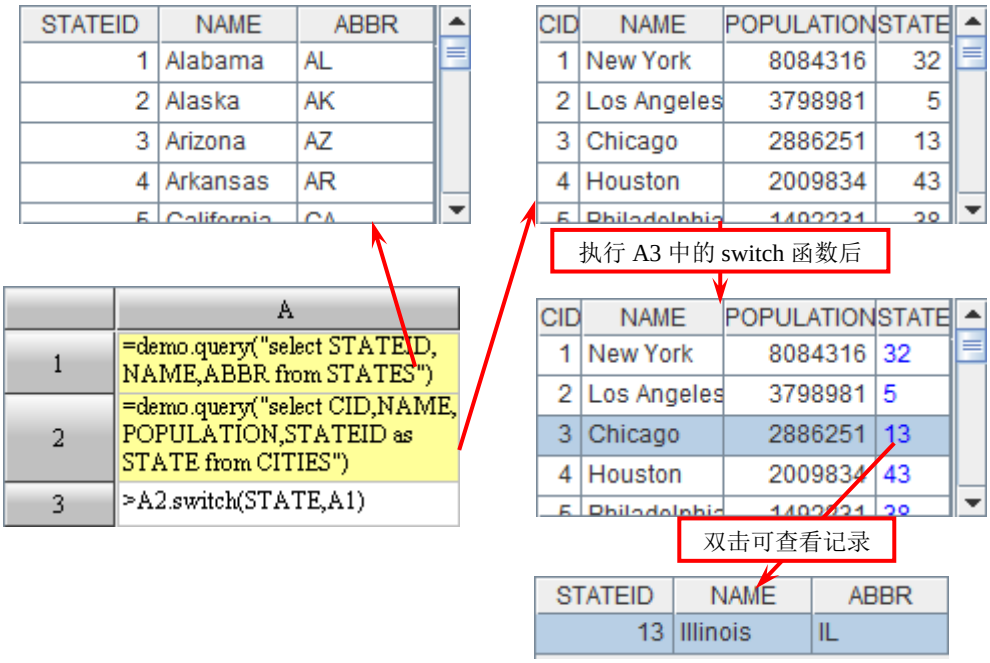
➢ ***P*.switch(F_i,A_i:*x*;…)**

将排列 *P* 中每一条记录的字段 F_i 赋值为 **A_i.select@1(*x*:F_i)**，即 A_i 中以 F_i 为主键的记录引用。如果 A_i 有索引表，则在查找时使用。当 *x* 省略时按照 A_i 的主键查找；当 A_i 也省略时，字段 F_i 赋值为 F_i.v()。特别地，*x* 为 **#** 时直接用序号定位。如果无法在 A_i 中找到相应的记录，则将相应字段赋值为 null。

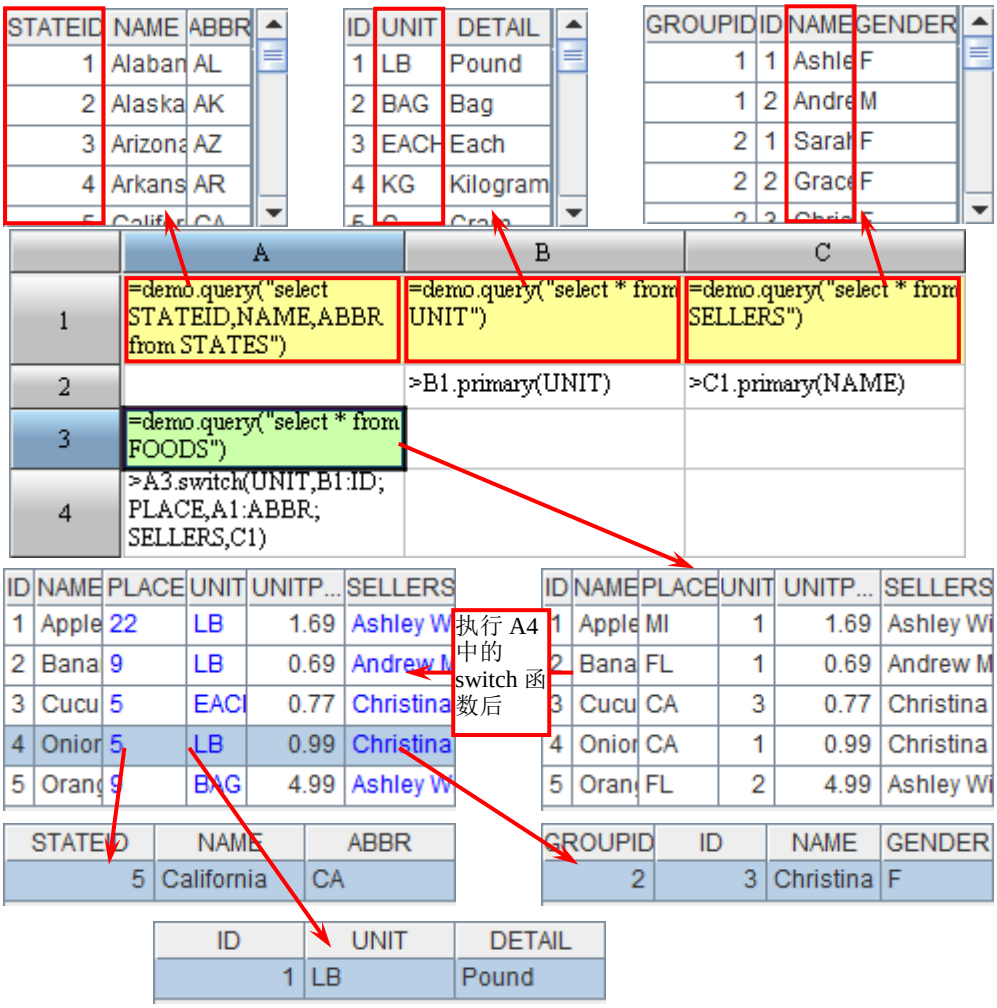
❖ **@i** 在计算中，如果 *P* 中的某条记录对每一个 A_i 全部无法找到 F_i 的对应值，则删除整条记录。

用 **switch** 函数可以更简单地完成记录引用赋值：

- ***A*.primary(编号)**
- ***B*.switch(销售员,*A*)**



使用 **switch** 函数，可以处理一些更为复杂的情况，比如同时变更多个字段的值：



7. 数据关联

7.1 数据产生

7.1.1 生成新序表

- ***A*.new(*x*$_i$:*F*$_i$,…)**

A 为序列或排列，先创建以 *F*$_i$,…为字段的序表，再循环 *A* 中的每个成员计算表达式 *x*$_i$ 生成新记录插入序表。

	A
1	=to(5).new(~:Value,~*~:Square,~*~*~:Cube)

Value	Square	Cube
1	1	1
2	4	8
3	9	27
4	16	64
5	25	125

用 ***A*.new()**函数，可以使用已有的排列数据来生成新的序表：

STATEID	NAME	POPULATION	ABBR	AREA	CAPITAL	REGIONID
1	Alabama	4779736	AL	52419	Montgomery	6
2	Alaska	710231	AK	663267	Juneau	9
3	Arizona	6392017	AZ	113998	Phoenix	8
4	Arkansas	2915918	AR	52897	Little Rock	7
5	California	37253956	CA	163700	Sacramento	9
6	Colorado	5029196	CO	104185	Denver	8

	A
1	=demo.query("select * from STATES")
2	=A1.sort(POPULATION:-1)
3	=A2(to(1,5)).new(ABBR:ID,NAME,POPULATION,AREA)

取出人口排名前 5 名的州的信息，生成新序表

ID	NAME	POPULATION	AREA
CA	California	37253956	163700
TX	Texas	25145561	268820
NY	New York	19378102	54556
FL	Florida	18801310	65755
IL	Illinois	12830632	54826

- ***A* regex(*rs*,*F*$_i$,…)**

对字符串序列 *A* 的每个成员执行 regex(*rs*)，将匹配结果拼成以 *F*$_i$,…为字段名的新序表。

- **@i** 大小写字母不敏感。
- **@u** 使用 unicode 匹配。

使用 ***A*.regex()**函数，可以将一个字符串序列拆分，生成一个新序表。如：

	A	B	C	D
1	Oliver Twist	Noah Claypole	Rose Maylie	Toby Crackit
2	=[A1:D1]			
3	=A2.regex("(\S*) (\S*)",FirstName,LastName)			

A3 中，将姓名序列拆分为名和姓，中间用空格分隔，A2 与 A3 中的计算结果如下：

Member
Oliver Twist
Noah Claypole
Rose Maylie
Toby Crackit

FirstName	LastName
Oliver	Twist
Noah	Claypole
Rose	Maylie
Toby	Crackit

当用正则表达式无法匹配某个字符串时，就无法生成记录，利用这种机制，我们可以过滤序列，选出指定的字符串用来生成序表，如：

	A
1	=demo.query("select NAME,STATE,SALARY from EMPLOYEE")
2	=A1.(~.string())
3	=A2.regex("(B\\S*),(.*),(\\d*)",Name,State,Salary)
4	=A2.regex("(.*),([C|T].*),(\\d*)",Name,State,Salary)

A2 中生成字符串序列如下：

Member
Rebecca,California,7000
Ashley,New York,11000
Rachel,New Mexico,9000
Emily,Texas,7000
Ashley,Texas,16000
Matthew,California,11000
Alexis,Illinois,9000

A3 根据字符串依次匹配员工姓名，所在州和工资，正则表达式中的点.表示一个非换行符非回车符的字符。匹配时，要求员工姓名以 B 开头，A3 中结果如下：

Name	State	Salary
Brandon	Pennsylvania	7000
Brandon	New York	10000
Brianna	Georgia	8000
Bryan	Illinois	12000
Brandon	Pennsylvania	6500
Benjamin	Florida	5000
Brooke	New York	12000

A4 仍然是依次匹配员工姓名，所在州和工资来生成序表，其中要求所在州以 C 或 T 开头，计算结果如下：

Name	State	Salary
Rebecca	California	7000
Emily	Texas	7000
Ashley	Texas	16000
Matthew	California	11000
Megan	California	11000
Victoria	Texas	3000
Jacob	Texas	12000

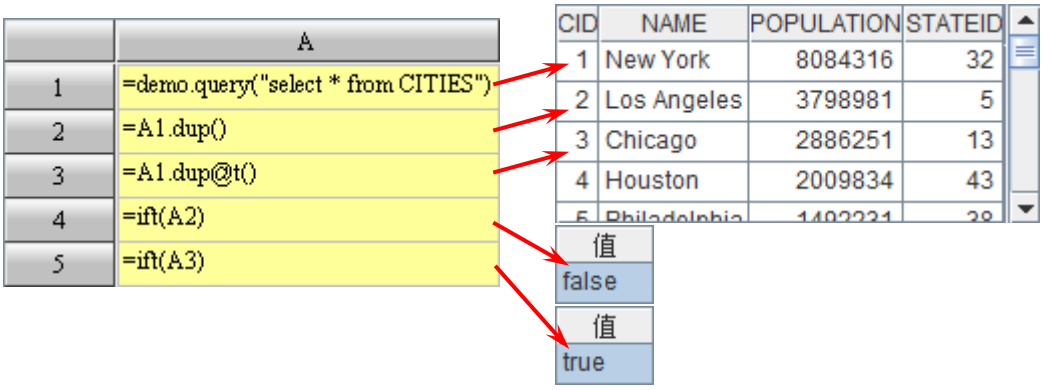
7.1.2 序表的复制

在 **3.5.1 使用函数生成产生列**中，我们讲到，复制序列可以使用 ***A*.dup()**函数，序表实际上也是一种序列，但是在复制序表时有一些需要注意的地方。

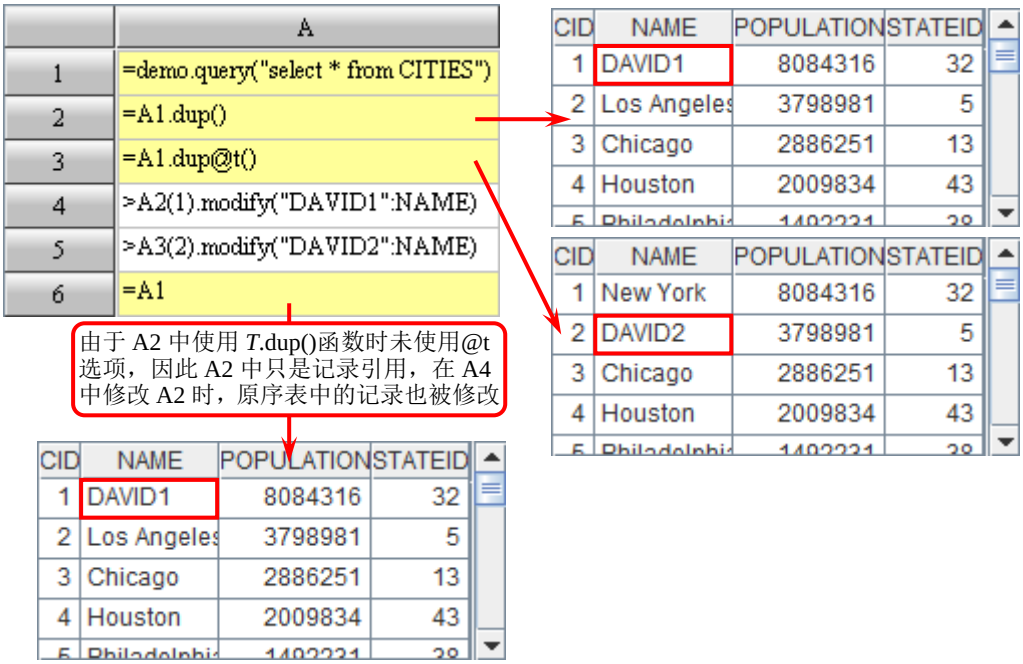
➢ ***T*.dup()**

把 *T* 作为序列来复制。注意，其结果是由 *T* 中记录引用所构成的排列，而不是序表。

❖ **@t** 在序表使用 **dup()**函数时，可以使用**@t** 选项，此时生成结果为序表。原序表中的记录会依次复制，在新序表中产生新记录。



需要注意的是，使用 ***T*.dup()**时，如果不使用**@t** 选项，在获得的新排列中的记录只是原序表的记录引用，记录本身未被复制。

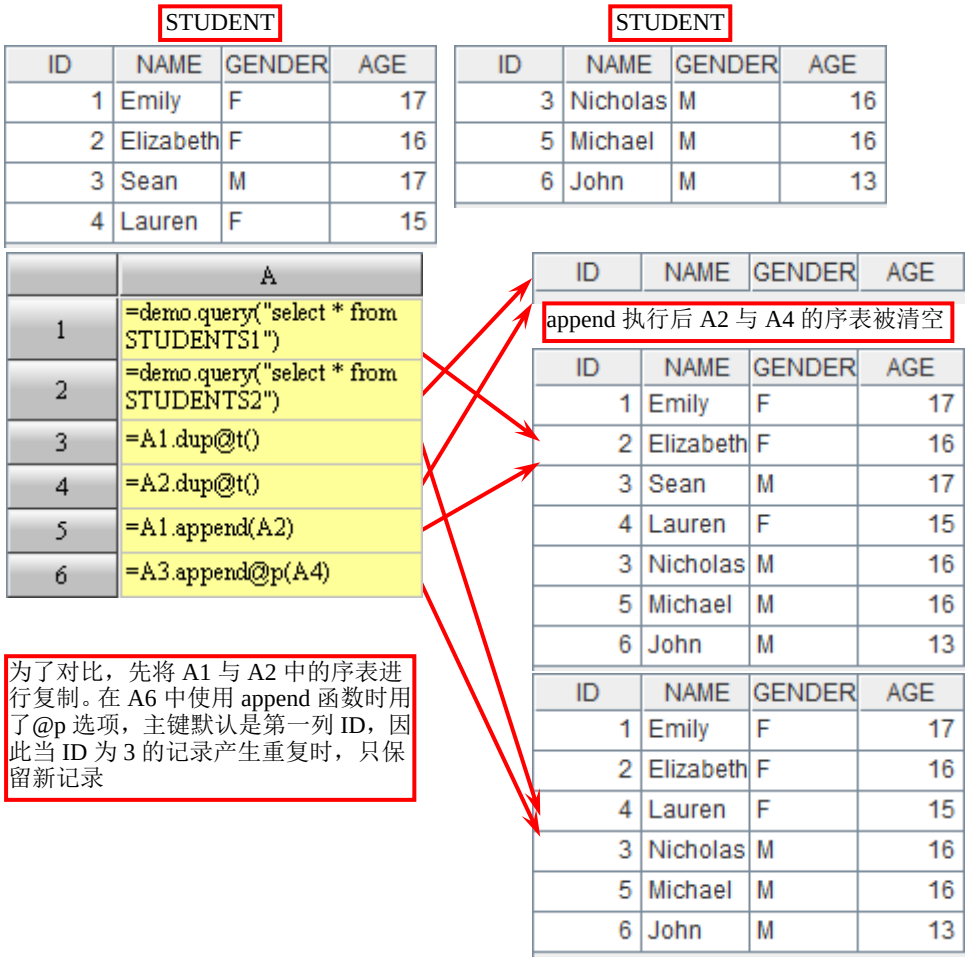


7.1.3 追加记录

➢ ***T*.append(T_i,…)**

T 与 T_i,…为字段数相同的序表，将 T_i,…中的记录追加入 *T* 中，同时清空序表 T_i,…中的记录。

❖ **@p** 使用**@p** 选项，在追加记录时，如果遇到主键相同的记录，则主键重复的原记录将被替换。



7.1.4 拆分记录

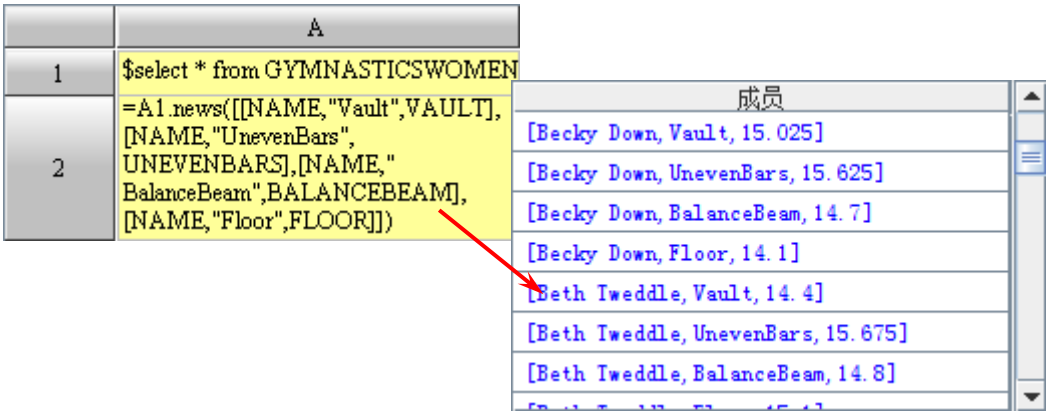
- ***A*.news(…)**

根据表达式…，对排列或序表 *A* 中的每条记录执行运算，将每一条记录拆分为序列或排列，并将所有序列中的成员，或者排列中的记录并在一起返回。最终返回的结果通常是序列或者排列。

例如，某次女子体操全能的成绩存储在名为 GYMNASTICSWOMEN 的数据库表中：

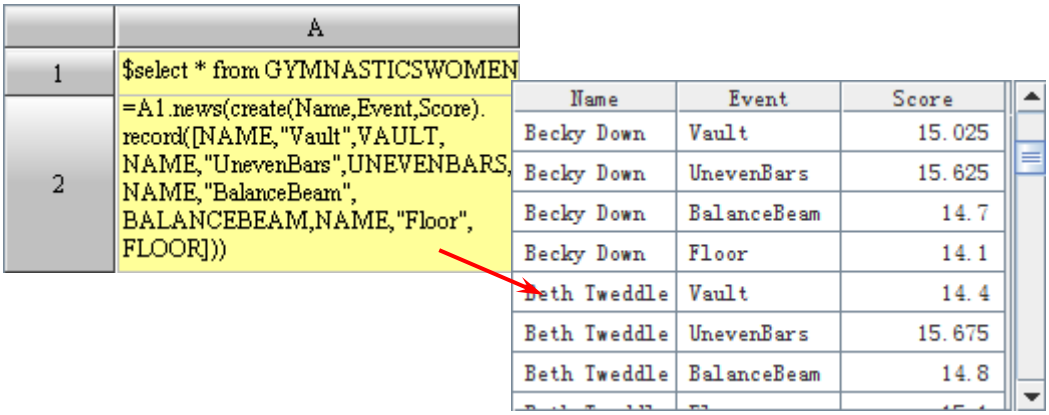
ID	NAME	COUNTRY	VAULT	UNEVENBARS	BALANCEBEAM	FLOOR
1	Ana Ailva	BRA	14.175	14.175	14.175	14.35
2	Anna Pavlor	RUS	15.275	14.525	15.975	15.05
3	Ariella Kask	SUI	15.35	14.275	14.425	13.95
4	Barbara Keesling	SUI	14.525	14.125	15.075	14.3
5	Becky Down	GBR	15.025	15.625	14.7	14.1
6	Beth Tweddle	GBR	14.4	15.675	14.8	15.1

下面需要得到每位运动员每项比赛的成绩：



可以看到，在 ***A*.news()**函数的参数中，表达式将原序表中的每条记录都拆分为一个序列，分别存储了一项比赛的成绩。最终返回的结果，是一个序列的序列，将所有运动员每项比赛的成绩列出。

为了更明晰地列出结果，也可以为每条记录生成一个序表：



此时，执行 ***A*.news()**函数时，会将各个序表中的记录合并在一起，返回排列。

7.2 数据的分组与汇总

7.2.1 计算唯一值

- ***A*.id(*x*)**

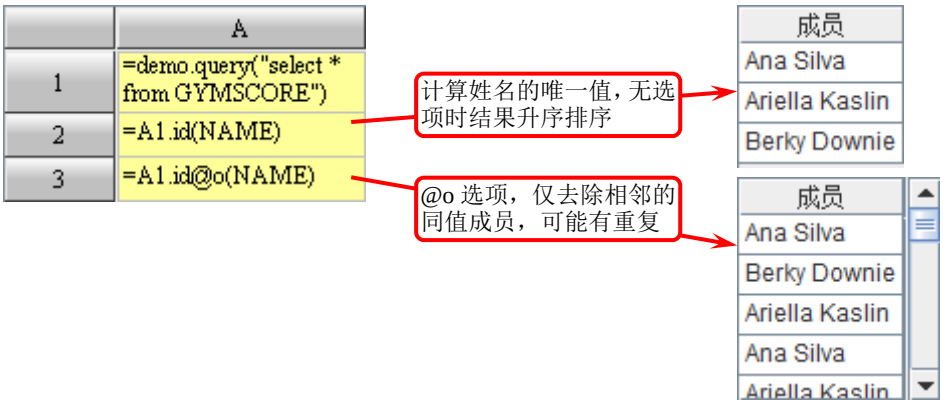
对序列 *A* 中的每个成员计算表达式 *x*，将结果构成的序列进行排序，去除相邻的同值成员后返回。当 *x* 省略时，将 *A* 的成员排序后去除相邻的同值成员后返回。

- **@o**　　不排序，直接去除相邻同值成员，结果可能包含同值成员。

某次体操比赛的成绩单如下：

ID	NAME	EVENT	SCORE
1	Ana Silva	Vault	14.175
2	Berky Downie	Vault	15.025
3	Ariella Kaslin	Vault	15.35
4	Ana Silva	UnevenBars	14.175
5	Ariella Kaslin	UnevenBars	14.275
6	Berky Downie	UnevenBars	15.625
7	Berky Downie	BalanceBeam	14.7
8	Ana Silva	BalanceBeam	14.175
9	Ariella Kaslin	BalanceBeam	14.425
10	Ariella Kaslin	Floor	13.95
11	Ana Silva	Floor	14.35
12	Berky Downie	Floor	14.1

我们来看一下 **id** 函数的执行情况：



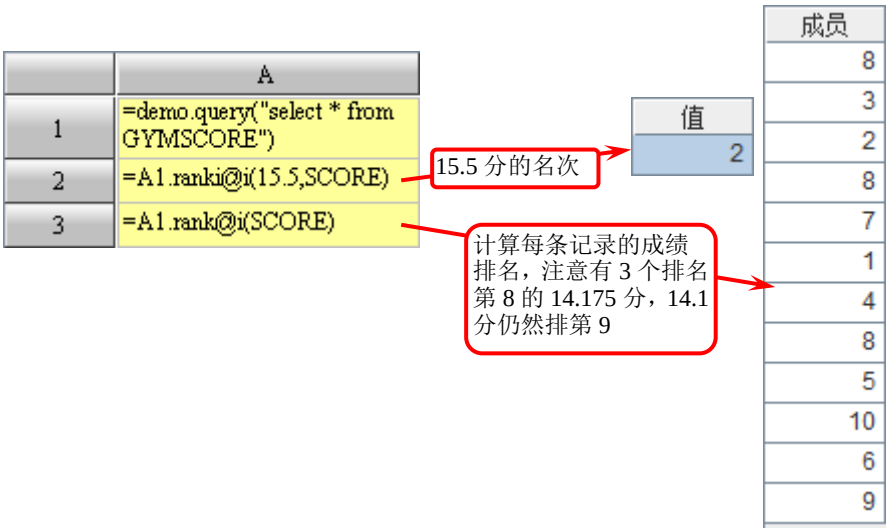
id 函数作用类似于 **SQL** 中的 **distinct** 运算。

- ***A*.ranki@i(*y*,*x*)**

对序列 *A* 中的每个成员计算表达式 *x*，去掉重复值后，*y* 的排名。

- ***A*.rank@i(*x*)**

对序列 *A* 中的每个成员计算表达式 *x* 的去重排名序列。



7.2.2 等值分组

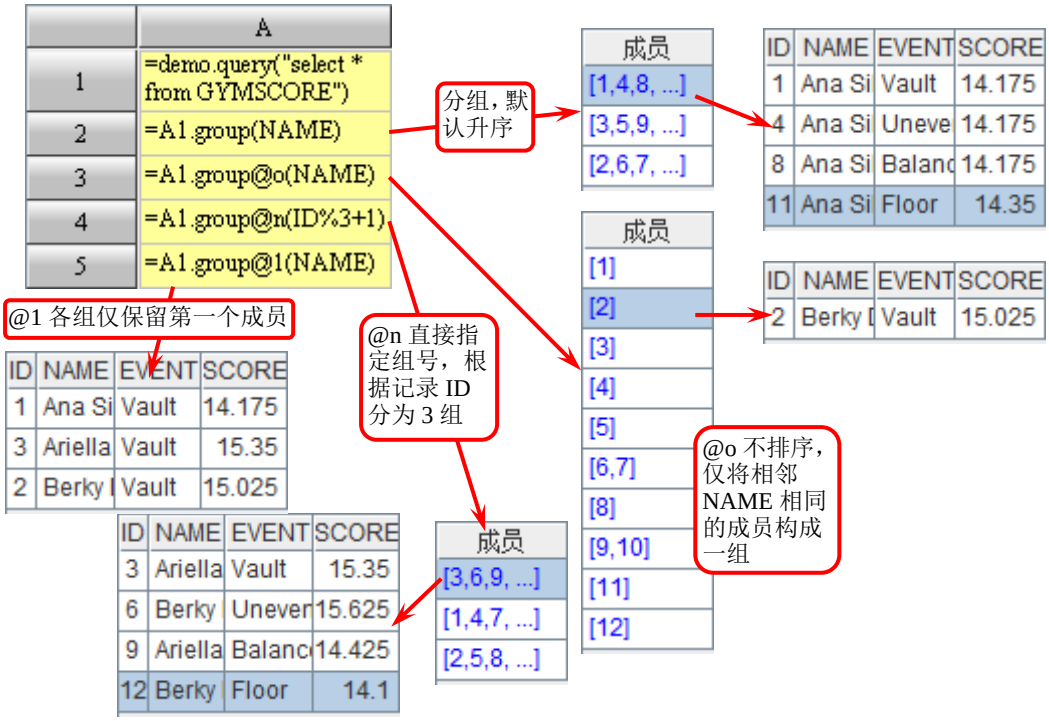
➢ ***A*.group(*x*ᵢ…)**

对序列 *A* 按表达式 *x*ᵢ…的计算结果升序分组，返回各组构成的序列。

❖ **@o** 不对 *x*ᵢ…值排序，只将相邻 *x*ᵢ…值相同成员并成一组。

❖ **@n** 表达式 *x*ᵢ…的计算结果为整数，作为分组序号直接用来定位，与**@o** 互斥。

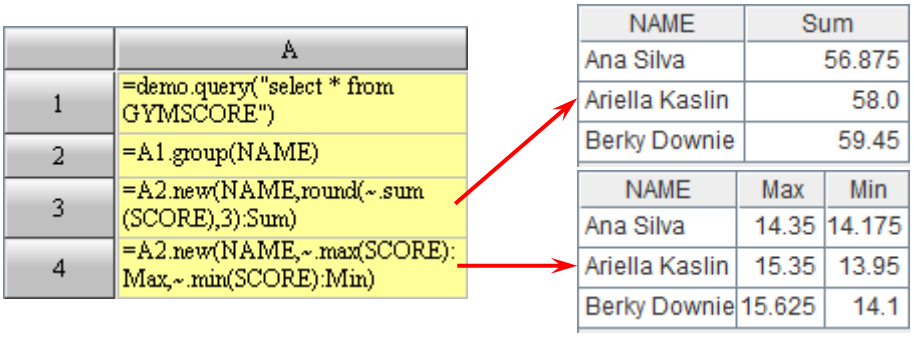
❖ **@1** 每一组中只保留第一个成员。



group 函数类似 **SQL** 中的 **group by** 运算。它的返回值是由各个分组构成的序列，每个分组则是原序列的产生列（或者是在原序列中的位置数列），最后结果是个由序列构成的序列。分组汇总值可基于分组结果继续多次计算。

这与 **SQL** 不同，**SQL** 中没有显式的集合数据类型，不能保存分组结果，必须在 **group by** 后立即计算出分组汇总值，运算后分组结果将被丢弃而不能重复利用。

在集算器中，分组后可以对分组的结果进行汇总：



	A
1	=demo.query("select * from GYMSCORE")
2	=A1.group(NAME)
3	=A2.new(NAME,round(~.sum(SCORE),3):Sum)
4	=A2.new(NAME,~.max(SCORE):Max,~.min(SCORE):Min)

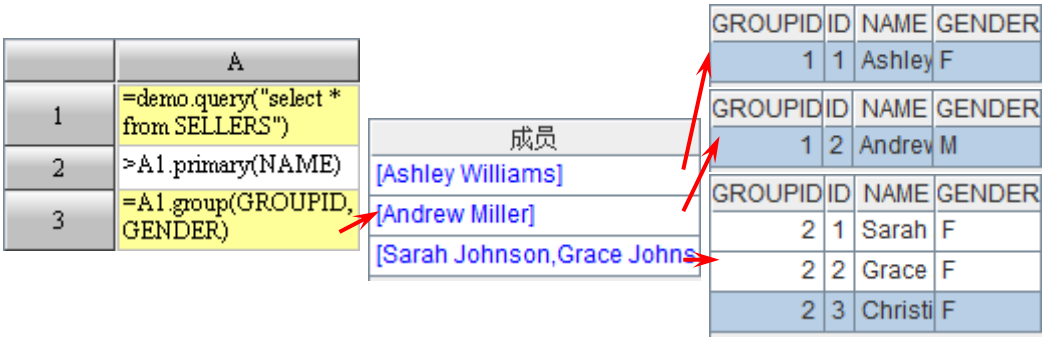
NAME	Sum
Ana Silva	56.875
Ariella Kaslin	58.0
Berky Downie	59.45

NAME	Max	Min
Ana Silva	14.35	14.175
Ariella Kaslin	15.35	13.95
Berky Downie	15.625	14.1

可以看到，在 **A3** 与 **A4** 中，对 **A2** 中的同一分组进行了不同的汇总计算。分组结果可以重复利用，这是集算器的主要特点之一。

再解说一下 A3 中的表达式：**=A2.new(NAME,round(~.sum(SCORE),3):Sum)**，由于 A2 是序表 A1 的分组结果，其每个成员都是一个记录的集合，即排列，因此 **new** 函数对 A2 进行循环时，其内部表达式都是对排列进行操作的。比如 **NAME** 表示取当前成员的第一条记录的姓名字段的值；**~.sum(SCORE)**表示对当前排列的成绩字段求和。

和 **SQL** 中的 **group by** 类似，**group** 函数也支持同时对多个字段（表达式）分组：



	A
1	=demo.query("select * from SELLERS")
2	>A1.primary(NAME)
3	=A1.group(GROUPID,GENDER)

成员
[Ashley Williams]
[Andrew Miller]
[Sarah Johnson,Grace Johns

GROUPID	ID	NAME	GENDER
1	1	Ashley	F

GROUPID	ID	NAME	GENDER
1	2	Andrev	M

GROUPID	ID	NAME	GENDER
2	1	Sarah	F
2	2	Grace	F
2	3	Christi	F

7.2.3 分组汇总

集算器也提供了直接计算出分组汇总值的函数：

- ***A*.groups(*x*:*F*,…;*y*:*G*,…)**

将序列 *A* 中的数据根据 *x*,…的计算结果进行分组汇总，结果聚合到结果集返回，相当于 ***A*.group(*x*,...).new(*x*:*F*,...,*y*:*G*,...)**。聚合时，不对 *A* 中的成员排序，而是依次循环每个成员计算，将用 *x*,…的结果按顺序匹配；聚合计算使用的表达式 *y* 只能用 **sum/count/max/min** 等简单聚合函数。计算完毕将返回新建序表，字段为 *F*,…,*G*,…，其中 *F*,…为主键字段。

- **@o**　　分组时使用@o 选项。
- **@n**　　*x* 的计算结果为分组的序号，聚合时直接定位。

	A
1	=file("D:/files/txt/employee.txt")
2	=A1.import@t()
3	=A2.groups(DEPT:Dept;count(~):Count,sum(SALARY):TotalSalary)

Dept	Count	TotalSalary
Administration	4	40000
Finance	24	177500
HR	19	138000
Marketing	99	733500
Production	91	663000
R&D	29	239000
Sales	187	1362500
Technology	47	344000

7.3 对齐与枚举分组

7.3.1 对齐

➢ ***P*.align(*A*:*x*,*y*)**

把序列 *P* 的成员按序列 *A* 的成员依次一一对应，*P* 中成员计算表达式 *y* 的结果，等于对应的 *A* 的成员计算表达式 *x* 的结果，*y* 省略则用 *P* 的成员值本身，*x* 省略则用 *A* 的成员值本身。默认情况下，只取 *P* 中第一个满足条件的成员。

在对齐时，与等值分组类似，经常使用以下几个选项进行不同的设定：

- **@a** 对齐分组，对齐时取出 *P* 中所有满足条件的成员，和 **group** 函数类似，**align@a** 函数的返回值也是个分组序列的序列。
- **@b** *A* 有序，查找时使用二分法提高效率。
- **@p** 返回由各组成员序号构成的序列。

	A
1	=demo.query("select * from GYMSCORE")
2	[BalanceBeam,UnevenBars,Floor,Vault]
3	=A1.align@a(A2,EVENT)
4	=A1.align@a(A2:~,EVENT)

A3 与 A4 计算结果是相同的

成员
[7,8,9]
[4,5,6]
[10,11,12]
[1,2,3]

ID	NAME	EVENT	SCORE
10	Ariella I	Floor	13.95
11	Ana Sil	Floor	14.35
12	Berky D	Floor	14.1

对齐分组后，可以通过序号把分组结果序列 **A3** 和 **A2** 关联起来进行统计，如：

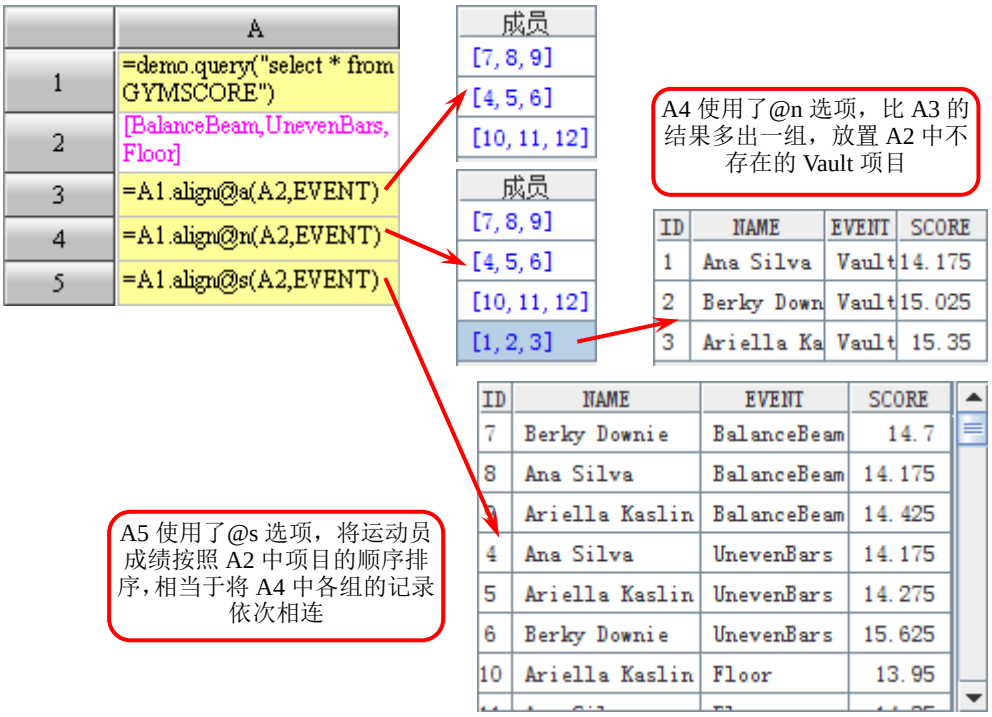
	A
1	=demo.query("select * from GYMSCORE")
2	[BalanceBeam,UnevenBars,Floor,Vault]
3	=A1.align@a(A2,EVENT)
4	=A1.align@a(A2:~,EVENT)
5	=A3.new(A2(#):Event,round(~.avg(SCORE),3):AvgScore)

Event	AvgScore
BalanceBeam	14.433
UnevenBars	14.692
Floor	14.133
Vault	14.85

这里解说一下表达式**=A3.new(A2(#):Event,round(~.avg(SCORE),3):AvgScore)**，表示对着 A3 循环计算，对 A3 的每一个成员计算生成新序表的对应记录。由于 A3 和 A2 的顺序是一样的，因此成员序号完全一致，所以通过**A2(#)**可以获得 A2 中同一位置上的成员，这里#是 A3 的当前成员序号。相当于**=A3.new(EVENT:Event,round(~.avg(SCORE),3):AvgScore)**。

❖ **@n** 计算时使用**@a** 选项，生成的结果最后多出一组，用来放置 *P* 中无法与 *A* 对应的成员。

❖ **@s** 将序列 *P* 的成员按 *A* 的顺序排序，其中无法与 *A* 对应的成员排在最后。



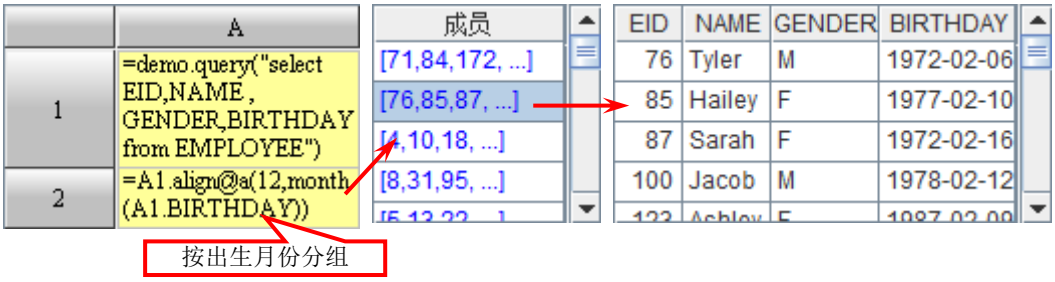
更简单的直接序号对齐：

➢ ***P*.align(*n*,*y*)**

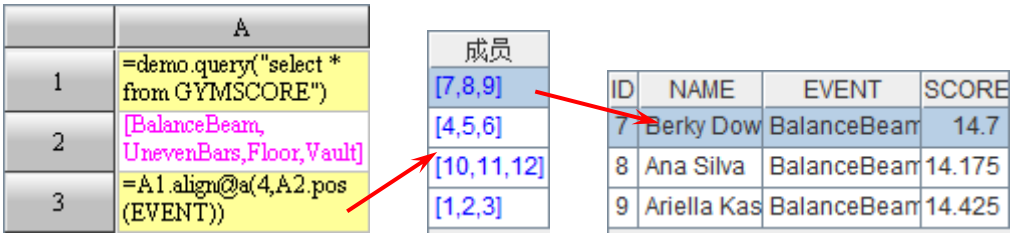
相当于 ***P*.align(to(*n*),*y*)**，即此时 *y* 值已经是对齐序号了。

使用序号对齐时，需要分组字段是连续整数，此时可以用 ***P*.align(*n*,*y*)**的方式来提高性能。使用序号对齐也可以使用**@a**、**@p** 选项。

❖ **@r** *y* 的计算结果不是序号，而是序号数列。此时 *P* 中的一条记录可以重复出现。



使用序号对齐时，也可以利用定位函数来生成连续整数。



7.3.2 枚举分组

➢ **eval(*x*,*a*)**

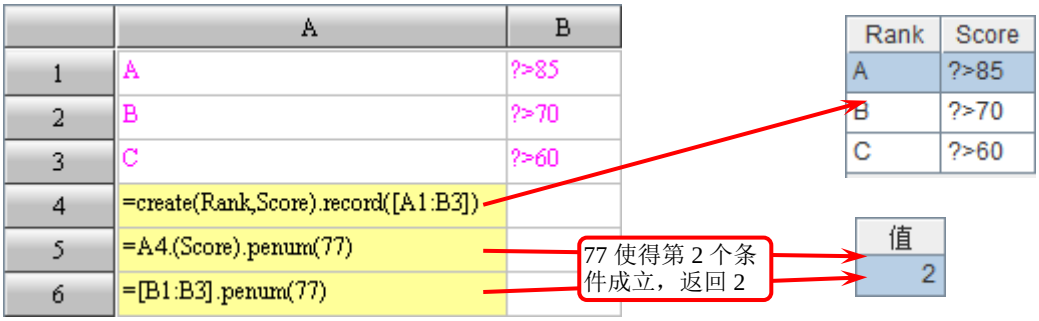
用参数 *a* 计算字符串 *x* 中的表达式，在 *x* 中用第 *i* 个?表示第 *i* 个参数，或者直接写明参数的位置?*i*，当参数少于?个数时，从第一个起循环

- **eval("?+3",5)** 返回 **8**
- **eval("?>3",5)** 返回 **true**
- **eval("?+3*?",5,7)** 返回 **5+3*7** 的结果，即 **26**。
- **eval("?+3*?+?",5,7)** 返回 **5+3*7+5** 的结果，即 **31**。

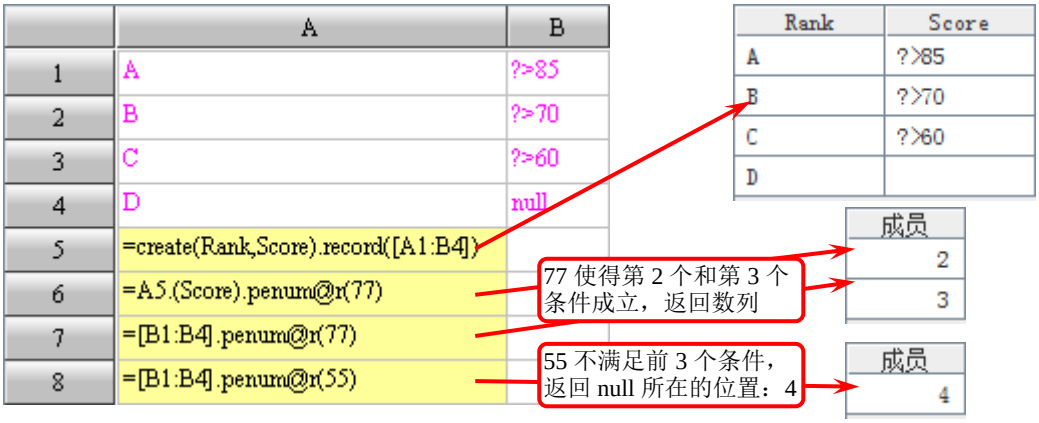
特别地，若 **eval(*x*,*a*)**返回 true 时，则称 *a* 满足条件 *x*。

➢ ***E*.penum(*y*)**

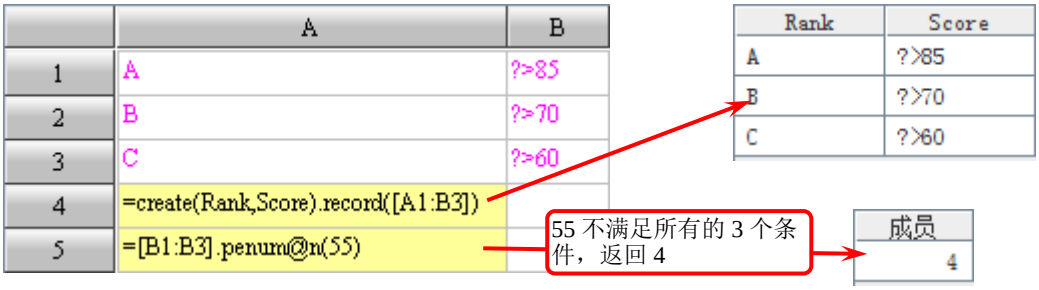
E 是由条件表达式构成的序列，计算表达式 *y*，看结果满足 *E* 中的哪个成员构成的条件，返回第一个满足条件的序号。特别的，当条件为 null 时视为满足条件。



❖ **@r** 条件可重复，返回满足的所有条件序号数列；只有其它条件均不满足时，才返回 null 条件的序号数列。



❖ **@n** 当所有条件均不满足时，返回 ***E*.len()+1**，与**@r** 选项互斥，此时不需要 null 条件。

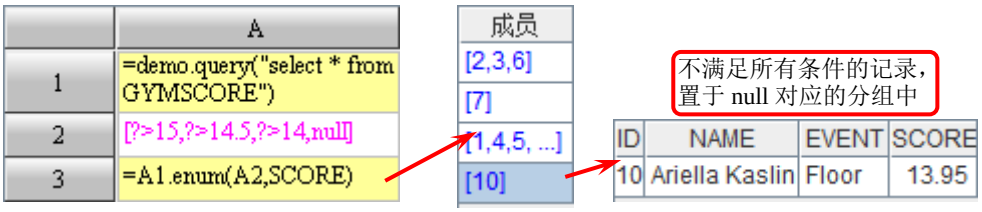


➢ ***P*.enum(*E*,*y*)**

把序列 *P* 的成员分成与条件表达式序列 *E* 的成员一一对应的组，每组成员计算表达式 *y* 的结果可满足对应的 *E* 的成员构成的条件，*y* 省略则用 *P* 成员本身的值。默认情况下，*P* 中的每个成员只会置入第一个满足条件的组中。

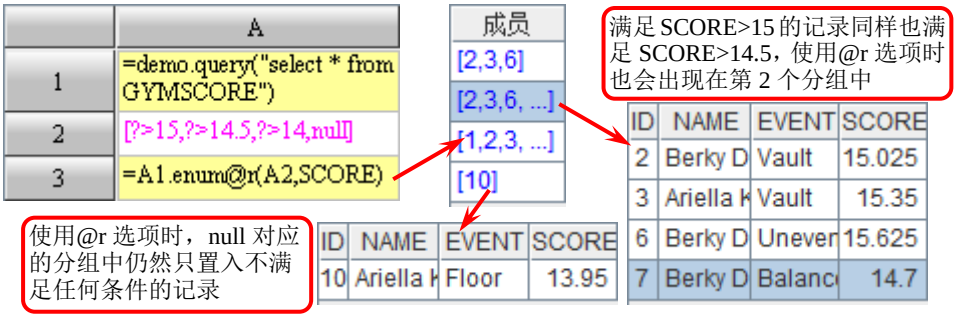
同样，**enum** 函数的返回值也是一个序列的序列。

当 ***E*.m(-1)==null** 时，由于条件为 null 时认为条件满足，**enum** 函数将会把 *P* 中所有不满足 *E* 中其它任何条件字段的成员收集起来归到与 ***E*.m(-1)**对应的分组中，相当于“其它”分组。



缺省情况下，**enum** 函数在计算时假定不会有重复分组，即 *P* 中成员不会同时满足两个条件表达式。如果需要重复分组时，需要使用**@r** 选项。

❖ **@r** 允许重复分组，即 *P* 中成员可能同时对应到多个 *E* 的成员。

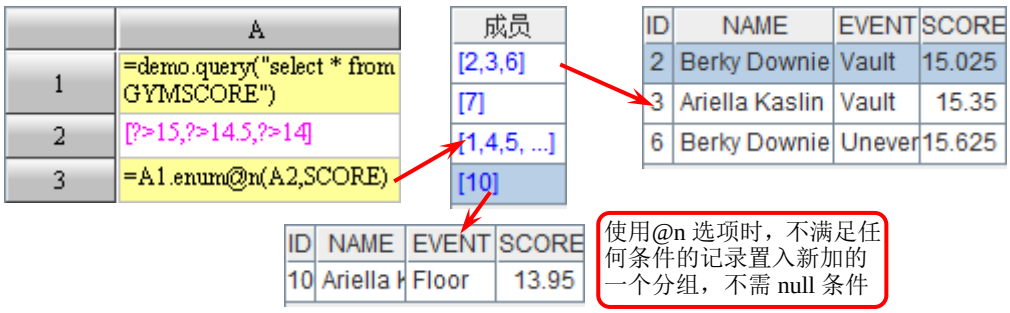


enum 函数还经常使用下面的选项进行不同的分组：

❖ **@p** 返回由各组成员序号数列构成的序列。

❖ **@n** 将 *P* 中所有不满足任何条件的成员，置于一个新增的分组中，与**@r** 选项

互斥，此时不需要 null 条件。



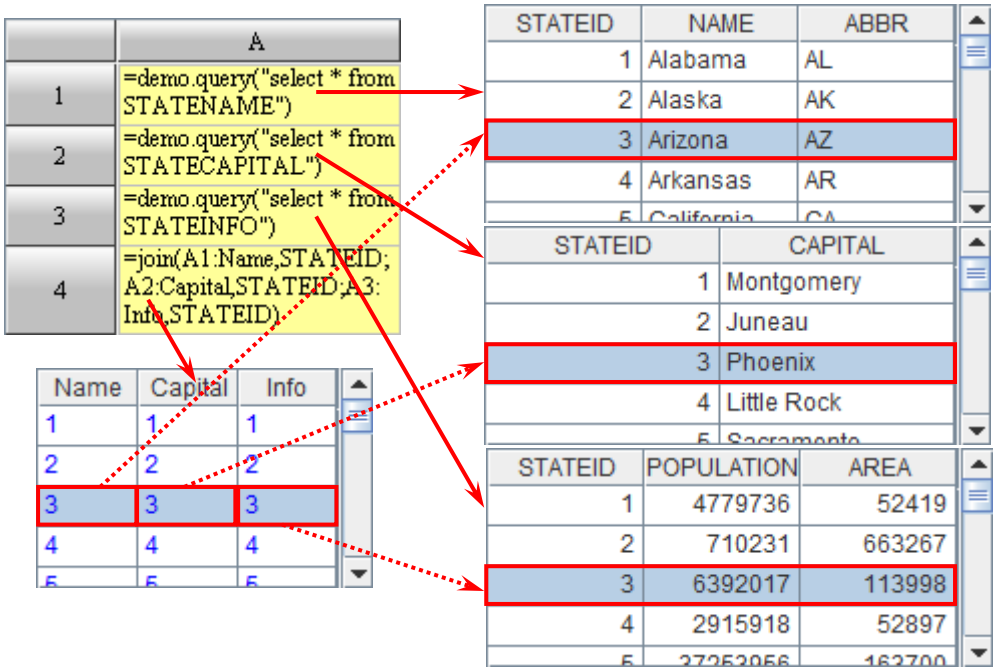
7.4 连接

集算器也提供类似 **SQL** 的连接运算，但在有了记录和排列引用机制和 **align** 等函数后，这种运算变得不大常用了。

7.4.1 等值连接

➢ **join(A_i:F_i,x_j,…;…)**

以对应的字段（表达式）x_j,…值相同为条件连接序列 A_i,…，产生以 F_i,…为字段的序表保存结果，F_i 取值为 A_i 的成员。



join 函数大体相当于在 **SQL** 中写：

– **select A_i.* as F_i,… from A_i,… where A_i.x_j =… and …**

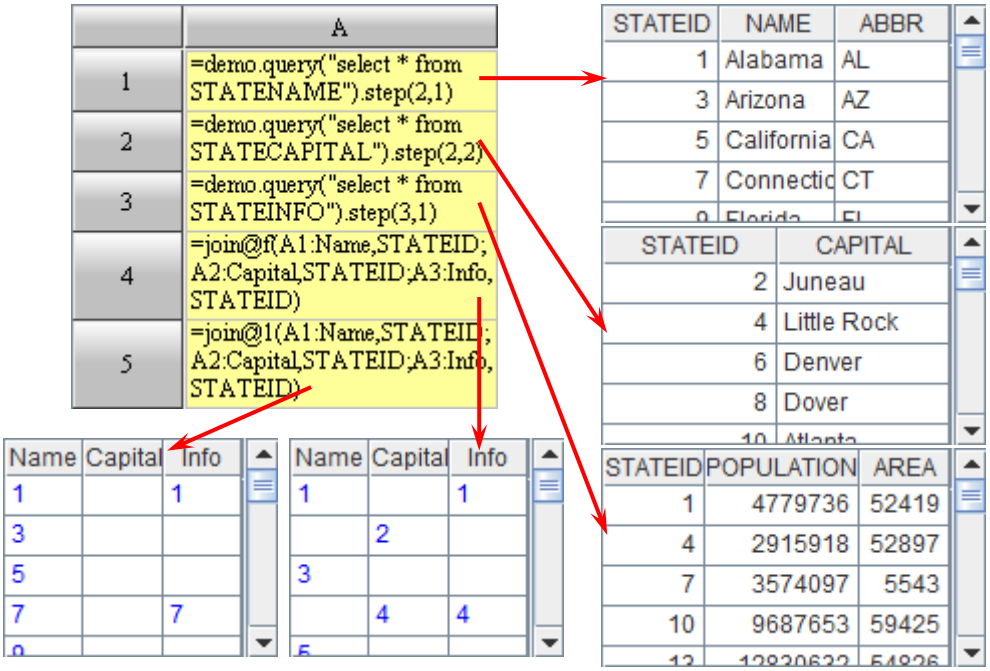
这也是 **SQL** 中最常见的连接写法。

join 函数经常会用到下面三个选项：

❖ **@f** 全连接，A_i 所有成员至少出现一次，无值对应补以 null。

❖ **@1** 左连接，A_1 所有成员至少出现一次，无值对应补以 null。注意选项是数字 1，再次强调，集算器中字母 l 不会作为选项出现。

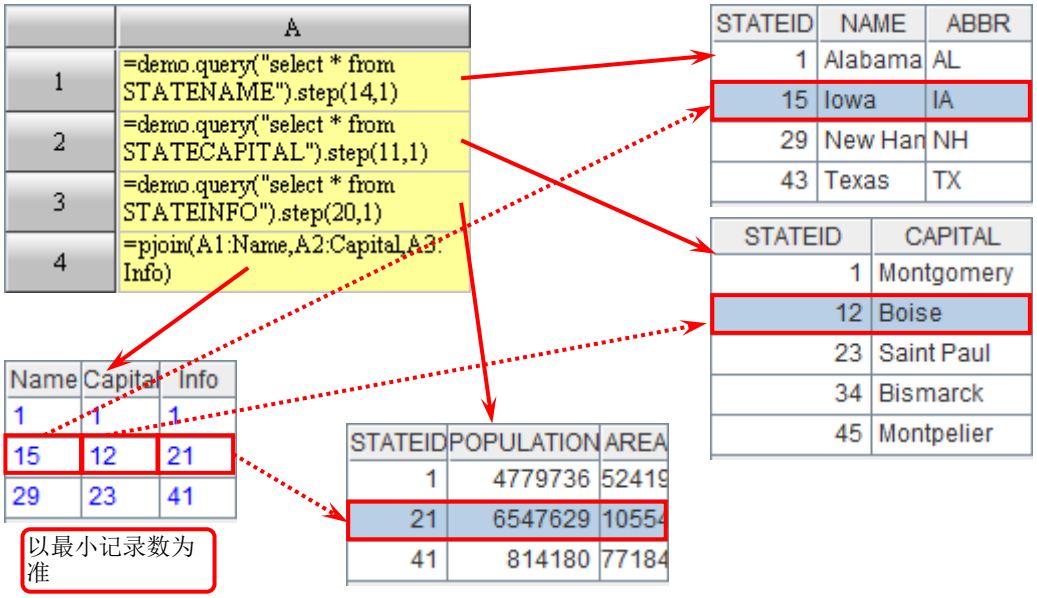
❖ **@m** 当所有 A_i 针对 x_i 均有序时，连接时使用归并算法以提高效率。



7.4.2 同位连接

➢ **pjoin(x_i:F_i,…)**

其中 x_i 为序列；产生以 F_i,…为字段的序表，第 k 条记录的 F_i 赋值为 x_i (k)，即将 x_i,…的成员按顺序依次连接，长度以最小的 x_i.len()为准，返回序表。



同位连接排列时，按记录顺序进行连接，字段里同样存储原表的记录。

7.5 数据关联的应用

数据关联的应用十分广泛。在处理数据分析、数据准备等工作时，都会遇到分组、汇

总、连接等操作。

7.5.1 按要求分组统计酒水库存

数据库中 LIQUORS 表中存储了各种酒的信息，包括酒的名称、种类、产地、库存等等：

LID	NAME	TYPE	PRODUCTION	STOCK
1	42Below Vodka	Vodka	New Zealand	301
2	Absolut Vodka	Vodka	Sweden	95
3	Appleton Estate Reserve	Rum	Jamaica	202
4	Bacardi Superior	Rum	Puerto Rico	741
5	Baileys Irish Cream	Cordials	Ireland	434

1) 统计各类酒各有多少种库存商品，以及各类酒的总库存量。

统计各类酒的库存信息，只需要按照种类 TYPE 字段进行分组，再分别统计即可：

	A
1	=demo.query("select * from LIQUORS")
2	=A1.group(TYPE)
3	=A2.new(TYPE,~.count():Count,~.sum(STOCK):Stock)

在 A1 中取出酒水信息；在 A2 中，根据种类进行分组；在 A3 中根据 A2 中的分组数据计算出结果序表。

网格计算后，在 A2 中可以看到 A1 中记录的分组情况：

成员
[11,16,21, ...]
[12,22,30, ...]
[5]
[14,24,32, ...]
[9,10,17, ...]

双击可以查看分组中记录的详细情况

LID	NAME	TYPE	PRODUCTION	STOCK
12	X O. Hennessy X. O. Cognac	Brandy	France	168
22	Martell Medaillon	Brandy	France	88
30	Martell Medaillon	Brandy	France	216
37	Club De Remy Martin	Brandy	France	29
44	XO. Courvoisier X.O.	Brandy	France	78

A3 中的结果序表如下：

TYPE	Count	Stock
Aperitif	6	800
Brandy	11	1941
Cordials	1	434
Gin	9	2410
Liqueur	13	2369

2) 统计朗姆酒、白兰地、金酒和威士忌这几类酒各有多少种库存商品，以及总库存量。

从上一项统计结果中可以看到，在直接使用 group 函数对种类 TYPE 字段进行分组时，是先按照 TYPE 字段进行排序的。当需要进行指定顺序的分组时，则经常使用对齐分组函数 **align@a**。

	A
1	=demo.query("select * from LIQUORS")
2	[Rum,Brandy,Gin,Whisky]
3	=A1.align@a(A2,TYPE)
4	=A3.new(TYPE,~.count():Count,~.sum(STOCK):Stock)

在 A3 中，将 A1 中酒的信息记录，按照种类 TYPE 字段分别对应序列 A2 中的成员值对齐分组，最后再 A4 中计算出结果序表如下：

TYPE	Count	Stock
Rum	6	2077
Brandy	11	1941
Gin	9	2410
Whisky	13	3813

3) 统计库存过多（>500）、库存不足（<100）以及库存正常（其余）的酒各有多少种库存，以及各组的总库存量。

当需要按照条件进行分组时，可以使用枚举分组。

	A	B
1	=demo.query("select * from LIQUORS")	
2	?>500	OverStock
3	?<100	LackStock
4	null	NormalStock
5	=A1.enum([A2:A4],STOCK)	
6	=A5.new([B2:B4](#):Condition,~.count():Count,~.sum(STOCK):Stock)	

在 A5 中根据分组条件，将 A1 中的记录按照库存 STOCK 进行分组；A6 中根据 A5 中的分组结果及各个条件对应的状态名称计算出结果序表如下：

Condition	Count	Stock
OverStock	2	1262
LackStock	16	927
NormalStock	50	14211

7.5.2 销售报表数据准备

数据库中的 SALES 表中存储了一段时间内的销售信息，表中包括订单号 ORDERID、客户名称 CLIENT、销售员编号 SELLERID、订单金额 AMOUNT 以及订单日期 ORDERDADE 等信息：

ORDERID	CLIENT	SELLERID	AMOUNT	ORDERDATE
1	UJRNP	17	392.0	11-02-2008 15:
2	SJCH	6	4802.0	11-09-2008 15:
3	UJRNP	16	13500.0	11-05-2008 15:
4	PWQ	9	26100.0	11-08-2008 15:
5	PWQ	11	4410.0	11-12-2008 15:
6	HANAR	18	6174.0	11-07-2008 15:
7	ECU	2	17800.0	11-06-2008 15:

现在，需要以年作为参数，取出这一年和前一年两年的销售数据，准备目标表。在目标表中，列出销售员编号、本年销售额、上一年销售额、增长率、客户数、大客户（销售额超过 5 万元的客户数），大客户占比。

由于需要计算每个销售员的数据，因此在把两年中的销售数据取出后，需要先按照销售员号码分组，再统计每一年的销售数据，并且需要按客户再次分组后统计各个客户的销售数据。

	A	B
1	=demo.query("select * from SALES where year(ORDERDATE) in (?)",[year,year-1])	
2	=A1.group(SELLERID)	
3	=create(SellerId,${string(year)+"Sales"},${string(year-1)+"Sales"},CustCount,BigCustCount,BigCustRate, GrowthRate)	
4	for A2	=A4(1).SELLERID
5		=round(A4.select(year(ORDERDATE)==year).sum(AMOUNT),2)
6		=round(A4.select(year(ORDERDATE)==year-1).sum(AMOUNT),2)
7		=A4.group(CLIENT).(round(~.sum(AMOUNT),2))
8		=B7.count()
9		=B7.count(~>=50000)
10		=round(B9/B8,2)
11		=round((B5-B6)/B6,2)
12		>A3.insert(0,B4,B5,B6,B8,B9,B10,B11)

在 **A1** 中，以 year 作为参数，取出该年以及前一年的销售数据；在 **A2** 中，将取出数据

按照销售员编号进行分组，在 **A3** 中，建立结果序表。第 4 行之后的代码，循环 **A2** 中每位销售员的销售数据，来进行统计。**B4** 中获取销售员编号；**B5** 中计算 2010 年总销售额；**B6** 中计算前一年 2009 年的总销售额。**B7** 中将销售数据按照客户再次分组，并计算出每个客户的订单总额。根据 **B7** 中的计算结果，**B8** 中计算出该销售员的客户总数；**B9** 中计算出其中大客户的数目。**B10** 中计算出大客户占比；**B11** 中计算出 2010 年该销售员的销售额增长率。计算完成后，在 **B12** 中，将该销售员的统计数据，填入 **A3** 的结果序表中。

计算完成后，在 **A3** 中可以查看数据准备的结果：

SellerId	2010Sales	2009Sales	CustCount	BigCustCount	BigCustRate	GrowthRate
1	265934.0	313228.0	22	1	0.05	-0.15
2	301502.0	298006.0	22	3	0.14	0.01
3	279986.0	142054.0	17	2	0.12	0.97
4	228070.0	237614.0	19	3	0.16	-0.04
5	175478.0	252294.0	19	2	0.11	-0.3
6	198150.0	246882.0	19	2	0.11	-0.2
7	188100.0	204208.0	16	2	0.13	0.08

7.5.3 员工薪酬计算

数据库中的员工表 EMPLOYEE 中存储员工信息，如姓名，所在州，性别，生日，基本工资等等：

EID	NAME	SURNAME	GENDER	STATE	BIRTHDAY	HIREDATE	DEPT	SALARY
1	Rebecca	Moore	F	California	1974-11-20	2005-03-11	R&D	7000
2	Ashley	Wilson	F	New York	1980-07-19	2008-03-16	Finan	11000
3	Rachel	Johnson	F	New Mexico	1970-12-17	2010-12-01	Sales	9000
4	Emily	Smith	F	Texas	1985-03-07	2006-08-15	HR	7000
5	Ashley	Smith	F	Texas	1975-05-13	2004-07-30	R&D	16000

在考勤表 ATTENDANCE 中，存储了员工当月的缺勤天数：

EMPLOYEEID	ABSENCE
1	1.00
3	2.00
6	1.50
9	3.00

在绩效表 PERFORMANCE 中，存储了员工当月的评估分数以及奖金数：

EMPLOYEEID	EVALUATION	BONUS
1	0.75	6000
2	0.90	10000
3	1.10	8000
4	0.80	800
5	1.40	3000
6	1.18	4000

- **应发工资=标准工资*(1-缺勤天数/(250/12))+标准工资*(绩效分-1)+奖金**

考勤表中缺少的人认为全勤，绩效表中缺少的员工认为绩效为 1，奖金为 0。找出应发工资与标准工资差距最大的前 5 名员工的姓名及其应发工资数。

由于考勤表与绩效表中只有部分员工的记录，因此无法直接用序号进行计算，需要先

按照员工编号对考勤表与绩效表中的记录进行对齐分组或者等值连接，然后再进行统计。

方法一：不用 join：

	A	B	C
1	=demo.query("select * from EMPLOYEE")		
2	=demo.query("select EMPLOYEEID, ABSENCE from ATTENDANCE")		
3	=demo.query("select EMPLOYEEID, EVALUATION,BONUS from PERFORMANCE")		
4	=A2.align(A1:EID,EMPLOYEEID)		
5	=A3.align(A1:EID,EMPLOYEEID)		
6			
7	for A1	=A7.SALARY	=B7
8		if A4(#A7)!=null	>C7=B7*(1-A4(#A7).ABSENCE/(250/12))
9		if A5(#A7)!=null	>C7=C7+B7*(A5(#A7).EVALUATION-1)+A5(#A7).BONUS
10		>A6=A6|round(C7-B7,2)	
11	=A6.psort(abs(~):-1)(to(1,5))		
12	=A1(A11).new(NAME+" "+SURNAME:FullName,A6(A11(#))+SALARY:Salary)		

在 A1、A2、A3 中，从数据库中取出员工信息、考勤信息和绩效信息。在 A4 和 A5 中，将考勤信息和绩效信息，分别根据员工号码进行对齐分组。A6 中准备一个空序列用来存储每位员工实际工资与标准工资的差额。

在 A7 的代码块中，循环计算每位员工的实际工资与标准工资的差额：B7 中获取标准工资，C7 中存储实际工资；当该员工有缺勤记录时，在 B8 中对缺勤天数进行工资折算；当该员工有绩效记录时，在 B9 中对绩效进行工资折算；在 B10 中，计算出该员工实际工资与标准工资的差额并存入 A6 的序列。

在 A11 中，计算出工资差距最大的前 5 位员工的序号，这个序号和 A1 员工表中的序号是相同的。最后在 A12 中，以这 5 位员工的信息计算出结果序表。

A12 中的结果序表如下：

FullName	Salary
Ashley Smith	25400.0
Ashley Wilson	19900.0
Rachel Johnson	17036.0
Matthew Johnson	16188.0
Rebecca Moore	10914.0

方法二，使用 join：

	A	B	C
1	=demo.query("select * from EMPLOYEE")		
2	=demo.query("select EMPLOYEEID, ABSENCE from ATTENDANCE")		
3	=demo.query("select EMPLOYEEID, EVALUATION,BONUS from PERFORMANCE")		
4	=join@1(A1:Info,EID;A2:Attendance, EMPLOYEEID;A3:Performance, EMPLOYEEID)		
5	>A4.derive(Salary)		
6	for A4	=A4.Info.	=B6
7		if A6.Attendance !=null	>C6=B6*(1-A6.Attendance. ABSENCE/(250/12))
8		if A6.Performance !=null	>C6=C6+B6*(A6.Performance. EVALUATION-1)+A6. Performance.BONUS
9		>A6.Salary=round(C6,2)	
10	=A4.sort(abs(Salary-Info.SALARY):-1)(to(1,5))		
11	=A10.new(Info.NAME+" "+Info. SURNAME:FullName,Salary:Salary)		

在 **A4** 中，通过员工表、考勤表以及绩效表中的员工号码进行等值连接，将每个员工的所有信息整合在一起构成序表，连接时需要使用**@1** 选项执行左连接。**A5** 中在 **A4** 的序表中添加实际工资字段。后面的计算和方法一是类似的。最终　在 **A11** 中返回工资差距最大的前五名员工的信息，结果是相同的：

FullName	Salary
Ashley Smith	25400.0
Ashley Wilson	19900.0
Rachel Johnson	17036.0
Matthew Johnson	16188.0
Rebecca Moore	10914.0

通过 **join** 进行连接，将不同表中的数据整合到同一个序表中，一目了然。在统计计算时，思路会更加清晰。

8. 数据库

8.1 数据库的连接

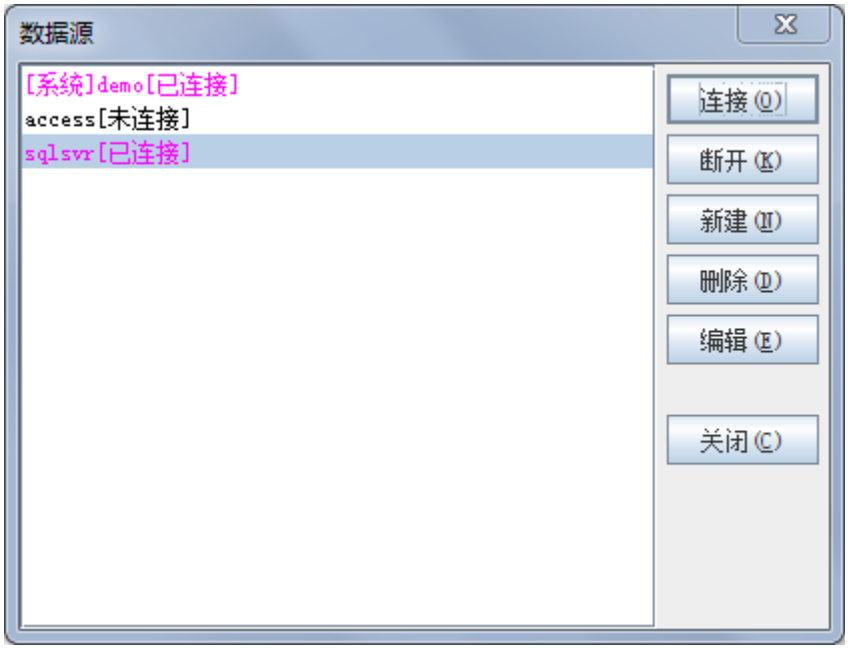
8.1.1 数据源管理器

连接关系数据库的步骤：

1) 在**工具**菜单项中，点击数据源连接**数据连接**打开数据源管理器
2) **新建**数据源，选择类型

3) 选择数据库类型、编码等信息，设置数据源的连接参数，并为之起名

4) 连接，数据源管理器会显示是否连接成功，在数据源管理器中，可以同时连接多个数据库。



8.1.2 数据库连接函数

除了在数据源管理器中连接和断开数据源，集算器中还可以使用数据库连接函数。

➢ **connect(*dbn*)**

连接名称为 *dbn* 的数据源，数据源的名称以及连接参数**在数据源管理器中配置**。返回数据库连接对象 *db*，该对象可以用单元格名直接调用。

❖ **@e** 出错时返回错误信息，由代码处理，否则将中断连接。

当 **connect** 函数使用**@e** 选项时，可以用代码查看错误信息：

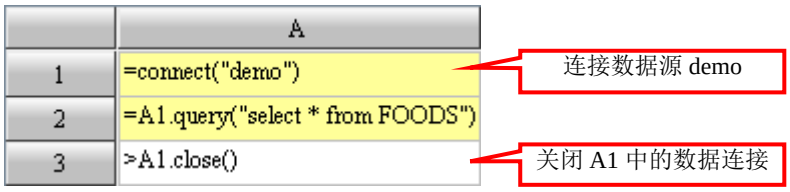
➢ ***db*.error()**

返回上一条与数据库相关的命令的错误信息代码，0 表示无错。

❖ **@m** 返回上一条与数据库相关的命令的错误信息串。

➢ ***db*.close()**

关闭数据库连接对象 *db*。



8.1.3 数据库的提交与回滚

在集算器中，在执行某些数据库函数时，是可以不进行自动提交的，这时，可以用函数来控制数据库的提交或回滚操作。

➢ ***db*.commit()**

将对数据库的操作提交到数据库。

➢ ***db*.rollback()**

回滚数据库，取消未提交的操作。

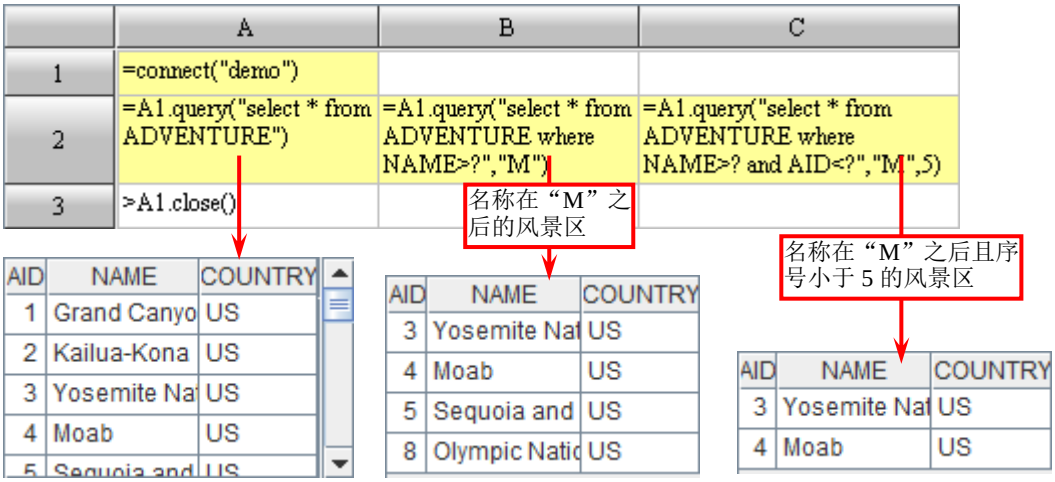
8.2 读取数据库中的数据

8.2.1 用 SQL 读取序表

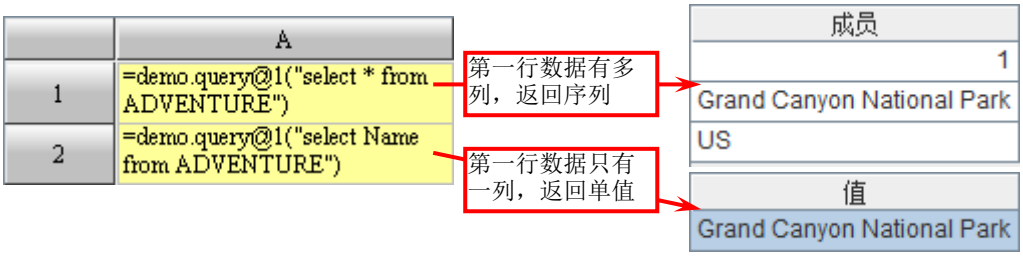
➢ ***db*.query(*sql*…)**

在数据源 *db* 中执行 *sql* 语句，返回结果序表。*sql* 语句可以带参数，写在后面即可。

- ***db*.query("select * from employee")**
- ***db*.query("select * from employee where dept=?", 3)**

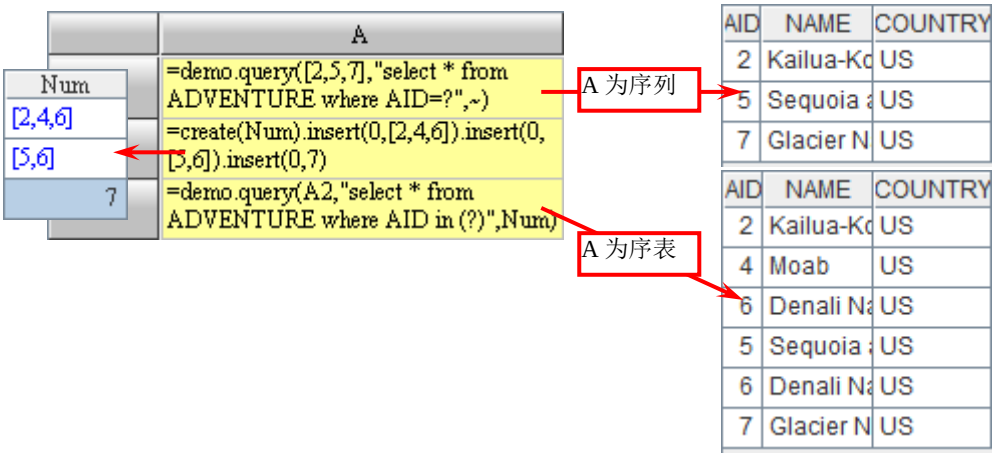


❖ **@1** 只返回第一行的数据，返回值为单值或者序列，而不是序表。



➢ ***db*.query(*A*,*sql*…)**

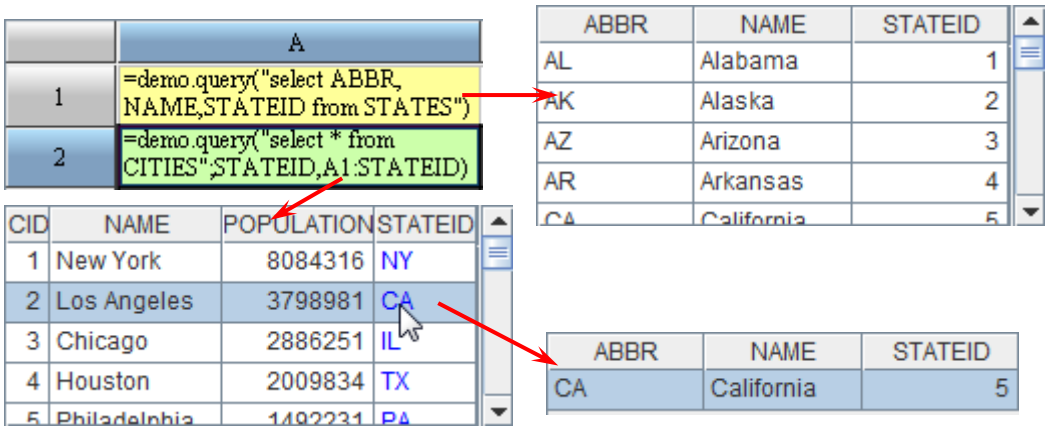
对序列 *A*，在数据源 *db* 中执行 *sql* 语句，返回结果集合并的序表。*sql* 语句可以带参数，写在后面即可。



如果在读取序表之后，需要对序表进行 **switch** 操作，可以简写：

➢ ***db*.query(…; …)**

先执行 **query** 函数，然后对结果序表进行 **switch** 操作。



8.2.2 *在数据库中执行存储过程

- ***db*.proc(*sql*,*a*:*t*:*m*:*v*,…)**

调用存储过程计算，如有返回结果，由参数中的 *v* 带出。

存储过程与普通 **SQL** 不同，除了存在输入参数，还会存在输出参数。存储过程的返回结果，需要由输出参数 *v* 带出。

8.3 写入数据库

8.3.1 在数据库中执行 SQL 命令

- ***db*.execute(*sql*,…)**

在数据源 *db* 中执行 *sql* 语句，…是参数。与 **query** 不同的是，**execute** 不返回结果集，一般用于改写数据库。

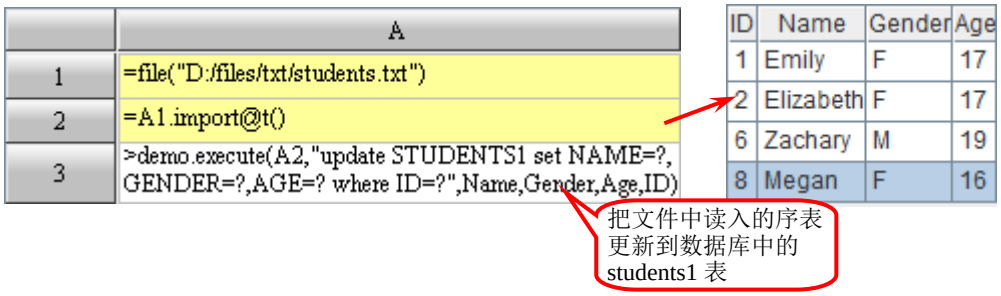
- ***db*.execute("delete from EMPLOYEE where DEPT=?", 3)**

与 **query** 函数类似，**execute** 函数还可以针对一个排列（序列）使用：

- ***db*.execute(*A*,*sql*,…)**

针对序列 *A* 的每个成员执行 *sql*，*A* 是排列时，参数表达式中可以像循环函数一样使用 *A* 的字段。

- ***db*.execute(*A*,"update EMPLOYEE set NAME=? where EID=?",Name,Id)**



execute 函数通常会在执行后自动提交到数据库，如果需要自行控制，可以使用选项。

- **@k** 执行 **SQL** 语句后，不提交事务。
- **@s** 执行静态 **SQL** 语句而不去预解析，此时不能使用参数。

当使用**@k** 选项时，事务不会自动提交，需要使用 ***db*.commit()**函数进行提交，或者使用 ***db*.rollback()**函数回滚，取消修改。

	A	B
1	=file("D:/files/txt/students.txt")	
2	=A1.import@t()	
3	=connect@e("demo")	建立数据库连接
4	>A3.execute@k(A2,"update STUDENTS1 set NAME=?, GENDER=?,AGE=? where ID=?",Name,Gender,Age,ID)	
5	=A3.error()	查看 A4 执行后的错误代码
6	if A5==0	>A3.commit() 无错误则提交
7	else	>A3.rollback() 有错误则回滚
8	>A3.close()	

8.3.2 更新整个序表

写出整个序表还可以用 **update** 函数。

- ***db*.update(*A*,*tbl*,*F*$_i$:*x*$_i$,…;*P*,…)**

根据 *A* 更新 *tbl* 表中的字段 *F*$_i$，取值为 *x*$_i$，省略用同名字段值。表达式中 *P*,…为更新时使用的主键，当未指定主键时，将从表 *tbl* 中读取，如果未读取到主键则使用 *A* 的主键。

- ***db*.update(A1,EMPLOYEE, NAME:FullName,EID)**

A 通常为排列，对于 *A* 中每条记录，如果 *tbl* 中该主键数据已存在，则将生成 **UPDATE** 语句，否则生成 **INSERT** 语句。

	A
1	=file("D:/files/txt/students.txt")
2	=A1.import@t()
3	>demo.update(A2,STUDENTS1,ID,NAME:Name,GENDER:Gender,AGE:Age;ID)

ID	Name	Gender	Age
1	Emily	F	17
2	Elizabeth	F	17
6	Zachary	M	19
8	Megan	F	16

把文件中读入的序表更新到数据库中的 STUDENTS1 表

使用 update 函数时，还经常使用下面几个选项来进行不同的操作：

- **@u** 只生成 **UPDATE** 语句。
- **@i** 只生成 **INSERT** 语句。
- **@a** 在执行更新前先清空目标表。
- **@k** 执行更新后，不提交事务，缺省将自动提交。
- **@1** 第一个字段为自增字段，在执行 **update** 时，不对它赋值。

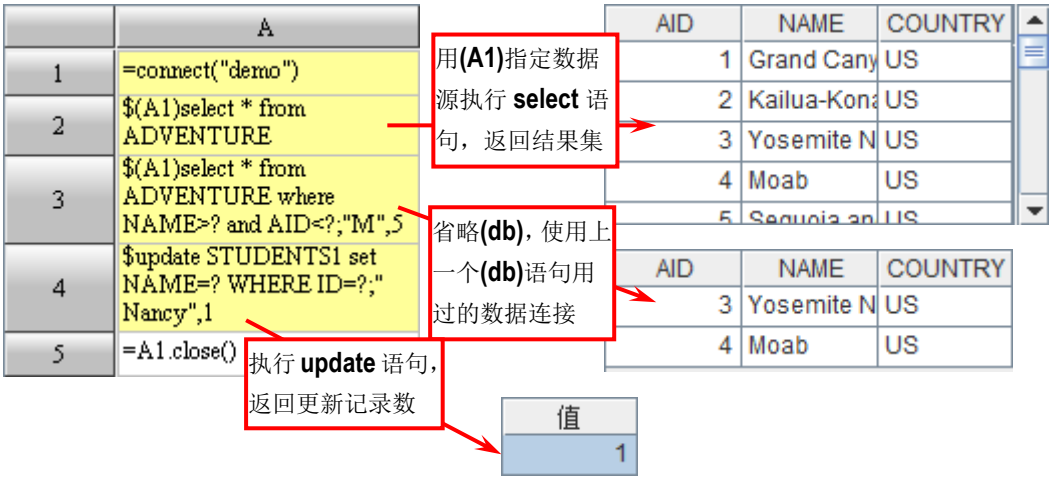
8.4 直接使用 SQL

- **$(*db*)*sql*;…**

直接在单元格中执行 *sql* 语句，在数据连接 *db* 中执行，如果(*db*)省略，则取之前最后一次使用的 *db*。*sql* 语句可以带参数，写在分号后面即可。使用 *sql* 语句时不在前方加等号，

也不必将语句用引号标记，且只能以 **select/insert/update/delete** 作为开头。如果使用 **select** 语句会返回结果集。

- **$(*db*)select * from EMPLOYEE**
- **$select * from EMPLOYEE where DEPT=?;3**
- **$update EMPLOYEE set NAME=? where EID=?;"Harry",1**

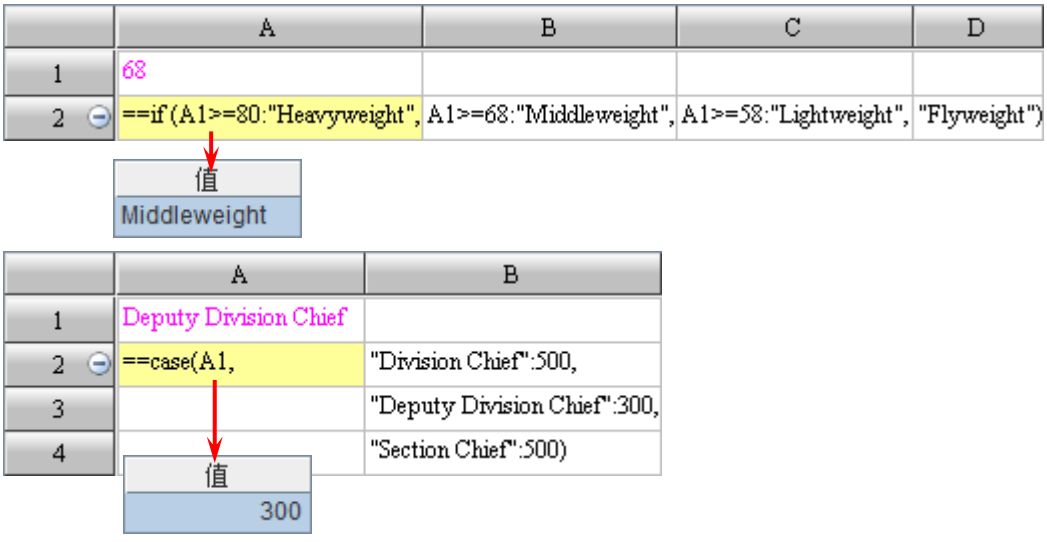


9. 高级代码与应用

9.1 长语句

某些较长的表达式写在一个格内看不全，可以用**续格规则**写在多个单元格中。

计算格或执行格以字符,、;、(结尾时，将自动拼接下一格内容，直到不是这些字符结尾或本代码块结束。

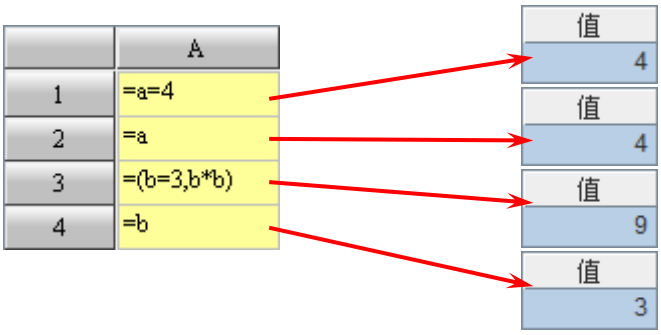


长语句的首格一般要用双等号"**==**"或者双大于号"**>>**"开头，这种写法将告诉集算器将以该格为主格的代码块整个作为一条语句处理，否则集算器还将再次执行该代码块中的其它格，可能会导致错误。

9.2 子计算语句

延用JAVA的习惯，赋值语句也作为一种表达式，将返回其值本身，并且支持括号运算符。

- **a=4** 将 **a** 赋值为 4 的同时将返回 4。
- **(a=3,a*a)** 将 **a** 赋值为 3，同时返回表达式的计算结果 9。

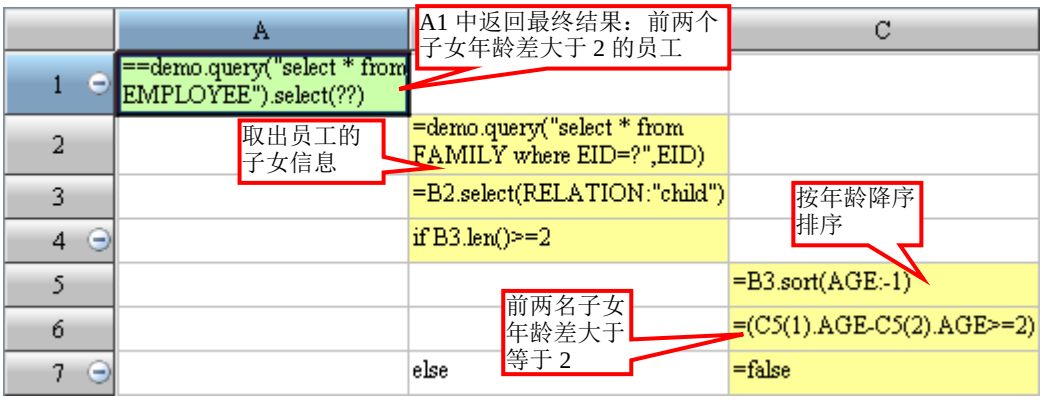


有的长语句比较复杂，其中还可能使用上述的临时变量做计算，如：

- **demo.query("select * from EMPLOYEE").select((a=demo.query("select * from FAMILY where EID=?",EID).select(RELATION:"child").sort(AGE:-1),if(a.len()<2,false,a(1).AGE-a(2).AGE>= 2)))**

将找出前两个子女年龄差大于等于 2 的员工

在这个长语句中使用了临时变量，无法用上述的长语句机制写到多个单元格中，这时候可以使用子计算语句规则将其写到多个单元格中。



在 A1 格中使用了**??**运算符。

➢ **??**

??相当于一个函数，它将依次计算以当前格为主格的代码块中除当前格外的单元格，并返回最后一个计算格的格值。

采用这样的写法，整个语句更清晰了。

作为一种特殊的长语句，子计算语句也要用双等号"**==**"开头，否则集算器还会再次执行代码块中其它格而导致错误。

9.3 赋值块、执行块和注释块

9.3.1 赋值块和执行块

上面例子中的长语句和子计算语句中，主格的格串均以==开头而非=。

这样书写的目的是告诉集算器，以该格为主格的整个代码块是一个语句，要整体计算，而不可拆开。这时，集算器在按规则执行完本条语句后将忽略本代码块中的其它单元

格，否则在缺省状态下，集算器还会继续计算代码块内剩余的单元格，会造成流程混乱，有可能会产生错误。

对于执行格类似，当以>>开头时表示以该格为主格的整个代码块是一个语句。

	A	B
1	=connect@e("demo")	=A1.query("select * from EMPLOYEE")
2	>A1.execute("drop table EMPLOYEE1")	>A1.execute("create table EMPLOYEE1 (EID int,FULLNAME varchar(30), GENDER varchar(10))")
3	>>A1.execute(	
4		B1,
5		"insert into EMPLOYEE1 (EID,
6		FULLNAME, GENDER) values (?,?,?)",
7		EID,NAME+" "+SURNAME,
8		GENDER)
9	=A1.query("select * from EMPLOYEE1")	
10	>A1.close()	

EID	FULLNAME	GENDER
1	Rebecca Moore	F
2	Ashley Wilson	F
3	Rachel Johnso	F
4	Emily Smith	F
5	Ashley Smith	F

这种代码块，按照执行语句的不同，分别称为**赋值块**和**执行块**。对应地，前面出现的以 "=="开头的语句块称为**计算块**。

9.3.2 注释块

类似地，将单元格的格串以//开头时，表示以该格为主格的整个代码块都是注释。这个代码块称为注释块。集算器碰到这种格串时会直接跳过整个注释块。

	A	B
1	//comment:	
2		1.note1...
3		2.note2...
4		3.note3...
5	=1+3	

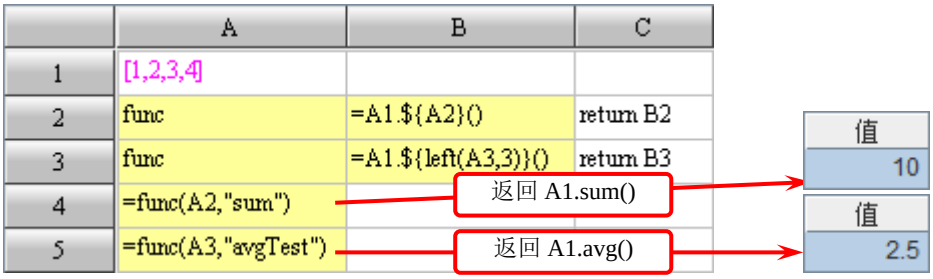
注释块首格的代码块中，都是注释

注释块结束行开始是代码

9.4 宏

为了增加代码的灵活性，有时需要动态生成语句串，这时可以使用宏。

宏用符号**${}**括起来，集算器将先计算**${...}**里的...表达式，将计算结果作为宏字符串值替换**${}**。如果**${}**位于引号内则不替换。在替换之后，集算器再进行语句解析，这样就可以实现动态生成表达式。

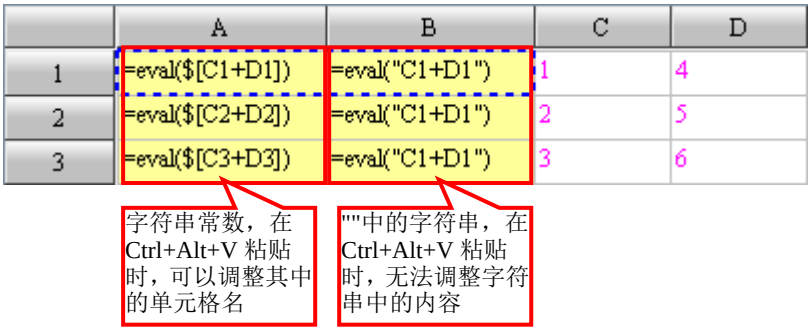


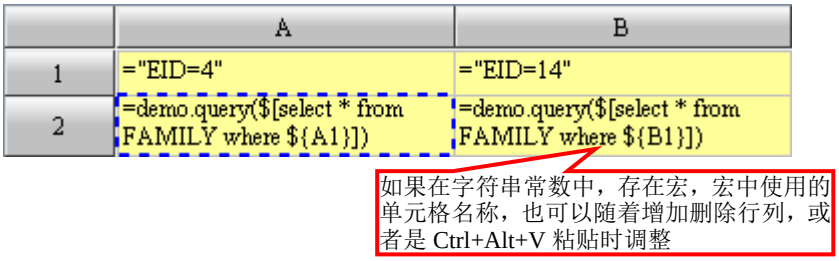
9.5 字符串常数

$[...]表示以...为内容的字符串常数，在集算器表达式中和**"..."**含义相同。

但在集算器的编辑界面中，**$[...]**和**"..."**则有很大的差别。**$[...]**中出现的单元格名将会随着编辑操作和其它正常表达式一起变迁，也可以在复制粘贴等操作时选择调整表达式，而**"..."**的内容则不会随着编辑操作而改变。

这样，在某些需要字符串也能跟随编辑而变迁的特殊情况下，就可以使用**$[...]**代替**"..."**，如 **eval** 的参数，**query** 中的 **SQL** 语句等，这些参数都表现为字符串，实质却是要被解析执行的表达式或 **SQL** 子句。





在**2.2.2字符串常数格**中，我们知道可以用'起头来定义字符串常数，字符串常数格中的数据会直接解析为字符串，不必再添加转义符。在字符串常数格中，可以定义等号开头，或者包含特殊字符的字符串。

9.6 特殊标识符

某些标识符可能含有非字母数字的字符，如%、()、空格等，此时可能被引擎误解析成百分号、括号。为了避免，可以用单引号'...'将标识符括起来，从而把标识符和特定符号区

分开。

	A
1	=demo.query("select * from STATES")
2	>A1.alter(ABBR,POPULATION,Area (Sq.mi.):AREA)
3	>A1.derive(round(POPULATION/Area (Sq.mi.)',2):Population/Area)

在列名中存在括号、空格、小数点等特殊字符，为了不产生混淆，需要用单引号标明

ABBR	POPULATION	Area (Sq.mi.)	Population/Area
AL	4779736	52419	91.18
AK	710231	663267	1.07
AZ	6392017	113998	56.07
AR	2915918	52897	55.12
CA	37253956	163700	227.57

9.7 跨网格调用

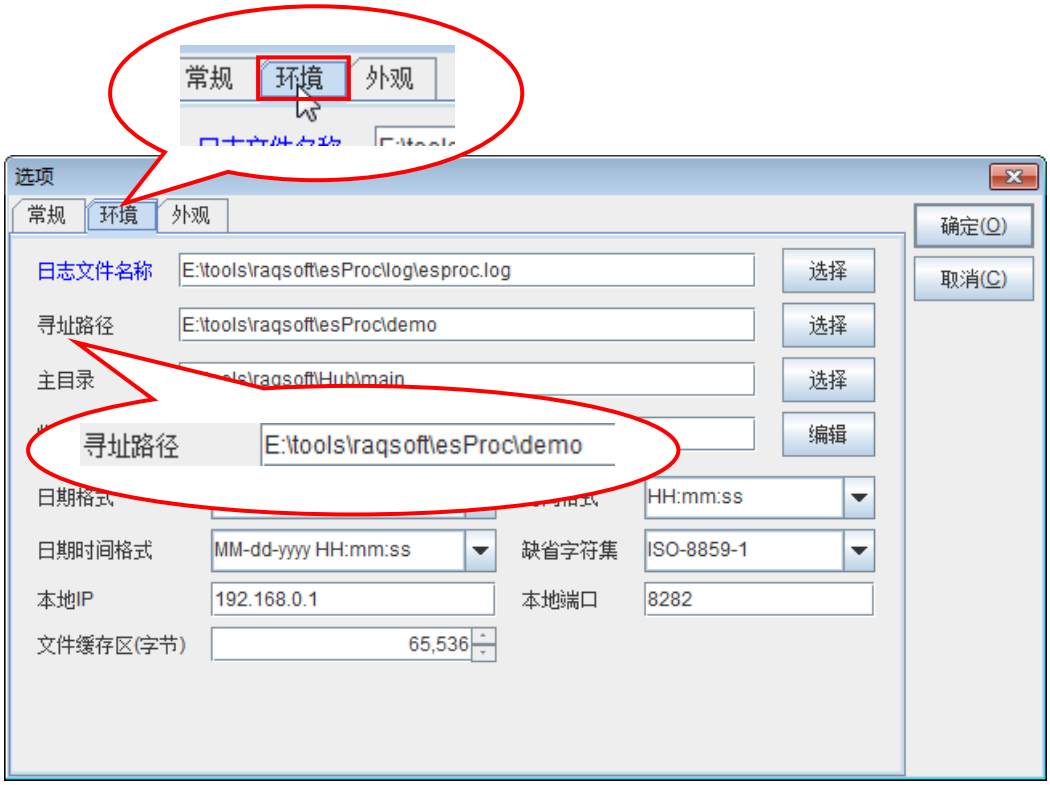
9.7.1 文件路径

❖ **file@s(*fn*)** 定义路径为 *fn* 的文件对象，*fn* 中可以不包含文件路径，对于不含路径的文件名，集算器会优先在类路径中查找，然后在寻址路径列表中查找。

	A
1	=file("D:/files/txt/students.txt")
2	=file@s("students.txt")

文件名中如果不是完整文件路径，可以用@s 选项，说明文件在应用路径中

寻址路径列表可以在菜单栏中点击**工具**菜单项中的**选项**按钮，在选项设定中的**环境**选项卡进行设置，当需要设定多个路径时，用分号分隔：



9.7.2 网格文件的返回值与结束

➢ **result x_i,…**

在集算器中，可以将整个网格作为一个子程序来执行，在网格中用 result 来返回表达式 x_i,…的计算结果，返回后结束程序执行，释放资源。

9.7.3 跨网格调用

在集算器中，可以进行跨网格调用，在程序中运行其它网格中的程序，或是使用其它网格中的计算结果。

➢ **call(*dfx*,…)**

调用网格 *dfx*，并在计算前将…作为参数依次设入。指定 *dfx* 时可以直接使用网格文件的绝对路径名，如果不使用绝对路径时，则自动使用**@s** 选项查找文件。如果 *dfx* 中存在网格参数，那么需要在调用 **call** 时使用参数…，这些参数值将依次设入 *dfx* 的各个参数中，与 *dfx* 参数列表中的参数名无关。*dfx* 还可以是文件对象，此时直接读出执行。

比如，网格文件 sort.dfx 的作用是将参数 arg1 中的序列，按照参数 desc 的设定来排序，desc 为 true 时降序，为 false 时升序：

	A	B
1	if desc	result arg1.sort(~:-1)
2	else	result arg1.sort()

每次运行前设置参数

序号	名称	值	备注
1	arg1	[1,5,3]	
2	desc	true	

可以在其它文件中，对 sort.dfx 进行跨网调用：

	A
1	=call("sort.dfx",[1,5,2,3],false)
2	=call("sort.dfx",[1,4,3],true)

将序列[1,5,2,3]
升序排序

成员
1
2
3
5

将序列[1,4,3]
降序排序

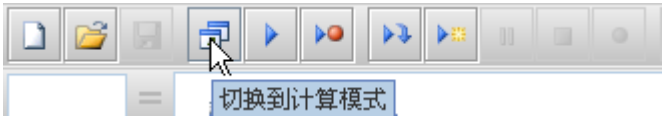
成员
4
3
1

单元格中，用 **result** 返回多个结果时，会返回序列。

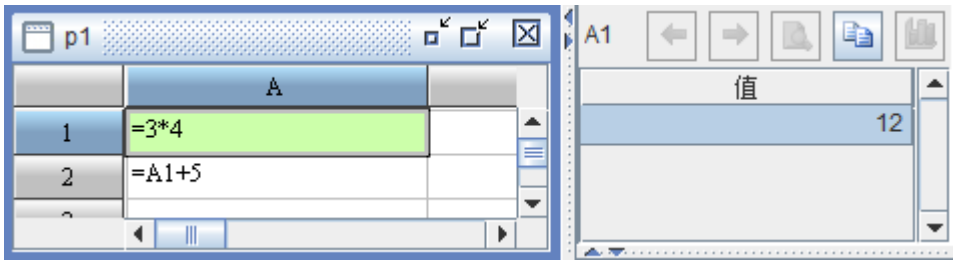
9.8 立即计算模式

集算器还可以在立即计算模式下运行。

在工具栏中点击“**切换到计算模式**”按钮，可以进入立即计算模式。



在立即计算模式下，在单元格上按下回车键 Enter，可以立即计算该单元格的格值：



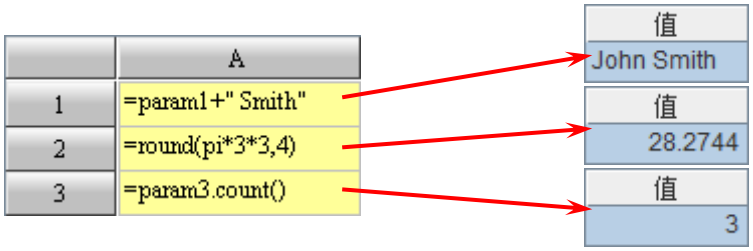
在立即计算模式下，由于每次只会计算选定的单元格值，因此在计算时，需要注意计算顺序。如果表达式中调用未计算的单元格时，就有可能产生错误。

9.9 在网格中使用常量

在网格计算中，经常会用到参数。对于一些会重复使用的参数，可以设定为常量。常量的使用和网格参数是相同的，用常量名称调用即可。但是常量与网格参数也有所不同，常量在网格中是不允许赋值的，而参数则可以赋值。

查看**网格常量**，可以点击 **工具** 菜单中的 **网格常量** 按钮：

常量可以是各种数据类型，如字符串、整数、日期、序列等等。在使用时，直接用常量名称调用：



9.10 用命令行执行网格文件

在集算器安装目录的 esProc\bin 路径下，可以找到 esprocx.exe 文件，在 dos 环境下可以

用命令行来计算网格文件。

➢ **esprocx <options> <dfxFile> [arg0] [arg1]...**

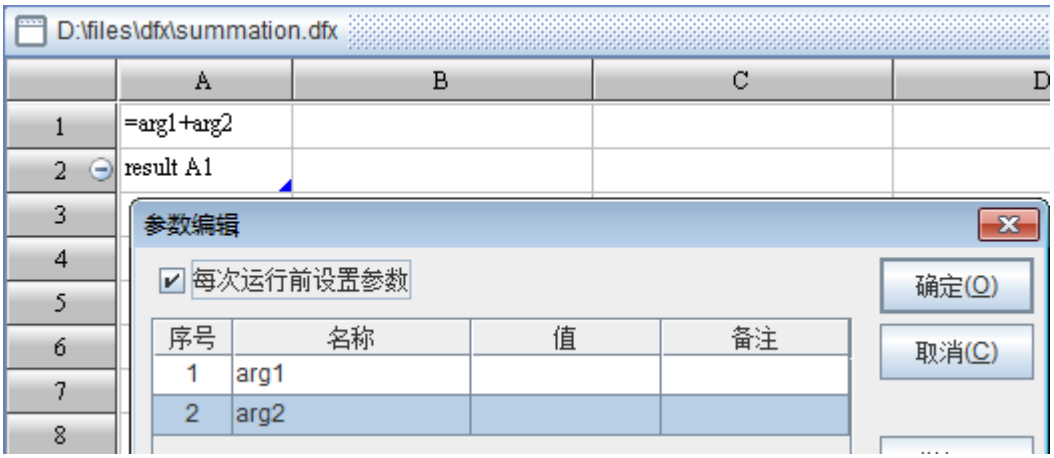
集算器可以在 Dos 下用命令行启动。命令行中的 **option** 为启动设置选项，**dfxFile** 为集算器计算的网格文件，**[arg0] [arg1]**...为计算使用的网格参数。计算网格文件时，程序会自动搜索 config.xml 读取环境配置信息。该文件包含了集算器的基本配置信息，如注册码、寻址路径、主目录、数据源配置等。

❖ **-R** 将 result 结果输出，序表/排列用 export 文本方式输出。

– **esprocx C:\c.dfx** 执行计算网格文件 c.dfx。

用命令行执行网格文件时，在屏幕上会打印出开始执行以及结束执行的消息。

（一）无输出结果



执行命令传入参数 100 和 5。运行结果如图：

```
D:\Program Files64\esProc\bin>esprocx F:\summation.dfx 100 5
[2014-01-23 16:01:21] : [DEBUG]  - Before calculating:Thu Jan 23 16:01:21 CST 20
14
[2014-01-23 16:01:21] : [DEBUG]  - After calculated:Thu Jan 23 16:01:21 CST 2014


D:\Program Files64\esProc\bin>
```

（二）使用**-R** 命令打印出 **result** 计算结果。

执行命令后。运行结果如图：

```
D:\Program Files64\esProc\bin>esprocx -R F:\queryEmployee.dfx
[2014-01-23 15:59:55] : [DEBUG]  - Before calculating:Thu Jan 23 15:59:55 CST 20
14

Result in A4:
综合部  肖梅    8000
研发部  李四    9000
销售部  李芳    10000
综合部  郑建杰  6000
研发部  赵军    4000
销售部  孙林    5500
研发部  金士鹏  3000
销售部  刘英玫  3500
销售部  张雪眉  5000
研发部  许赵阳  6000
研发部  沈蓉芳  7000
[2014-01-23 15:59:56] : [DEBUG]  - After calculated:Thu Jan 23 15:59:56 CST 2014


D:\Program Files64\esProc\bin>
```

10. 文件系统与游标

在数据计算和分析处理的过程中，文件访问是常见且重要的需求。文件的读写，文件数据的计算等，都属于对文件的访问。

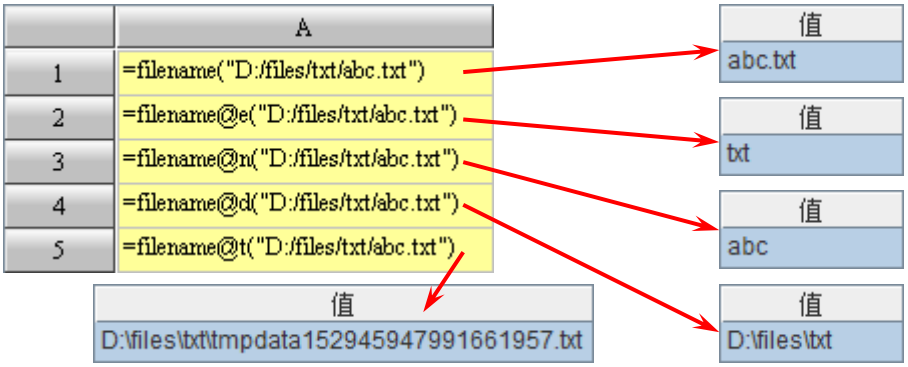
10.1 文件系统

在**2.8用文件对象读写数据**中，讲解了集算器中文件对象的基本使用，在这里，进一步讲解文件系统的相关函数。

- **filename(*fn*)**

  拆分出全路径文件名 *fn* 的文件名部分，带扩展名。

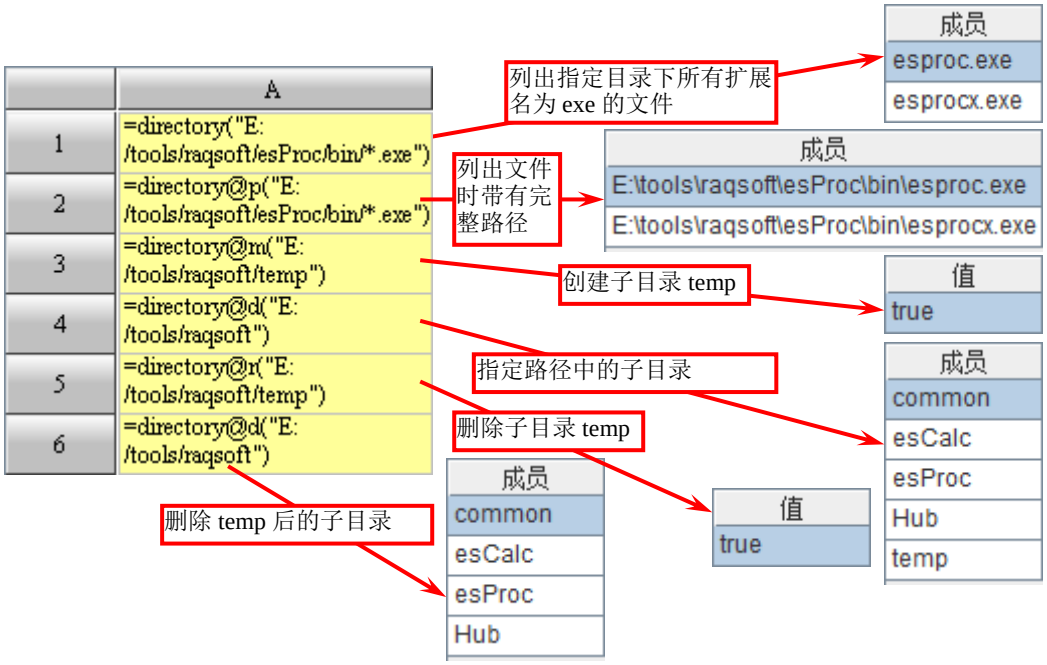
- **@e**　只拆分出扩展名
- **@n**　拆分出的文件名不带扩展名
- **@d**　拆分出目录部分
- **@t**　仅限于本地使用，在 *fn* 同目录下，产生同扩展名的临时文件，并返回文件名。当 *fn* 省略时，在系统临时目录下，产生相对于主目录的文件名。



- **directory(*path*)**

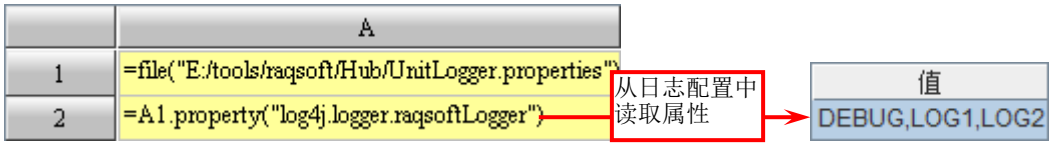
  仅限本地使用，列出满足通配符路径 *path* 的文件名，不包含路径名。

- **@d** 列出子目录
- **@p** 仍带有完整路径名
- **@m** 创建目录
- **@r** 删除空目录



- ***f*.property(*p*)**

从属性文件对象 *f* 中，获取名称为 *p* 的属性的值；当 *p* 省略时，返回所有属性构成的序表。



10.2 数据库游标

在集算器中使用数据库游标，可以实现分批读入较多数据。

- ***db*.cursor(*sql*,…)**

根据 *sql* 创建数据库游标返回。

- ***cs*.fetch(*n*:*x*)**

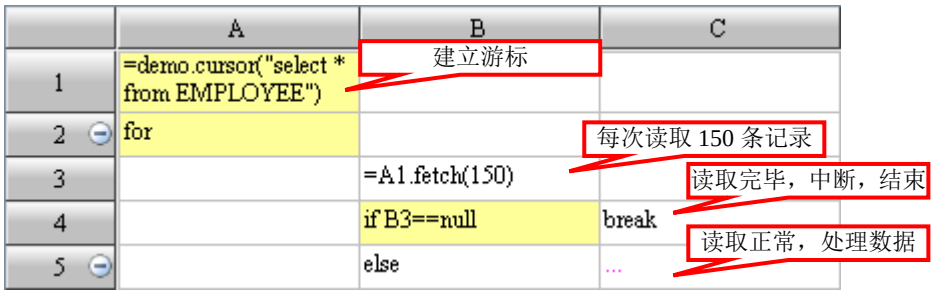
用游标 *cs* 读出 *n* 条记录或者至表达式 *x* 有变化，形成序表返回，已到头返回 null。当数据库中的剩余记录不足 *n* 条时，读出所有剩余记录，所有记录读完后关闭游标。

- ***cs*.skip(*n*:*x*)**

跳过 *n* 条记录或者至表达式 *x* 有变化，返回实际跳过的记录数。

- ***cs*.close()**

直接关闭游标。

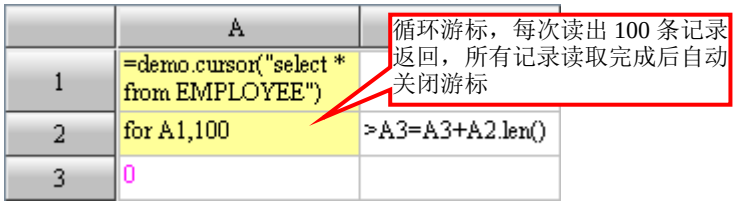


	A	B	C
1	=demo.cursor("select * from EMPLOYEE")		
2	for		
3		=A1.fetch(150)	
4		if B3==null	break
5		else	...

当使用 **fetch** 或 **skip** 执行完游标中的最后一条记录后，游标会自动关闭。

- **for *cs*,*n*:*x***

循环游标，每次读 *n* 条记录，或读出记录至 *x* 改变，返回成序表，读取结束后关闭。



	A	
1	=demo.cursor("select * from EMPLOYEE")	
2	for A1,100	>A3=A3+A2.len()
3	0	

10.3 文件对象游标与内存排列游标

在 **10.2 数据库游标**一节中，我们介绍了在读取数据库数据时使用游标的基本方法。除此之外，集算器中还可以在读取文件数据时使用游标，也可以用内存排列构成游标。

10.3.1 文件对象游标

- ***f*.cursor(*F*$_i$:*type*$_i$,…;*s*,*b*:*e*)**

根据文件 *f* 创建游标返回。 创建游标与从文件中读取序表类似，也可以定义数据的起止位置，需要读出的字段等；也同样可以使用**@t**、**@b**、**@z** 和**@s** 等各种选项。相关内容可以查看 **5.3.1 从文件中读写序表**这一小节的内容。

- **@x**　　文件对象关闭时自动删除文件。

文件对象游标与数据库游标的使用方法是相同的：

	A	B	C
1	=file("D:/files/txt/employee.txt")		
2	=A1.cursor@t()		
3	for	=A2.fetch(150)	
4		if B3==null	break
5		=B3.select(STATE=="California")	
6		>A7=A7|B5	
7			

在这个例子中，每次用文件对象游标读取 150 名员工的数据，并从中选出 California 的员工，存储在 A7 中，结果如下：

EID	NAME	SURNA...	GENDER	STATE	BIRTHD...	HIRED...	DEPT	SALARY
1	Rebecca	Moore	F	California	1974-11-	2005-03-	R&D	7000
6	Matthew	Johnson	M	California	1984-07-	2005-07-	Sales	11000
8	Megan	Wilson	F	California	1979-04-	1984-04-	Marketing	11000
23	Joseph	Turner	M	California	1983-08-	2003-08-	Finance	6000
27	Alexis	Jones	F	California	1983-12-	2003-12-	Marketing	10000
29	Olivia	Harris	F	California	1979-08-	2009-08-	Sales	8000

➢ ***F*.cursor()**

根据二进制文件对象序列 *F* 创建游标返回，其中每个文件占用一个字段。

10.3.2 内存排列游标

➢ ***P*.cursor()**

将某个排列转为游标，目的就是为了将排列数据用于游标函数中使用，如 **merge@x**，**join@x** 等。

	A	B	C
1	Administration	4	40000
2	Finance	24	177500
3	HR	19	138000
4	Marketing	99	733500
5	Production	91	663000
6	R&D	29	239000
7	Sales	187	1362500
8	Technology	47	344000
9	=create(DEPT,Count,TotalSalary).record([A1:C8])		
10	=A9.cursor()		

A10 中的游标即由 A9 中序表生成的内存排列游标。

还可以用某个 dfx 文件生成游标：

➢ **pcursor(*dfx*,…)**

调用网格 *dfx*，并在计算前将…作为参数依次设入，将其中用 **result** 命令返回的计算结果转为游标。

10.4 游标的使用

10.4.1 使用游标读取数据

除去在 **10.2 数据库游标**一节中，曾经介绍过的游标的读取、略过等基本方法，在集算器中还可以使用很多的游标相关函数，在这一小节中将继续讲解。

➢ ***cs*.select(*x*)**

将游标 *cs* 中的记录根据条件 *x* 进行筛选，返回成游标。相当于对 *cs* 中的记录执行 **select(*x*)**。

❖ **@c** 只返回第一段连续使 *x* 为 true 的记录。

	A
1	=file("D:/files/txt/employee.txt")
2	=A1.curson@t()
3	=A2.select(STATE=="California")
4	=A3.fetch()
5	=A2.fetch()

用游标直接筛选数据，和取数后再筛选结果一致，A4 中的结果，与 **10.3.1 文件对象游标**中的相同。需要注意的是，由于 A3 中的游标是用 A2 中的游标生成的，所以 A3 中的游标是使用 A2 中的游标来读取数据的，当 A3 中游标取数完毕时，A2 中的游标也已经完成了取数，因此 A5 中的结果是 null。

- ***f*.export(*cs*,*x*:*F*,…;*s*)**

将游标 *cs* 的数据写出到文件对象 *f* 中。支持 **export** 函数的各个选项**@t**、**@a**、**@b** 和**@g**。与 **5.3.1 从文件中读写序表**中的 ***f*.export(*A*,*x*:*F*,...;*s*)**类似，本函数也可以指定 *x*,*F*,*s* 等参数。

- ***F*.export(*cs*,x_i:F_i,…;*s*)**

将游标 *cs* 的数据分别写出到文件对象序列 *F* 中的各个文件对象中。每个字段写出到一个文件。

❖ **@a** 追加写。

- ***cs*.new(*x*…)**

根据 *x* 中的定义创建新序表，取数时，用游标 *cs* 中的记录计算表达式获得新记录，返回成游标，类似于对 *cs* 中的记录执行 **new(*x*...)**。

	A
1	=file("D:/files/txt/employee.txt")
2	=A1.curson@t()
3	=A2.new(NAME+" "+SURNAME:FULLNAME, interval@y(BIRTHDAY,now()):AGE)
4	=A3.fetch()
5	=A2.fetch()

用文件中的数据，算出员工的全名和年龄并返回，***cs*.new()**类似于用序表生成新序表。同样，A3 中新的游标在读取数据时需要使用 A2 中的游标，A3 读取完成后，A2 也已经读取完成。

A4 中计算结果如下：

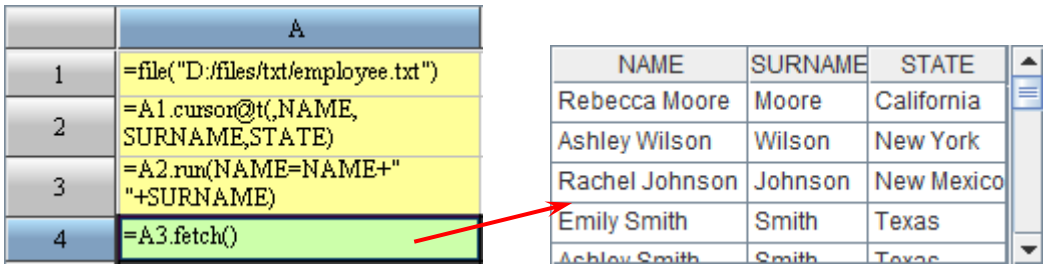
FULLNAME	AGE
Rebecca Moore	39
Ashley Wilson	33
Rachel Johnson	43
Emily Smith	28
Ashley Smith	38
Matthew Johnson	29

➢ ***cs*.derive(…)**

为游标 *cs* 添加计算字段，返回成新的游标，类似于对 *cs* 返回的结果集执行 **derive(**…**)**。

➢ ***cs*.run(*x*)**

用游标 *cs* 中的记录计算表达式 *x* 获得新记录，返回成游标，类似于对 *cs* 中的记录执行 **run(*x*)**。



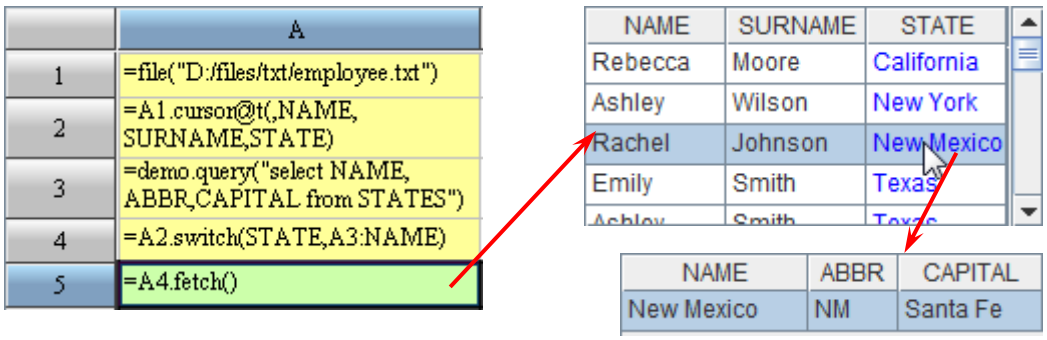
	A
1	=file("D:/files/txt/employee.txt")
2	=A1.curson@t(,NAME,SURNAME,STATE)
3	=A2.run(NAME=NAME+" "+SURNAME)
4	=A3.fetch()

NAME	SURNAME	STATE
Rebecca Moore	Moore	California
Ashley Wilson	Wilson	New York
Rachel Johnson	Johnson	New Mexico
Emily Smith	Smith	Texas

➢ ***cs*.switch(*F*$_i$, *A*$_i$:*x*;…)**

用游标 *cs* 中的记录进行 **switch** 转换获得新记录，返回成游标，类似于对 *cs* 中的记录执行 **switch(**…**)**。

❖ **@i** 在计算中，如果某条记录对每一个 *A*$_i$ 全部无法找到 *F*$_i$ 的对应值，则删除整条记录。



	A
1	=file("D:/files/txt/employee.txt")
2	=A1.curson@t(,NAME,SURNAME,STATE)
3	=demo.query("select NAME,ABBR,CAPITAL from STATES")
4	=A2.switch(STATE,A3:NAME)
5	=A4.fetch()

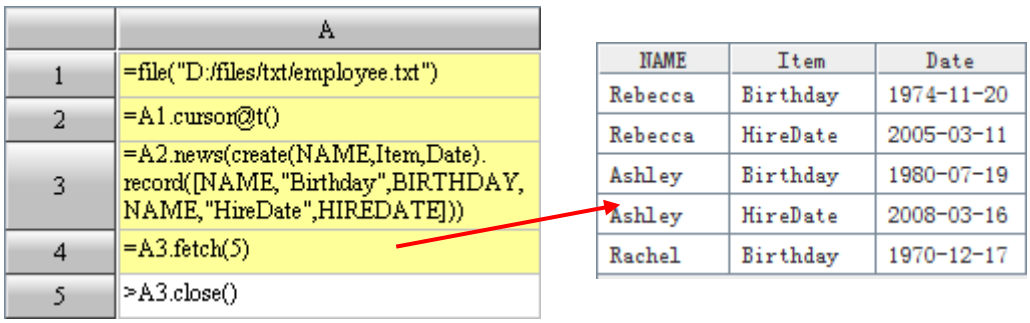
NAME	SURNAME	STATE
Rebecca	Moore	California
Ashley	Wilson	New York
Rachel	Johnson	New Mexico
Emily	Smith	Texas

NAME	ABBR	CAPITAL
New Mexico	NM	Santa Fe

从上面的几个游标函数中可以看到，当游标函数生成了新的游标时，旧的游标会失效，原则上不应该再使用，否则在获取数据时，这些游标会互相影响，取数时有可能出现未知情况或者取不到数据。

➢ ***cs*.news(…)**

根据表达式…，用游标 *cs* 中的记录执行运算，将每一条记录拆分为序列或排列，并将结果中的成员或记录并在一起，返回成游标，类似于对 *cs* 中的记录执行 **news(...)**。



	A
1	=file("D:/files/txt/employee.txt")
2	=A1.curson@t()
3	=A2.news(create(NAME,Item,Date).record([NAME,"Birthday",BIRTHDAY,NAME,"HireDate",HIREDATE]))
4	=A3.fetch(5)
5	>A3.close()

NAME	Item	Date
Rebecca	Birthday	1974-11-20
Rebecca	HireDate	2005-03-11
Ashley	Birthday	1980-07-19
Ashley	HireDate	2008-03-16
Rachel	Birthday	1970-12-17

可以看到，在用 **fetch** 函数取数时，记录条数按照拆分后的结果计算。

➢ ***cs* regex(*rs*,*F*$_i$,…)**

对串游标 *cs* 的每个成员执行 regex(*rs*)，将匹配结果拼成以 F_i,…为字段名的序表。

- **@i**　大小写字母不敏感。
- **@u**　使用 unicode 匹配。

游标的 **regex** 函数与序表的相似，所不同的是，仍然是在执行 **fetch** 取数时才会执行。

10.4.2 游标序列的归并与无条件连接

在有些时候，数据来自于多个游标，为此集算器提供了一些函数，可以将多个游标中的数据合并或连接在一起。

- ***CS*.merge@x(*x*$_i$,…)**

将游标序列 *CS* 中的记录归并计算，并返回游标。归并时，要求每个游标中的记录都必须对表达式 x_i,…有序且顺序相同。与 ***A*.merge()**类似，函数中同样可以使用**@u**、**@i** 和**@d** 选项，在归并时做并集、交集或差集处理。

如下面的例子，各个部门的员工信息都存储在与部门名称相同的 txt 文件中，如销售部员工信息存储在 Sales.txt 中，每个部门中的员工数据均已按员工编号 EID 从小到大排序。现在将每个部门的数据用文件对象游标读出后，按员工编号列出所有部门的员工信息。为此需要将各个游标中的记录归并：

	A
1	[Finance,Sales,HR,Administration, Marketing,Production,R&D,Technology]
2	=A1.(file("D:/files/txt/"+~+".txt"))
3	=A2.(~.cursor@t())
4	=A3.merge@x(EID)
5	=A4.fetch()

A2 根据 A1 中的部门序列，生成各个部门的文件对象序列；A3 用文件序列生成文件对象游标序列，生成游标时，用**@t** 选项，将文本文件首行读出，构成各个游标的数据结构；A4 将 A3 中各个游标的数据归并；在 A5 中可以查看结果如下：

EID	NAME	SURNAME	GENDER	STATE	BIRTHDAY	HIREDATE	DEPT	SALARY
59	David	Texas	M	Pennsylva	1977-12-24	2007-12-24	Sales	5000
60	Taylor	Smith	F	Texas	1974-12-16	2004-12-16	Productio	12000
61	Joseph	Smith	M	California	1974-11-11	2006-04-01	Sales	5000
62	Emily	Jones	F	Texas	1977-05-06	2008-06-01	Sales	6500
63	Samantha	Martin	F	Missouri	1976-11-21	2006-01-01	Sales	7000
64	Nathan	Johnson	M	Michigan	1986-09-29	2003-08-01	Sales	5000
65	Michael	Smith	M	Connectic	1971-03-03	2004-03-01	Sales	8000
66	Madison	Davis	F	Florida	1982-11-08	2010-01-01	Sales	6500
67	Cameron	Jones	M	New York	1970-03-14	2003-07-01	Sales	5000
68	Ashley	Moore	F	New York	1980-05-09	2007-04-01	Sales	10000
69	Brandon	Wilson	M	Pennsylva	1981-05-15	2002-11-01	Sales	7000
70	Brandon	Johnson	M	New York	1976-07-26	2008-02-01	Sales	10000

一次读取出游标中的数据只是为了说明结果的结构，在实际的大数据计算中，数据往往不能一次读入内存，此时如果需要将多台服务器返回的数据合并后再计算时，就可以使用游标的归并计算。

归并计算时使用的游标序列，可以由各种游标构成，如文件对象游标、数据库游标或者远程游标等。

在使用游标读取数据时，有时需要将不同游标中的数据合并在一起使用，此时可以使用游标的无条件连接与结合。

- ***CS*.pjoin()**

游标序列 *CS* 中的各个游标 cs_i 横向无条件连接并返回成新的游标，长度以 cs_i 的最小记录数为准。横向连接即将多个游标的字段依次添加到结果序表中。

	A
1	=file("D:/files/txt/employee.txt")
2	=A1.cursor@t(NAME,SURNAME,STATE)
3	=demo.cursor("select NAME,ABBR,CAPITAL from STATES")
4	=[A2,A3].pjoin()
5	=A4.fetch()

A5 中读取数据的结果如下：

NAME	SURNAME	STATE	NAME	ABBR	CAPITAL
Rebecca	Moore	California	Alabama	AL	Montgomery
Ashley	Wilson	New York	Alaska	AK	Juneau
Rachel	Johnson	New Mexico	Arizona	AZ	Phoenix
Emily	Smith	Texas	Arkansas	AR	Little Rock
Ashley	Smith	Texas	California	CA	Sacramento
Matthew	Johnson	California	Colorado	CO	Denver

从结果中可以看出，简单横向连接只是拼接不同游标中的字段，即使有重名字段(如NAME)，在无条件连接时，也不会做处理。从结果游标返回的数据中，记录数以各个游标中的最小记录数为准。在使用无条件连接时，通常要求各个游标中的**记录数相同**。

- ***CS*.conj@x()**

将游标序列 *CS* 中的各个游标 cs_i 纵向无条件连接并返回成游标，数据结构以第一个游标 cs_1 为准。纵向连接即将多个游标的记录依次添加到结果序表中，通常要求所有游标的**字段数相同**。读取数据时，会依次从 cs_i 中读取。注意，为了和普通序列的结合函数 ***A*.conj()** 区别，游标序列的结合函数需要添加选项**@x**。

	A
1	=file("D:/files/txt/employee.txt")
2	=A1.cursor@t(NAME,SURNAME,STATE)
3	=demo.cursor("select NAME,ABBR,CAPITAL from STATES")
4	=[A2,A3].conj@x()
5	=A4.fetch()

A5 中读取数据的结果如下：

NAME	SURNAME	STATE
Nicole	Smith	South Carolina
Joseph	Smith	Pennsylvania
Alabama	AL	Montgomery
Alaska	AK	Juneau
Arizona	AZ	Phoenix
Arkansas	AR	Little Rock

从结果中可以看出，游标的结合只是将记录依次拼接在一起，数据结构由 cs_1 决定，cs_i 中读取到的数据会按位置依次填入 cs_1 的字段。

10.4.3 游标的连接

除了无条件连接以外，在集算器中还可以设定等值条件，将游标中的数据与其它序表或者游标中的数据关联起来。

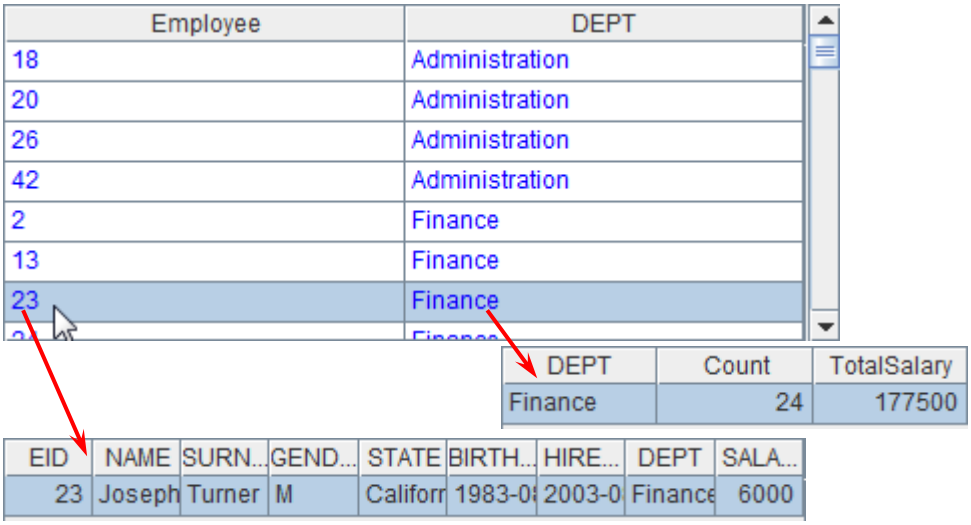
- **join@x(cs_i:F_i,x_j,…;…)**

以对应的字段或表达式 x_j,…值相同为条件连接游标 cs_i,…，产生以 F_i,…为字段的游标返回。读取时，F_i 取值为 cs_i 中读取到的记录。由于每个游标中的记录都是逐条读取的，因此要求每个游标中的记录都必须对表达式 x_i,…有序且顺序相同。与 **join()**类似，函数中同样可以使用**@f** 和**@1** 选项，在连接时计算全连接或左连接。

如，下面的例子中，将每个员工记录与他所在部门的记录连接：

	A	B	C
1	Administration	4	40000
2	Finance	24	177500
3	HR	19	138000
4	Marketing	99	733500
5	Production	91	663000
6	R&D	29	239000
7	Sales	187	1362500
8	Technology	47	344000
9	=create(DEPT,Count,TotalSalary).record([A1:C8])		
10	=[A1:A8].(file("D:/files/txt/"+~+".txt"))		
11	=A10.(~.cursor@t())		
12	=A11.merge@x(DEPT)		
13	=join@x1(A12:Employee,DEPT;A9.cursor():DEPT,DEPT)		
14	=A13.fetch()		

注意在 A13 执行连接前，需要保证 A12 和 A9 中游标的记录，对于部门均是有序的。在 A14 中可以看到从游标中一次读取出的数据：



在很多时候，我们只是需要根据两个或多个表中的列之间的关系，从这些表中将查询数据并展现出来，类似于 **SQL** 中的 **join** 语句，而不是想获得上例中记录构成的序表。此时，可以直接根据关联关系在游标中添加字段。

➢ ***cs*.join(*x*:…, *A*:*y*:…,*z*:*F*,…;*x*:…, *A*:*y*:…,*z*:*F*,…;…)**

将游标 *cs* 与排列 *A* 连接，连接条件是 *cs* 中记录计算表达式 *x*:…的结果与 *A* 中记录计算表达式 *y*:…的结果相等。其中 *y* 可以省略，省略时连接条件使用 *A* 的主键。连接后，在 *cs* 的记录中增加字段 *F*,…，其值为在 *A* 中计算表达式 *z* 的结果。使用这个函数时，还可以同时连接多个排列，多个排列的连接参数之间用 ；分隔。

❖ **@i**　当无法从 *A* 中找到匹配的外键时，则删除 *cs* 中的该条记录。

另外，连接函数 ***cs*.join()**中，用游标 *A* 代替排列，在计算时，会首先从游标 *A* 中读出全部记录生成序表，也就是说，在函数中使用游标 *A* 时，要能保证该游标中并非大数据，可以读入内存。

这个连接函数常用于大数据表，可以在获取游标 *cs* 中的记录时，根据条件关联其它序表或者游标。

	A
1	=file("D:/files/txt/employee.txt")
2	=A1.cursor@t()
3	=A2.new(EID,NAME+" "+SURNAME:FullName,STATE,GENDER)
4	=create(Code,Gender).record(["F","Female","M","Male"])
5	=demo.cursor("select * from STATES")
6	=A3.join(GENDER,A4:Code,Gender:FullGender;STATE,A5:NAME,ABBR:StateAbbr)
7	=A6.fetch()

其中，A3 从文件对象游标 A2 中计算出我们需要的字段；A6 中，用 A3 返回的游标连接 A4 中的序表与 A5 中的数据库游标，获取员工性别的全称和所在州的简称。在例子中，只是为了说明函数的用法，在 A7 中一次读取所有数据：

EID	FullName	STATE	GENDER	FullGender	StateAbbr
1	Rebecca Moore	California	F	Female	CA
2	Ashley Wilson	New York	F	Female	NY
3	Rachel Johnson	New Mexico	F	Female	NM
4	Emily Smith	Texas	F	Female	TX
5	Ashley Smith	Texas	F	Female	TX
6	Matthew Johnson	California	M	Male	CA

实际使用中 ***cs*.join()**返回的结果集往往是非常庞大的，可以每次从游标中读出部分记录。

10.4.4 游标的分组与汇总

- ***cs*.groupx(*x*)**

对游标 *cs* 中的记录做 **group@n()**式的分组，返回游标序列。由于大数据表中的数据无法一次读入内存，也就无法和普通序表一样分组计算，为此集算器专门提供了这个游标的分组函数。在分组时，要求表达式 *x* 的计算结果直接指定分组序号，执行分组后会生成多个文本数据临时文件记录各个分组的数据，并返回由这些文件的游标构成的序列。

	A	成员
1	=file("D:/files/txt/employee.txt")	com.raqsoft.dm.cursor.BFileCursor@1e7e4bb
2	=A1.cursor@t()	com.raqsoft.dm.cursor.BFileCursor@bf75e3
3	=A2.groupx(if(GENDER=="F",1,2))	

游标的分组汇总，类似于 **7.2.3 分组汇总**中的 ***A*.groups(*x*:*F*,...;*y*:*G*,...)**，需要使用**聚合分组汇总**。

- ***cs*.groups(*x*:*F*,…;*y*:*G*,…)**

用游标 *cs* 中的数据进行聚合分组汇总，返回结果序表。其中参数与 ***A*.groups()**类似，也同样可以使用**@o** 及**@n** 选项。聚合时，计算使用的表达式 *y* 与 ***A*.groups()**相同，只能用 **sum/count/max/min/topx** 等简单聚合函数。

游标的聚合分组汇总，可以处理大数据的分组汇总计算，使用方法与序表的聚合分组汇总类似。另外，也可以用于数据分为多个游标返回时，如并行计算中，由服务器返回多个远程游标时。

如，修改 **10.4.2 游标序列的归并与无条件连接**这一小节中的 dfx，使用返回的游标列表归并之后再汇总分组，计算各州男女员工的人数：

	A
1	[Finance,Sales,HR,Administration, Marketing,Production,R&D,Technology]
2	=A1.(file("D:/files/txt/"+~+".txt"))
3	=A2.(~.cursor@t())
4	=A3.conj@x().groups(STATE;count (GENDER=="M"):Male,count (GENDER=="F"):Female)

将返回的远程游标列表聚合分组汇总后，在 A4 中可以查看结果：

STATE	Male	Female
Maryland	3	1
Massachusetts	2	6
Michigan	16	12
Minnesota	1	2
Mississippi	4	0
Missouri	2	7

10.4.5 大数据游标的排序

- ***cs*.sortx(*x*…;*n*)**

将大数据量游标 *cs* 中的记录中的数据进行排序，将结果返回为游标。参数中的 *n* 是缓冲区大小，在创建游标时，每次从 *cs* 中读取 *n* 条记录进行排序，将每次的排序结果缓存到临时文件，循环执行上述操作。当从结果游标中读取数据时，将从所有临时文件的文件游标中归并读取。

	A
1	=file("D:/files/txt/employee.txt")
2	=A1.cursor@t()
3	=A2.sortx(STATE;50)
4	=A3.fetch()

EID	NAME	SURNA...	GEN...	STATE	BIR...	HIR...	DEPT	SAL...
102	Christian	Smith	M	Alabama	1972	2000	Sales	12000
282	Kayla	Davis	F	Alabama	1970	2006	Sales	6500
419	Sara	Davis	F	Alabama	1984	2004	Sales	5000
477	Lauren	Smith	F	Alabama	1975	2007	Marke	6500
100	Danielle	Smith	F	Arizona	1986	2000	Tech	7000

***cs*.sortx()**的使用方法和普通的 ***T*.sort()**相同，也会返回相同的结果。在对大数据量表的排序中，往往由于内存有限，无法将所有记录读出，或者不能容纳结果序表，此时就应该使用大数据排序函数 ***cs*.sortx()**。

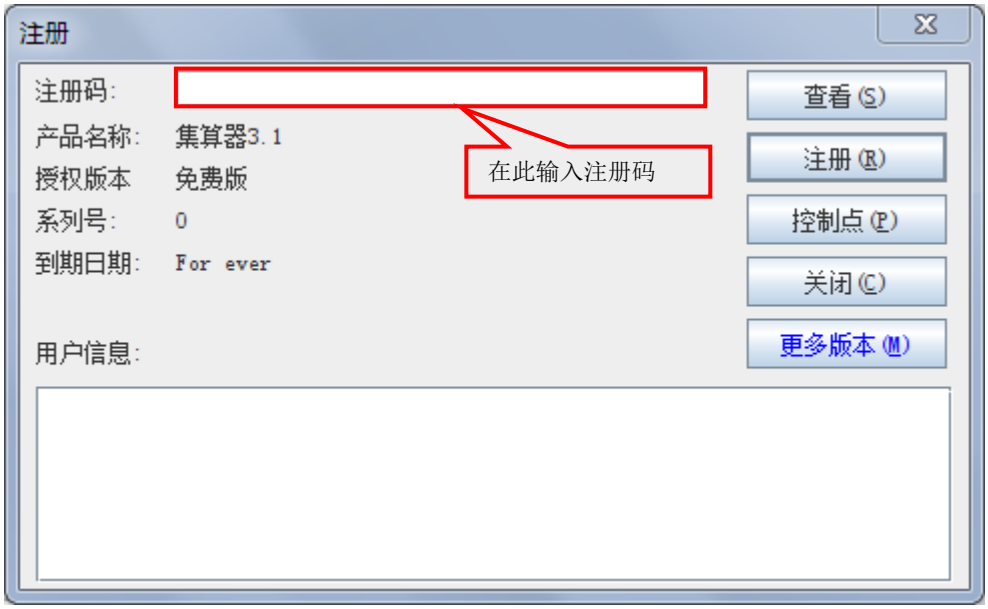
11. 授权

11.1 注册码

集算器的版本通过注册码来控制，各版本之间的区别及如何获得注册码，请参阅**官方网站**。

获取注册码后，可以在集算器中进行注册，操作步骤如下：

1) 打开集算器，执行 **帮助** 菜单中的 **注册** 按钮。



2) 输入注册码之后，点击**查看** 按钮检查注册码是否正确，如果注册码无误，则可以点击 **注册** 进行注册。免费版的注册码为空。

11.2 网格描述与单元格提示

11.2.1 网格描述

在集算器中，可以在网格文件中设置网格描述信息。

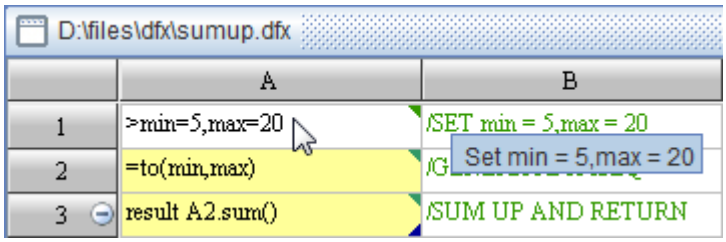
在菜单栏的**工具**选项中，点击**网格描述**按钮，可以弹出网格描述窗口，在窗口中查看或者设置网格描述。

11.2.2 单元格提示

在单元格中，可以添加提示信息，对格中的运算做出说明。

在选定的单元格上单击右键，在右键菜单中选择**单元格提示**，可以弹出单元格提示窗口，窗口中可以设置单元格提示。

如果单元格中设置了提示，在集算器中，该单元格的右上角会出现一个绿色的三角形标明。如果鼠标停留在设置了提示的单元格中，在鼠标指示的格子旁边会出现单元格中设置的提示信息：



为了说明，B 列中的单元格中填写了注释信息，需要注意的是，注释信息和单元格提示是不同的。

11.3 文件加密

11.3.1 权限

为防止未经许可查看或访问网格文件，集算器提供了网格文件加密功能，加密有如下

两层权限：

执行权：可以填写参数并执行网格。可以查看网格描述以及单元格提示，但不能修改这些信息。在只拥有执行权时，无法看到单元格中的任何字串以及网格变量。在运行后，可以看到运行后单元格的格值。

完全控制权：可以对网格文件进行任何操作。

11.3.2 设置密码

在菜单栏中的**工具**选项中，点击**网格密码**按钮，可以弹出网格密码设定窗口：

点击**完全控制权**，可以设定完全控制权密码。

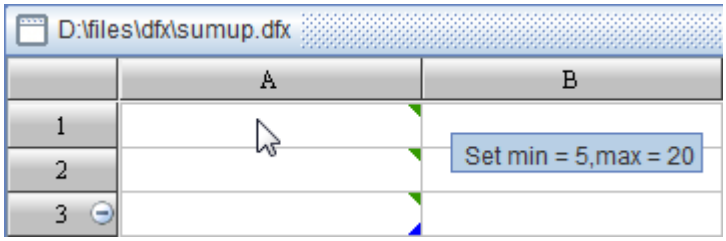
点击**执行权**，可以设定执行权密码。

注意：解除某个权限即用该权限密码进行解除即可。使用完全控制权密码打开文件后可以解除完全控制权和执行权密码，使用执行权打开文件则只能解除执行权密码。

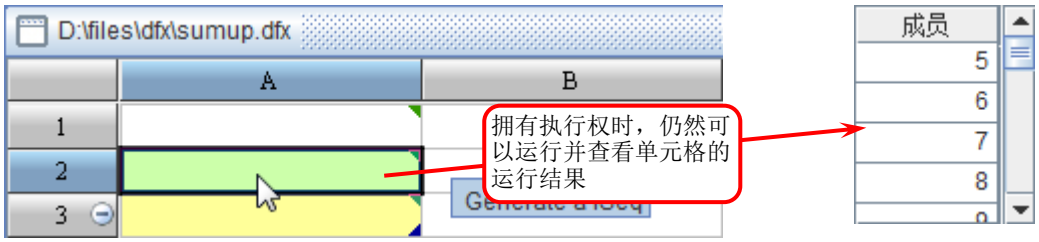
11.3.3 权限控制

如果网格文件设定了密码，在集算器中打开网格时，会弹出输入网格密码窗口，在窗口中输入密码，如果密码正确，则会按照密码的相应权限打开网格文件。当执行权与完全控制权的密码相同时，打开网格时将按照最高级别打开，也就是完全控制权。

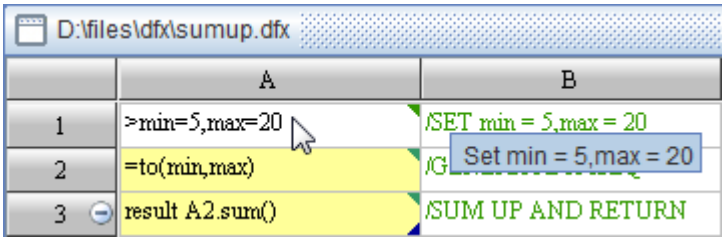
当只拥有**执行权**时，无法在网格中看到单元格字串，即使是单元格中的注释文字也无法看到，也无法查看网格中使用的网格变量，但是可以看到单元格提示信息：



此时可以正常执行网格进行计算，如果网格需要在计算前设置参数，也可以设定执行参数，执行后，可以查看运行后单元格的格值：



当拥有完全控制权时，可以查看网格文件的所有内容，允许对网格文件执行任意操作：


D:\files\dfx\sumup.dfx
A
B
1
>min=5,max=20
/SET min = 5,max = 20
2
=to(min,max)
Set min = 5,max = 20
3
result A2.sum()
/SUM UP AND RETURN