Ch3：DATA 步编程和数据处理：基础

程序是 SAS 语言的主要内容，是数据处理和分析的主要手段。数据步（DATA Step）和过程步（PRPC Step）是构成 SAS 程序的两种基本程序步。数据步整理（manipulate）数据，过程步分析数据并生成输出结果。过程步的功能还包括 SAS 文件管理。

3.1 SAS 程序

SAS 程序是按一定次序排列的 SAS 语句（statements）形成的文本，执行后完成特定的任务。SAS 语句由 SAS 单词（words）组成，SAS 单词也称为 SAS 标记（token），是能被 SAS 识别的最小语言单位。SAS 关键词和 SAS 名字是常用的 SAS 单词。

3.1.1 SAS 语句

由 SAS 关键词、SAS 名字、SAS 表达式组成的文本段，以分号;结尾。它要求 SAS 执行一个操作或给 SAS 系统提供信息。关键词相当于语句的"动词"，告诉 SAS 要执行的操作，除个别语句（如赋值语句、累加语句、空语句和注释语句）外，SAS 语句以关键词开始。例如，语句 data tt;中 data 是关键词，告诉 SAS 要生成一个新的数据集。SAS 名字相当于语句的"宾语"，可理解为关键词的作用对象。例如，语句 data tt 中的 tt 是 SAS 名字，告诉 SAS 生成数据集的名字为 work.tt。常用的 SAS 名字包括数据集名、变量名、过程名和输出/输出格式名等。

SAS 语句的书写需遵循如下规则

- 一个 SAS 语句可以跨行编写，多个 SAS 语句可以在同一行编写，以分号;作为语句结束标记和语句间的分隔。
- 语句中关键词大多以空格分隔。
- SAS 语句中的 SAS 关键词和 SAS 名字不区分大小写。
- SAS 语句中的 SAS 关键词和 SAS 名字只能是英文，要在英文输入状态下输入。尤其需要注意的是语句结束标志分号;，中文输入状态下的分号“；”不能被 SAS 识别，运行时提示语法错误，是国内 SAS 使用者容易犯的错误。

图 3.1 给出 DATA 步程序和 PROC 步程序中的语句及其成分。其中过程步中的 PROC 和 Print 都是关键词。在强化编辑器窗口中，关键词以蓝色显示。

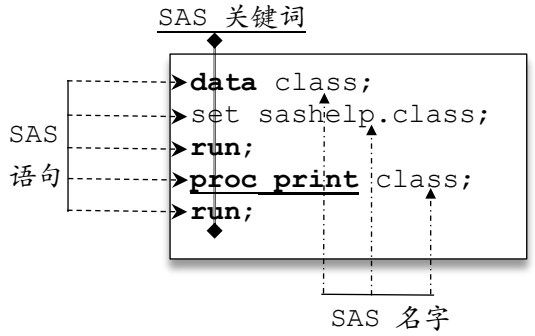


图 3.1 SAS 语句构成

3.1.2 SAS 表达式

运算符将运算对象连接起来形成表达式，正确的表达式经运算和取值后得出有效的结果。SAS 运算分为字符运算、数值运算、比较运算和逻辑运算，运算对象分为常量、变量和函数。

1. SAS 常量

SAS 常量（constant）也称为 SAS 常数，是固定的 SAS 值，分为字符常量、数值常量和日期-时间常量。字符常量是引号引起来的字符串（string），如'Tom'、'806'等，字符常量包含的字符区分大小写。也可以用双引号（""）来界定字符串。当单引号（双引号）是字符串的一部分时，需用双引号（单引号）来界定，例如"It's a pity"。数值型常量是一个数字，由正负号、数字和小数点构成，如 1、-2、33.5。SAS 中纯小数书写时可省略整数部分的 0，如-0.07 可写为-.07。当数值较大时，可采用科学计数法，如 0.5E-2 表示 0.5×10^{-2} 等。

日期常量用在引号引起来的日期后面添加字母"d"表示，如'03JAN2019'd。由于时间格式繁多，为避免混淆，时间常量中时间格式只能是"日日月月月年年"或者"日日月月月年年年年",即前一位或者两位数字表示日、中间三位英文大写字母表示月以及最后两位或者四位数字表示年。例如"03/05/2018"d 是错误的表示方式，没有输入格式作为引导，SAS 不知道如何读取其中的日子、月份、和年份。时间常量的表示方式是在引号引起来的时间后面添加字母"t"，如"09:27:25.23"t。日期时间常量的表示方式类似，将日期时间用引号引起来后添加字母"dt"，如'03JAN19 09:27:25.23'dt。日期和时间之间要空格隔开。

2． SAS 变量

变量是 SAS 程序中创建的数据容器（container）。变量由其属性确定，包括变量名、类型、宽度、输入/输出格式和标签等。DATA 步中的变量可分为 6 类

- 数据集变量。用 set 语句打开的 SAS 数据集中的数据集变量。
- 用 set 语句打开 SAS 数据集时定义的临时变量，如 end=v1、in=v2 定义临时变量 v1 和 v2。
- 用赋值语句定义的变量。
- 用 input 语句读入外部数据定义的变量。
- 用 SAS 关键词定义的变量，如关键词 length、attrib、format/informat 等定义的变量。
- DATA 步产生的自动变量（automatic variables）。很多自动变量的变量名用下划线开头和结尾，如_N_、_Error_，用户定义变量时尽量避免这种形式的变量名。

SAS 变量名的命名规则与 SAS 数据集和数据集变量的命名规则相同，变量引用中可以采用缩略列示，缩略列示方式与 SAS 数据集变量相同（见 Ch2-2.2）。

DATA 步编程中，有定义的变量（可理解为赋过值）才能被引用。

程序 3-1

程序段 1
```
data test;
x=y+1;
run;
```
程序段 2
```
data test;
x=y+1;
y=0;
run;
```
程序段 3
```
data test;
y=0;
x=y+1;
run;
```

程序段 1 和 2 都引用了没有定义的变量 y。引用没有定义的变量时，程序运行后日志窗口并不出现错误提示，而是给出信息（NOTE）。程序段 1 运行后给出的第一个 NOTE"变量 y 未被初始化"表示 y 没有定义，第二 NOTE"缺失值生成是对缺失值操作的结果"表示 y 的值缺

失导致参与的操作 x=y+1 结果（x 值）为缺失。程序段 2 运行后没有“变量 y 未被初始化”的 NOTE，在程序执行之前的编译阶段，y 已经存在，但语句 y=0;还没有执行，y 没有赋值，为缺失值，因此仍然会给出第二个 NOTE。

3． SAS 函数

函数是 SAS 编程语言的重要组件，可接受参数、执行计算或其他操作并返回函数值，函数值是数值型或者字符型，可用于赋值语句和表达式。SAS 函数由函数名和参数组成，参数可以是常量、变量和表达式，参数之间用逗号隔开。SAS 函数的一般形式为

函数名(参数 1<,参数 2,…,参数 n>)

如果参数为变量列表，则需要前置 of，变量列表可以采用缩略列示法。以求和函数 sum()为例，sum(x1,x2,x3)与 sum(of x1 x2 x3)同等效果,sum(x,of y1-y3,of z2-z6)与 sum(x,of y1-y3 z2-z6)同等效果。SAS9.4M5 中的函数多达几百个，种类繁多，功能齐全，很多运算可通过函数实现。

本节介绍一些常用函数，其它函数可以通过菜单“帮助”→“SAS 产品”→“Base SAS”→“SAS®9.4 Functions and Call Routines:Reference”查看，内容包括函数格式、功能说明和例子等。

❋ 数学函数和算术函数

数学函数包括常用的初等函数（幂、指数、对数、三角和反三角）和其它数学函数如 gamma 函数、贝塞尔函数等，对数函数有自然对数函数 log(x)、10 为底的对数函数 log10(x)和 2 为底的对数函数 log2(x)。算术函数包括算术运算函数（如和函数、平均值函数等）和其它数值运算函数（如取整数函数、求余函数等），表 3.1 给出常用的算术运算函数。例如 round(1234.567,1)和 int(35.4)的函数值分别为 1235 和 35。

表 3.1 给出常用的算术运算函数

函数	说明
Sum(a<,b,…,c>)	对参数 a、b、… ,c 求和
Mean(x1<,x2,…,xn>)	对参数 x1、x2、…、xn 求均值
Round(a<,d>)	对参数 a 按参数 d 进行舍入
Int(c)	对参数 c 取整数部分

❋ 字符函数

字符函数主要用于字符的操作，包括截取、替换、查询以及字符串属性（如长度）等。表 3.2 给出常用的字符函数，其中的 str1、str2、…、strn 表示字符参数，n1、n2 表示数值参数。例如 substr(‘SAS 课’,4,2)、scan(‘My dear’,1)和 cat(‘a’,’man’)的函数值分别为字符’课’、’My’和’aman’

表 3.2 常用字符函数

函数	说明
Length(str1)	计算 str1 的长度
Left(str1)	将 str1 头部空格移至尾部（左对齐）
Trim(str1)	删除 str1 尾部空格，空字符的返回值为空格
Index(str1,str2)	计算 str2 在 str1 中的位置，没有 str2 则取值 0
Substr(str1,n1<,n2>)	从 str1 的第 n1 个字符截取 n2 个字符，截至尾部则省略 n2
Scan(str1,n1<,str2>)	截取 str1 中第 n1 个单词（word），str2 为单词分隔符,默认空格

Cat(str1,str2,…,strn)	将字符串 str1 到 strn 首尾连接，连接时不删除各字符串的头尾空格

❋ **时间-日期函数**

时间-日期函数用来获取日期时间信息，计算给定日期中的日子、月份、年份、星期和给定时间中的时、分、秒和分秒。表 3.3 列举一些常用日期函数，其中 y 和 t 分别表示日期和时间参数，可以是变量、常量或者表达式。例如 day('12mar2019'd)、week('1may2019'd)和 minute('9:52:21't)的函数值分别为 12、3 和 52。

表 3.3 常用日期-时间函数

函数	说明
Date()	获取系统当前日期，没有自变量
Day(y)	获取日期 y 中的日子
Weekday(y)	获取日期 y 中日子是星期几（7 对应星期日）
Time()	获取系统当前时间，没有自变量
Hour(t)	获取时间 t 中的小时
Minute(t)	获取时间 t 中的分钟
Yrdif(a,b<,base>)	按 base 标准计算两个日期 a 和 b 间隔的年数

❋ **概率统计函数**

概率统计函数在统计分析和计量经济学分析中起着重要作用。Base SAS©9.4 提供了丰富的概率函数。

概率分布函数

随机变量ξ的概率分布函数定义为$F_\xi(x)=P(\xi\le x)$。假设检验中，检验统计量的值计算出来后，根据其概率分布（精确分布或者渐进分布）用概率分布函数计算检验的 p 值（p-value）。Base SAS©9.4 中有两种概率分布函数，具体函数和通用函数。

具体概率函数的函数名具有由前缀 "PROB"加分布名称构成（泊松分布除外），例如 PROBNORM(x)为正态分布的分布函数，PROBCHI(x,n)为χ^2分布的分布函数，n 为自由度参数，PROBNORM(1.96)的函数值为 0.975。表 3.4 给出具体概率函数，其中 x、y 表示实数参数，n1、n2 为正整数，nc 为非中心参数，r、a、b 为分布参数，参数可以是变量、常量或者表达式。

表 3.4 具体概率函数

函数	说明	参数值范围
PROBBETA(x,a,b)	贝塔分布函数	
PROBBNML(x,n1,n2)	二项分布函数	
PROBBNRM(x,y,r)	二元正态分布函数	
PROBCHI(x,n1<,nc>)	χ^2 分布函数	
PROBF(x,n1,n2<,nc>)	F 分布函数	
PROBGAM(x,a)	伽玛分布函数	
PROBNORM(x)	标准正态分布函数	
PROBNEGB(x,n1,n2)	负二项分布函数	
PROBT(x,<,nc>)	t 分布函数	
POISSON(n1,n2)	泊松分布函数	

通用概率函数用 CDF 作为统一的函数名，通过参数确定分布类型，不同分布类型对应不同的分布参数，一般形式为

CDF('分布名称',x<,参数1,参数2,…,参数k>)

例如：CDF('NORMAL',1.96)的函数值为 0.975，与 PROBNORM(1.96)的函数值相同。通用概率函数包含概率分布类型更多，除了表 3.4 给出的分布类型外，还有贝努里（BERNOULLI）、柯西（CAUCHY）、指数（EXPONENTIAL）、拉普拉斯（LAPLACE）、逻辑（LOGSTIC）、对数正态（LOGNORMAL）、混合正态（NORMALMIX）、帕累托（PARETO）、Tweedie（TWEEDIE）、均匀（UNIFORM）、沃尔德（逆高斯）（WALD）、威布尔（WEIBULL）分布等。

分位数函数

分为数（Quantile）函数是分布函数的逆函数，自变量为概率值 x，函数值为$F^{-1}(x) = inf\{y: y \geq x\}$。同概率分布函数一样，有两种分位数函数，具体函数和通用函数。具体函数给出了 6 种概率分布的分位数函数，例如 PROBIT()是标准正态分布的分位数函数。这里仅对通用分位数函数进行介绍，其一般形式如下：

Quantile('分布名称',<x,参数1,参数2,…,参数k>)

其中'分布名称'的种类和用法同通用概率分布函数，x 为概率值，参数 1-参数 k 为分布函数中的参数。例如 Quantile('NORMAL',0.975)、Quantile('t',0.975,5)和 Quantile('t',0.975,25)的值分别为 1.9599、2.5706 和 1.9882，表明 t 分布比正态分布有更厚的尾部。

概率密度函数

概率密度函数（Probability Density Function）只有通用函数，用于计算不同分布的概率密度函数值。一般形式为

PDF('分布名称',x<,参数1,参数2,…,参数k>)

适用的分布类型和有关参数设定与 CDF 函数相同。离散型随机变量的概率密度函数指其概率函数。

除概率分布函数和概率密度函数外，Base SAS ©9.4 还提供了生存函数（一般形式）SDF()、对数分布函数（一般形式）LOGCDF()、对数密度函数（一般形式）LOGPDF()和对数生存函数（一般形式）LOGSDF()，用于计算生存函数、分布函数对数、密度函数对数、生存函数的对数，十分方便。

统计函数

统计函数用来计算一组数值的描述统计量，例如均值函数 mean()、最大值函数 max()、最小值函数 min()、非缺失数据个数函数 N()、缺失数据个数函数 Nmissing()以及求和函数 sum()等。表 3.5 给出其它一些常用描述统计量函数，其中$x_1, x_2, \ldots, x_n$为数值型数据，$\bar{x} = n^{-1}\sum_{i=1}^{n} x_i$。

表 3.5 常用描述统计量函数

函数	说明	计算公式
VAR(x1,x2,… xn)	方差函数	$s^2 = (n-1)^{-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$
STD(x1,x2,… xn)	标准差	s
STDERR(of x1-xn)	均值标准差	$s/\sqrt{n}$
RANGE(of x1-xn)	极差	$\max(x_1, x_2, \ldots, x_n) - \min(x_1, x_2, \ldots, x_n)$
CSS(x1,x2,… xn)	校正平方和	$\sum_{i=1}^{n}(x_i - \bar{x})^2 = (n-1)s^2$
USS(x1,x2,… xn)	未校正平方和	$\sum_{i=1}^{n} x_i^2$

SKEWNESS(of x1-xn)	偏度	$\frac{n}{(n-1)(n-2)}\frac{\sum_{i=1}^{n}(x_i-\bar{x})^3}{s^3}$
KURTOSIS(of x1-xn)	峰度	$\frac{n(n+1)}{(n-1)(n-2)(n-3)}\frac{\sum_{i=1}^{n}(x_i-\bar{x})^4}{s^4}-\frac{3(n-1)^2}{(n-2)(n-3)}$ ①

需要注意，统计函数计算时是按非缺失值自变量计算的。例如 sum(4,9,3,8,.)=24，mean(4,9,3,8,.)=6。

随机数函数

随机数（random number）函数从给定分布中产生随机数，随机数可看做从分布中抽取的简单随机样本（独立同分布样本）。随机数函数也称为随机数发生器（generator），分为具体随机数函数和一般随机数函数。具体随机数函数的函数名以 RAN 为前缀后面是分布函数名的简称或者缩写，例如 RANNOR 为正态分布随机函数的函数名。不同的随机函数具有不同的参数值。表 3.6 列出常用具体随机函数，其中 s 为产生随机数需要的种子（seed），取 1~(2^31-1)范围内正整数，如果取值为负，则以系统日期中的日子（day）作为种子。

表 3.6 随机数函数

函数	说明	参数值范围
RANNOR(s)	正态分布随机数函数	标准正态分布
RANBIN(s,n1,p)	二项分布随机数函数	n1>0,0<p<1
RANCAU(s)	柯西分布随机数函数	
RANEXP(s)	指数分布随机数函数	分布函数参数$\lambda=1$
RANGAM(s,a)	伽马分布随机函数	a>0
RANPOI(s,n1)	泊松分布随机函数	n1≥0
RANUNI(s)	均匀分布随机函数	(0,1)上均匀分布

一般随机函数的函数名为 RAND，通过第一个参数'分布名称'确定随机数抽样的概率分布类型，通过其它参数确定概率分布函数。例如 RAND('BETA',s,a,b)和 RAND('F',s,n1,n2）为参数为 a、b 的贝塔分布和参数为 n1、n2 的 F 分布的随机数函数。一般随机数函数的形式为

RAND('分布名称',s<,参数 1,参数 2,…,参数 k>)

其中分布函数名共有 29 个，除了常见的正态分布（NORMAL）、t 分布（T）、χ^2分布（CHISQUARE）之外，还有戈珀兹分布（GOMPERTZ）、龚贝尔分布（GUMBEL）、尔朗分布（ERLANG）和极值分布（EXTRVALUE）等重要的分布。用一般随机函数能够从更多种类的分布中生成随机数。生成随机数对 Monte Carlo 模拟和贝叶斯分析都很重要，除随机函数外，例程（routine）调用也能产生随机数，参考例程调用。

4． SAS 运算和运算符

SAS 运算包括算术运算、比较运算、逻辑运算与字符运算。运算通过运算符体现，运算符将运算对象连接形成表达式。

❋ 算术运算符

表 3.7 算术运算符

① 从定义看出，函数 Kurtosis( )计算的实际上是超额峰度（excess kurtosis），即实际峰度超过正态分布峰度（3）的部分。

符号	+	−	*	/	**
含义	加	减	乘	除	乘方

算术运算是最为常用的运算，除乘方运算符可能与其它软件有区别外，运算符都采用相同的符号。X^{-y}表示为 x**(-y)。

❋ **比较运算符**

表 3.7 比较运算符

操作符	LT	GT	EQ	LE	GE	NE
符号	<	>	=	<=	>=	^=
含义	小于	大于	等于	小于等于	大于等于	不等于

也可以用~=、=<和=>作为不等于、小于等于和大于等于的运算符。比较运算的结果为逻辑结果，用数值 1 表示真，0 表示假。比较运算的对象须是同一类数据。如果比较的对象中一个是数值，一个是字符，则比较时要么报错，要么系统将字符转换为数值后进行比较，并在 log 窗口给出一个信息提示：字符变量××被转换为数值。

字符串比较从左到右逐个按 ACCII 码排列顺序进行比较，如'GARY'>'JONES,C.'的比较结果为 0（假），而'gary'>'GARY'的结果为 1（真）。两个不等长度的字符串比较时，在短字符串后面添加空格进行比较，空格的 ASCII 码值是 32。缺失值比任何字符都小。

SAS 支持把变量写在中间，前后都是运算符的比较运算表达式写法。如 7<=x<=10、'Johan'<name<='John'等。

Max 和 Min

Max 用来对两个量进行大小比较并取大。例如 Max（1,3）的结果为 3。也可以对变量大小进行比较，比较的内容是变量当前值，如果 x>y，则 Max(x，y)为 x 的当前值。Min 比较两个量的大小并取小，使用方法和 Max 相同。

可以用符号<>和><代替 Max 和 Min。但在 where 语句和 where 表达式中<>等同于 NE（不等于），而不是 Max。

集合比较运算符 in

集合运算符 in 是 SAS 特有的比较运算符，用于一个值（常数值或者变量值）是否在给出的值列表中。列表中给出的值必须是常数、缺失值或者遍历器（数组情形），用逗号隔开，当列表中的值是连续的正整数时，可以用 M:N 形式的缩略列示法。例如语句

```
if x in (1,2,3,4,6);
if x in (1:4,6);
if name in ('Alice','Jack','Tom');
if 3 in (0.4,5,7,10);
```

中都是 in 的正确用法。

in 前置逻辑运算符 not 可以取反，实现比较中的排除操作。例如语句

```
if x not in ('A','B');
```

的比较结果当 x 取值不是字符'A'或者'B'时为真。

不等长字符比较修饰符：

在字符比较中，如果参与比较的两个字符串（变量或者常量）长度不相等，在运算符后添加:，可以将长的字符从左到右截取到和短字符相同长度后再进行比较。例如

```
if name=:'S';
```

将变量 name 当前取值的首字母和字符'S'比较，而

```
if stkcd=:'600';
```

将股票代码的前 3 位和字符'600'比较。修饰符:也可加在其它比较运算符后面，例如语句

```
if lastname>=:'Joh';
```

中的比较运算也是对 lastname 的前 3 个字母进行的。

比较运算的结果是数值，可以用在赋值语句中。如：

```
c=5*(x<y)+12*(x>=y);
```

❋ **逻辑运算符**

表 3.7 逻辑运算符

操作符	AND	OR	NOT
符号	&	|	^
含义	和	或	非

SAS 程序中 0 和缺失值被识别为假，非 0 值的非缺失数值都识别为真，单个数值变量或者常量可以参与逻辑运算。例如

```
if 0 then delete;/*不会执行 delete 语句*/
if 1 or (a<=b);/*条件总是满足*/
```

❋ **字符运算符||**

字符连接符||将前后的两个字符串连接成一个字符串。例如'I"m'||'a'||'student'的运算结果为字符串'I'm a student'。注意由于空格是字符串的一部分，连接时予以保留。||是唯一的字符运算符，其它字符操作通过字符函数实现。函数 cat()也可以实现字符连接。

除了以上的运算符外，括号也是常用的运算符，其功能是改变各种运算的优先次序。SAS 将函数作为运算符（operator）。大多数情况下常量的函数还是常量，而变量的函数仍然是变量，因此表达式中的函数也可归为常量和变量。

5． where 表达式

出现在 where 语句中或者数据集选项 where=()中，用于 DATA 步或者 PROC 步中 SAS 选择数据集观测时的条件设定。where 表达式中除可使用通用比较运算符外，还有其专有的运算符和表达方式，设定条件时更为灵活。

❋ **between and**

在比较运算中设定取值范围，范围的边界既可以是常量，也可以是包含变量的表达式。例如

```
where taxes between salary*0.30 and salary*0.50;
```

between-and 运算符前置 not 可实现取反。例如语句

```
where date not between '01JAN2013'd and '01JAN2015'd;
```

将变量 date 值在'01JAN2013'd 和'01JAN2015'd 之间的观测排除在外。

❋ **like**

用于字符比较运算。在 like 后面的字符串常量中添加通配符（wildcard）可以实现更为灵活的字符匹配比较。下划线_通配符表示一位任意字符，百分号%通配符则表示任意位任意字符。如果 name 变量有 5 个观测值，分别为：

'Diana'、'Dinae' 、'Dianna'、'Dianthus'和'Dyan'，

在语句 where name like …;中采用不同的通配符得出符合匹配条件的观测（名字）为

统配符：	'D_an'	'D_an_'	'D_an__'	'D_an%'
名 字：	'Dyan'	'Diana'、Diane	'Dianna'	全部

like 运算符前置 not 实现取反，例如

```
where name not like 'D_an_';
```

的匹配结果是'Dianna'、'Dianthus'和'Dyan'。

如果字符和通配符相同，可用/和关键词 escape 避免统配。例如

```
where name like 'a/_b' escape '/';
```

匹配时不把字符中的_作为通配符，如果 name 有'axb'和'a_b'两个值，匹配结果为'a_b'，而语句

```
where name like 'a_b';
```

的匹配结果则是'axb'和'a_b'。

❋ **=***

进行"听起来像是"（sound like）的字符比较。例如

```
where name =* 'Smith';
```

会将 name 取值为'Schmitt'、 'Smith'、 'Smithson'、 'Smitt'、 'Smythe'的观测挑选出来。运算符 not=*进行"听起来不像是"的字符比较。

运算符=*和 not=*只适用于英语，不适用其它语种。

❋ **？(contains)**

？或者 contain 运算符用于判断一个字符串中是否包含另一个字符串。例如

```
where name ? "罗";
```

当两个对象都是字符变量时，对变量的当前值进行比较。例如

```
where name1 ? name2;
```

运算符？和 contains 都能通过前置 not 取反。例如语句

```
where company not contain 'Bay';
```

❋ **is missing(null)**

is missing 和 is null 以当前取值是否为缺失值进行比较运算，如果当前取值为（非）缺失值，则比较结果为真（假）。例如

```
where stkcd is missing;
```

missing 和 null 都可以前置 not 取反。例如

```
where salary is not missing;
```

❋ **same and**

same-and 算符用于给前面已经出现过的 where 语句添加比较条件。一般形式

```
where 表达式 1;
 语句 1;
 …
 语句 n;
where same and 表达式 2;
```

语句 1 和语句 n 之间（包括语句 1 和语句 2）不再有 where 语句。当需要对参与程序操作的观测值施加更多条件时，不需要重复输入以前的 wehere 语句，采用 same-and 添加即可。same and 运算符常用在 Proc 步中，例如程序 3-1

```
proc reg data=sashelp.class;
model weight=age height;
where age>11;
run;
where same and sex='男';
run;
```

Max 和 Min

where 表达式中不支持运算符><，运算符<>则等价于 NE。常将 Max 和 Min 的运算表达式用括号括住以增加程序可读性。例如

```
where x=(a max b);
```

where 表达式的逻辑运算

可以用逻辑运算符 and、not 和 or 将 where 表达式连接，形成更复杂和更为灵活的 where 表达式。一个表达式包含多个逻辑运算符时，运算的优先级为 not→and→or。添加括号可以改变运算次序且可读性强。例如

```
where skill not eq 'java' or years not eq 4;
```

等价于

```
where (skill not eq 'java') or (year not eq 4);
```

3.1.3 例程调用语句

例程是例行程序（routine）的简称,也称子程序，类似于函数，但不返回任何值，也不能用于赋值语句和表达式的运算，只能通过 call 语句调用。通过调用例程来定义变量并给变量赋值，很多函数都有相对应的功能类似的例程，但例程比函数功能更为强大。例如生成标准正态分布随机数的随机数函数和例程具有相同的名字 RANNORM，但使用方法不同。例如语句

```
x=RANNOR(2346);
```

和语句

```
call RANNOR(2346,x);
```

都生成一个标准正态分布随机数，但例程对随机数流的控制更强，可以完全重复生成同一个随机数序列。在 SAS 帮助系统中，例程和函数是放在一起的，找到函数的帮助即可找到例程的帮助。

3.1.4 空语句

空语句是只包含语句结束标志;的语句。在 data 步中空语句和 datalines 语句配合使用，用来表示程序中内嵌数据行的结束。如果数据行的开始语句用 datalines4，此时的空语句要用四个语句结束标志，即;;;;。

3.1.5 注释语句和程序注释

为了对程序代码进行注释而增强可读性，SAS 语言设立了注释语句。根据注释方式不同，注释语句有两种形式。

（1）块注释：语句以/*开始，以*/结束，注释内容可包含;以及任何长度的文本，也可以跨行。

（2）行注释：语句以*开始，以;结束。

行注释语句以;结束，容易和所注释的执行语句混淆，故一般情况下建议使用块注释语句。例如

程序 3-2

```
data sasuser.test1;
input stkcd $ date yymmdd10. clp 6.2;/*股票代码、交易日期和收盘价
*/
/*cards以下为数据行*/
datalines;
600837 2019/03/13 6.28
600837 2019/03/14 6.30
;/*null statement*/
run;
```

3.1.6 语句作用范围

SAS 语句按作用范围分为 DATA 步语句、PROC 步语句和全局语句。DATA（PROC）步语句是只在 DATA（PROC）步程序中有效的语句，全局语句则在所有程序中有效。例如 input 语句为 DATA 步语句，而空语句和注释语句为全局语句。

SAS 程序边界

SAS 程序按步划分，一个步是一个完整的程序段，用边界来界定，SAS 按步边界进行独立编译和执行。DATA 步程序的开始边界是 DATA 语句，结束边界为下一个 DATA 步或者 PROC 步的开始，如果没有后续的 DATA 步或者 PROC 步，以语句 run;作为结束边界。没有结束边界的 DATA 步将不被执行。DATA 步的一般语法格式及其边界如图 3.2

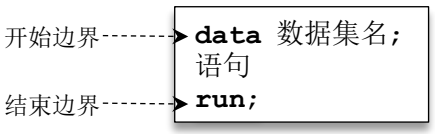


图 3.2 DATA 步边界

3.2 DATA 步原理

DATA 步用来进行数据的加工处理，生成需要的数据集。既可以在不读入数据的情况下用赋值语句定义变量生成 SAS 数据集，也可通过读入已有数据通过编辑生成 SAS 数据集，读入的数据既可以是 SAS 数据集，也可以是外部数据。

3.2.1 DATA 步的几个重要概念

1．程序数据向量 PDV—数据处理“车间”

程序数据向量（Program Data Vector）是 SAS 为 DATA 步数据处理开辟的内存区域，DATA 步的数据处理都在 PDV 中进行。每个 DATA 步都有自己的 PDV 用以存放步中的变量及其取值，包括自动变量、SAS 数据集变量和新定义的变量。

2．输入缓冲区 IB—数据预处理“车间”

输入缓冲区（Input Buffer）是 SAS 为 DATA 步读入外部数据开辟的内存区域，用于外部数据的“预处理”。外部数据需要通过输入格式设定等操作的预处理才能为 SAS“认识”，经预处理后的数据才能从 IB 读入 PDV 进行处理。如果 DATA 步读入的数据来自 SAS 数据集，SAS 将数据集观测中的数据直接复制到 PDV 中，不需要输入 IB。

3．DATA 步的内置循环

DATA 步处理数据以观测为单位，采用逐观测处理的方式。从 DATA 语句开头到结束边界的所有语句都只对当前观测（PDV 中的观测）进行操作，当前观测处理完毕后对下一条观测重复同样的操作，直到所有观测处理完毕。DATA 步处理方式是一种循环流程，这种循环是 DATA 自动执行的，是内置循环。自动变量_N_对循环次数进行计数，自动变量_Error_记录当前观测被处理时是否发生错误，取值 1（0）表示（不）发生错误。

4．输入、输出数据集

输出数据集指用 DATA 步生成的数据集，数据集名位于 DATA 语句中关键词 DATA 之后；输入数据集指用 set 语句打开并从中读取观测到 PDV 中的 SAS 数据集，数据集名位于 set 关键词之后。输入和输出数据集名都采用两级文件名。

5．DATA 步的编译和执行

同大多数编程语言一样，SAS 程序的执行分为编译和执行两个阶段，DATA 步程序也是先编译后执行。

❋ **DATA 步的编译**

检查程序语法，并为程序执行准备环境。DATA 步编译阶段完成的工作如下：

（1）语法检查和变量标识。扫描 DATA 步每一行语句的每个单词，执行语法检查，检查是否有语法错误。编译器也会标识 DATA 步中每个变量的名称、类型和长度信息，判断是否需要为后续变量引用进行数据类型转换。

（2）建立 PDV，为 DATA 步每个变量分配 PDV 单元，变量的值为缺失值。变量包括输入数据集中允许的变量、input 语句定义的变量、赋值语句定义的变量、自动变量以及其他语句定义的变量。

（3）建立输入缓冲区。当在 DATA 步中扫描到 input 语句时，为外部数据的输入建立输入缓冲区。

（4）生成输出数据集描述信息。给每个要生成的 SAS 数据集创建描述信息（元数据），包括数据集属性和变量属性信息。

如果在编译阶段发现语法错误（syntax error），SAS 尝试按自己的理解纠正错误，如果能够纠正，则进入运行阶段，并在信息窗口发出警告（warning）对所做纠正进行说明，如果无法纠正错误，则 DATA 步不进入执行阶段，并在信息窗口发出信息，提示语法错误。

❋ **DATA 步的执行**

通过编译的 DATA 步进入执行阶段。SAS 按如下步骤执行程序

(1) SAS 从 DATA 语句（开始边界）开始执行，顺次执行步中的各个语句。给 PDV 中变量赋值。首先为自动变量_N_赋值 1，_Error_赋值 0。如果有 set 语句打开的输入数据集，则将输入数据集中指定的观测值（默认第一个观测）读入 PDV，赋值给对应的数据集变量。如果有 input 语句，则将外部数据首先读入输入缓冲区进行整理，然后读入 PDV 赋值给对应的变量。遇到赋值语句或者例程调用（call），则进行有关的计算并将计算结果赋值给变量。

（2）如果出现错误，则停止 DATA 步执行，_ERROR_变量赋值 1，log 窗口首先在发生执行错误语句的下面给出错误信息（ERROR:），然后在 DATA 步程序的后面给出提示:“NOTE: 由于出错，SAS 系统停止处理该步”。由此看出，自动变量_ERROR_记录的是是否发生执行错误而不是语法错误。

（3）遇到步结束边界，输出 PDV 中各变量值到输出数据集中，在输出数据集中生成一条观测，完成 DATA 步一次内置循环。这种输出操作不需要执行特定的语句，自动执行，称为隐含输出（implicit output）。

（4）返回 DATA 语句开始下一次内置循环。此时 PDV 中的变量中，自动变量_N_的值增加 1，输入数据集变量的值保持上次循环取值直到被本次循环新值覆盖，而赋值语句、input 语句等定义的变量值则初始化为缺失值，直到本次循环被赋以新值。

注意：(1)没有结束边界的 DATA 步不被执行；（2）即使编译阶段发现语法错误而不被执行，DATA 步也会生成输出数据集，输出数据集中有 0 个观测。

6． 声明语句与执行语句

声明（declarative）语句在编译环节起作用，向 SAS 提供必要的信息。DATA 步开始边界的 DATA 语句、设置输入数据集观测过滤条件 where 语句、标明其下面为数据行的 datalines 语句以及改变变量输入输出格式的 informat/format 语句等，是声明语句。执行（executable）语句在执行环节起作用，在 DATA 步每次内置循环中产生一些动作。读入外部数据的 input 语句、条件成立执行后续语句的 if 语句、打开 SAS 集读取观测的 set 语句以及停止当前 DATA 步执行的 stop 语句等，是执行语句。DATA 步语句按声明和执行的分类可参考“帮助”→SAS 产品→Base SAS→SAS®9.4 DATA step statements: Reference→About DATA step statements→DATA step statements。

下面给出 DATA 步运行的三种情形，更为清晰理解 DATA 步运行机制，

3.2.2 DATA 步流程—定义变量生成数据集

定义变量生成数据集的 DATA 步较为简单。以程序 3-3 为例说明 DATA 步的数据处理流程。

程序 3-3

```
data test1;
x=0;
y=x+1;
put _all_;
run;
```

编辑阶段：

(1) 创建 PDV,内置变量_N_、_ERROR_、x 和 y，并取缺失值；

(2) 生成输出数据集描述信息：数据集名 work.test1、变量 x、y 属性等。

执行阶段：

(1) 给 PDV 中自动变量赋值：_N_=1、_ERROR_=0，此时 PDV 中 x 和 y 取缺失值；

(2) 执行语句 x=0，给 PDV 中 x 赋值 0，此时 y 取缺失值；

(3) 执行语句 y=x+1，先计算 x+1，然后赋值 y，y=2；

(4) 遇到 DATA 步结束边界 run;语句,将当前 PDV 中变量值输出到数据集 work.test1，形成一条观测，DATA 步结束。

语句 put _all_;将当前 PDV 内容输出到信息窗口，据此观察各个执行环节 PDV 内变量取值的变化。

```
x=0 y=1 _ERROR_=0 _N_=1
NOTE: 数据集 WORK.TEST1 有 1 个观测和 2 个变量。
NOTE: “DATA 语句”所用时间（总处理时间）:
      实际时间          0.54 秒
      CPU 时间          0.01 秒
```

图给 3.3 给出了处理流程示意图。

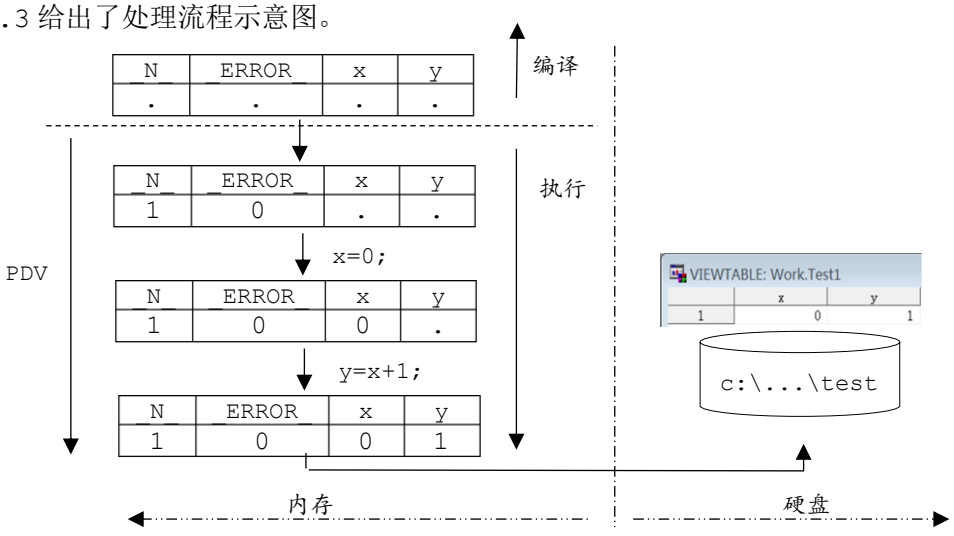


图 3.3 DATA 步处理流程

提交程序段

程序 3-4

```
data test1;
put _all_;
x=0;
```

```
put _all_;
y=x+1;
put _all_;
run;
```

运行后在信息窗口显示如下信息：

```
x=. y=. _ERROR_=0 _N_=1
x=0 y=. _ERROR_=0 _N_=1
x=0 y=1 _ERROR_=0 _N_=1
```

DATA 内置循环一次，循环次数计数器_N_=1。

4 DATA 步流程——以输入数据集为基础生成数据集

读入已有 SAS 数据集并进行编辑是最为常用的数据集生成方法。由于 SAS 数据的数据格式为 SAS 系统所熟悉，读入操作相对简单。通过 set 语句读入输入数据集观测，程序的一般语法格式为

```
data name1;
 set name2;
 语句
run;
```

set 语句的功能是打开数据集 name2 并从中逐条读取观测到 PDV 中。以程序 3-5 为例说明输入数据为 SAS 数据集情形下 DATA 步的数据处理流程。为简单起见，输入数据集采用 sashelp.class 编辑后得到的数据集 sashelp.s，包含变量 name、sex、age 和 h，有 5 条观测。

程序 3-5

```
data class1;
set sashelp.s;
h1=h*.025;
;
run;
```

编译阶段：

(1) 创建 PDV,内置变量_N_、_ERROR_、name、sex、age、h 和 h1,变量值缺失；

(2) 为 set 语句建立输入数据指针，在执行阶段引导 SAS 读取数据指针指定的观测；

(3) 生成输出数据集描述信息：数据集名 sashelp.class1 及其变量属性等。

执行阶段：

(1)给 PDV 中自动变量赋值：_N_=1、ERROR=0，set 语句的数据指针指向输入数据集 sashelp.s 的第一条观测,SAS 将第一条观测值读入 PDV 赋值给对应的变量 name、sex、age 和 h，此时 h1 取缺失值；

(2)进行运算 h*0.025，将运算结果赋值为 h1；

(3)执行一条空语句（不做任何操作）；

(4) 遇到 DATA 步结束边界 run;语句，将当前 PDV 中变量值输出到数据集 sashelp.class1 形成第一条观测。

(5)返回 DATA 语句开始下一轮内置循环。此时 PDV 中的变量取值情况为：_N_值增加 1，赋值语句定义的变量 h1 的值置为缺失，数据集变量 name、sex、age 和 h 的值保留。

(6)数据指针指向 sashelp.s 的第二条观测，SAS 将第二条观测值读入 PDV 覆盖对应变量 name、sex、age 和 h 的值，此时 h1 取缺失值。

(6)运算 h*0.025，将运算结果赋值为 h1；

(7)遇到 run 语句，将当前 PDV 中变量值输出到数据集 sashelp.class1 形成第二条观测。

(8)重复以上内置循环。当处理完 sashelp.s 第 5 条观测后，返回 DATA 步开头进行下一次（第 6 次）内置循环，数据指针到达输入数据集底部发现文件底部标志（end-of-file indicator），已没有可以输入的观测，DATA 步内置循环结束，输出数据集 sashelp.class1 生成。

图给 3.4 是 DATA 执行阶段处理流程示意图。

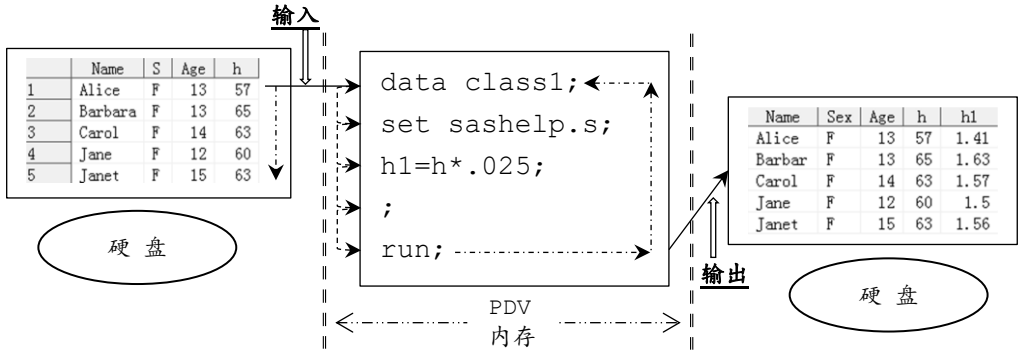


图 3.4 DATA 步执行处理流程

需要注意的是：

1． DATA 步是逐条处理观测的：一次读入一条输入数据集观测，该观测经历数据步所有语句的作用和操作，输出后形成一条输出数据集的观测；

2． set 语句的数据指针在默认情况下是按观测号顺序读取观测，但可以用选项改变读取顺序。一个 DATA 步可以有多个 set 语句，每个 set 语句都有自己的数据指针。一个 set 语句如果打开的是多个输入数据集，也只有一个数据指针。任何一个 set 语句的数据指针指向其打开文件（可以是多个）的文件底部时，DATA 步结束执行。

3． DATA 步语句对读入 PDV 内的数据按其在步中的顺序依次起作用，但可以用控制语句改变语句执行流程。

4． 用 SAS 输入数据集生成新数据时，DATA 步有输入和输出两个环节实现硬盘文件和内存 PDV 的数据传输。在对观测进行操作和筛选的 SAS 语句和数据集选项中，有些是在输入环节起作用，而另一些是在输出环节起作用的。例如输入数据集选项在输入环节起作用，对输入数据集读入 PDV 的观测和变量进行筛选（如 where=、keep=等）或者操作（rename=），而输出数据集选项则在输出环节起作用，对 PDV 写入到输出数据集的观测和变量进行筛选。

5． 自动变量_N_的值是 DATA 步内置循环的次数，不是输入数据集读入 PDV 的观测个数，后者与 set 语句有关。

6． 自动变量_N_最后一个取值是 6 而不是 5。可以通过在 set 语句前添加语句 put _all_ 观察_N_的值。

5 DATA 步流程—读入外部数据生成数据集

DATA 步用 input 语句将外部数据读入，经处理后生成 SAS 数据集。外部数据不是 SAS 数据集格式，读取时需要用输入格式进行引导，在读入 PDV 被 DATA 步语句操作之前，需要先读入缓冲区进行预处理。外部数据分为外部文件数据和程序内嵌数据行（也称流式数据行），程序内嵌数据是程序中位于 DATA 步中数据行形成的数据块，以语句 datalines;和空语句;作为开始边界和结束边界。input 语句定义变量，规定各变量读取内嵌数据行的方式和格式，由 datalines;语句引导将内嵌数据行读入输入缓冲区，经预处理后送入 PDV 进行 DATA 步内置循环。下面以程序 3-6 为例说明 DATA 步流程，input 语句中变量名后面的$符号，规定该变量读取数据时按字符读入，没有$符号则按数值读入。

程序 3-6

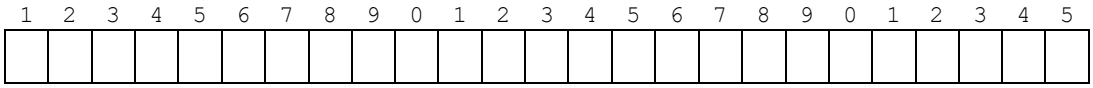
```
data score;
input name $ score1 score2;
Total=score1+score2;
datalines;
Zhangsan 87   75
Lisi 90 64
;
run;
```

DATA 步流程

编译阶段：

(1) 创建输入缓冲区 IB 和 PDV,在 PDV 内置变量_N_、_ERROR_、name、sore1、score2 和 Total，变量取缺失值，为 input 建立数据指针（pointer）;

IB



PDV

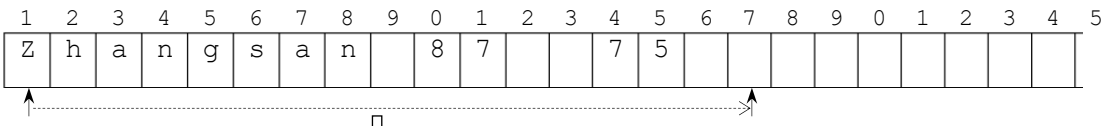
_N_	_ERROR_	name	Score1	Score2	Total
.	.		.	.	.

(2) 生成输出数据集描述信息：数据集属性及其变量属性等。

执行阶段：

(1)给 PDV 中自动变量赋值：_N_=1、ERROR=0；input 语句数据指针（↑）指向 IB 开始位置，按行依次读取数据，同一行内按分隔符（默认为空格）分隔数据。首先读入 datalines 语句下面的第一行数据，数据指针指向 IB 开始位置，并从左至右移动，依次读取数据直至行尾。完成第一行读取后将数据写入 PDV 依次赋值给 input 语句中定义的变量。

IB



PDV

_N_	_ERROR_	Name	Score1	Score2	Total
1	0	Zhangsan	87	75	.

(2)PDV 中变量经历 DATA 步的各个语句,完成第一轮内置循环:首先计算 score1+score2 并赋值给 Total，然后遇到结束边界 run 语句将当前 PDV 中各变量值输出形成数据集 score 的第一条观测。

(3)返回到 DATA 开头，input 语句定义的变量的值重置为缺失值。读取第二行数据到 IB 并赋值给 PDV 变量，开始第二轮内置循环。

(4)第三次循环开始后，发现已没有输入数据行，DATA 步结束循环。

可以在程序的不同地方添加 put _all_语句，观测 PDV 中变量在不同阶段的取值，见程序 3-7。

程序 3-7

```
data score;
input name $ score1 score2;
Total=score1+score2;
datalines;
```

```
Zhangsan 87   75
Lisi 90 64
;
run;
```

信息窗口显示如下信息：

```
name=  score1=. score2=. Total=. _ERROR_=0 _N_=1
name=Zhangsan score1=87 score2=75 Total=162 _ERROR_=0 _N_=1
name=  score1=. score2=. Total=. _ERROR_=0 _N_=2
name=Lisi score1=90 score2=64 Total=154 _ERROR_=0 _N_=2
name=  score1=. score2=. Total=. _ERROR_=0 _N_=3
```

最后一行中_N_=3 表明执行阶段第(4)步的执行:首先执行第一个 put 语句，此时的_N_为 2，input 读取不到数据行，DATA 步结束，第二个 put 语句未执行，data 进行了第 3 次内置循环，此时_N_=3。

3.3 DATA 步—数据操作

3.3.1 DATA 语句—DATA 步的开始

data 语句是 DATA 步的开始边界，为创建的输出数据集命名。当 DATA 步用于创建视图(View)或者经编译的 DATA 步程序时，data 语句为创建的视图或者 DATA 步程序命名。

为创建 SAS 数据集命名，语句格式为

data <name1 <name2 … namen>>;

name1、… 、namen 为输出数据集名称，当一次生成多个数据集时，数据集名之间空格隔开。如果不给出任何 SAS 数据集名，系统将生成的数据集命名为 data1、data2、…等。如果用自动变量_data_作为数据集名，则等价于不给出数据集名。不生成数据，语句格式为

```
data _null_;
```

语句只运行 DATA 步中的语句而不生成数据集，DATA 步的目的是完成其他的操作而不是生成数据集。

为创建 SAS 视图命名，语句格式为

```
data name/view=name;
```

选项/view=表明生成的是视图而不是数据集。创建一段编译的 DATA 步程序，语句格式为

```
 data 数据集名/pgm=程序名;
```

选项/pgm=表明 DATA 步被作为程序编译和保存，并不执行。运行语句

```
 data pgm=程序名;
```

执行保存的 DATA 步程序。

3.3.2 SET 语句—SAS 数据集观测的读取

set 语句每执行一次，SAS 将输入数据集的一条观测读入 PDV。DATA 步编译时为 set 语句设定数据指针(pointer)，set 语句将数据指针指向的观测读入 PDV。如果没有限制，set 语句从输入数据集中按观测序号顺序读入观测：当前 set 语句执行完后，该 set 语句下一次执行将数据指针按观测号下移到下一个观测读取，读遍输入数据集所有观测后数据指针到达文件底部遇到文件结束标志（end-of-file indicator），DATA 步结束运行。

一个 set 语句可以读取多个数据集观测，DATA 步从第一个数据集读取观测，按数据集次序依次读取其它数据观测，在读完一个数据集观测之后和读取下一个数据集第一条观测之前，PDV 中数据集变量值初始化为缺失。在读取所有输入数据集最后一条观测后，到达最后一个数据集文件结束标志，DATA 结束运行。

一个 data 步中可以有多个 set 语句。不同 set 语句的数据指针独立工作，互不影响。只要有一个 set 语句数据指针到达文件结束标志，DATA 步结束运行。

可以通过语句选项控制 set 语句读取观测的顺序和方式。set 语句的一般格式为

set <name1 <name2 … namen> <选项>;

其中 name1、…、namen 为数据集名，可以采用缩略列示方法。如果没有数据集名，则默认为刚生成的数据集。

程序 3-8

```
data class;
set sashelp.class;
put _all_;
run;
```

注意程序 3-9

程序 3-9

```
data class;
set sashelp.class;
set sashelp.class;
run;
```

中是两个独立的 set 语句，有各自数据指针，不是同一个 set 语句执行两次，第二个 set 语句读取的观测与第一个一样，并不是第一个 set 语句读取观测的下一个观测。

set 语句选项

❋ **curobs=vname**

产生一个临时变量，其取值为输入数据集中当前读入 PDV 中观测（current observation）的观测号，vname 为变量名，由编程者给出。该临时变量只存在于所在 DATA 步，并且不输出到输出数据集，但可以用在编程语句。例如

程序 3-10：set 语句顺序读取观测

```
data class;
set sashelp.class curobs=gc;
p=gc;
run;
```

中赋值语句 p=gc;中用到临时变量 gc，输出数据集 class 中包含 p。

❋ **end=vname**

产生一个临时变量，用于标识当前读入 PDV 的观测是否输入数据集的最后一条观测。vname 是变量的名字，由编程者给出，当前读入观测是最后一条时取值 1，否则取值 0。该临时变量只在所在 DATA 步存在，并且不输出到输出数据集，但可以用在编程语句。程序 3-11 输出数据集 sashelp.class 的最后一条观测到 class 中：

程序 3-11：set 语句顺序读取观测

```
data class;
set sashelp.class end=gc;
if gc then output;
run;
```

其中 if-then 为有条件执行语句，ouput 的功能是将 PDV 中变量值输出。

❋ **indsname=vname**

产生一个临时变量，其取值为读入PDV的观测属于的数据集名字（包括逻辑库名）。vname是变量的名字，由编程者给出。该临时变量只在所在DATA步存在，并且不输出到输出数据集，但可以用在编程语句。关键词indsname是in data set name的缩略。程序3-12生成三个数据集sashelp.code、work.date和sasuser.clp，分别是股票代码、交易日期和收盘价，用set语句以及选项indsname=生成数据集work.price,包含变量stkcd、date和clp：

程序 3-12

```
data sashelp.stkcd;
input stkcd $;
datalines;
600013
```

```
;
run;
data date;
input date ymmdd8.;
datalines;
20030310
;
run;
data sasuser.clp;
input  clp;
datalines;
5.23
;
run;
data price;
set sashelp.stkcd date sasuser.clp indsname=dsn;
name=dsn;
run;
```

点击打开数据集通过变量name的取值查看变量dsn取值。

❋ **nobs=vname**

产生一个临时变量，取值等于输入数据集观测个数，多个输入数据集时，取值等于所有输入数据集观测个数总和。vname为变量名，由编程者给出，该变量只存在于所在DATA步，不输出到输出数据集。

❋ **point=var**

只读取观测序号等于变量 var 值的观测。var 是事先定义的变量。point=选项将 set 语句读取观测的方式控制为随机读取，而不再是默认情况下顺序读取，set 语句数据指针不会碰到文件结束标志，导致 DATA 步死循环，因此需要结合 stop 语句使用。程序 3-13 输出一条观测,所有变量的值为缺失值。

程序 3-13

```
data ss;
set sashelp.class end=j point=j;
output;
stop;
run;
```

因为在没有到达最后一条观测时j的值为0，point=选项发生错误，output语句将编译阶段PDV中的值输出为一条观测。程序3-14输出sashelp.class第一条观测，

程序 3-14

```
data ss;
set sashelp.class end=j;
set sashelp.class point=j;
output;
stop;
run;
```

因为第二个 set 语句执行时发生错误，没有观测读入到 PDV 中，output 语句输出的仍然是第一个 set 语句读入的观测值。

point=选项不能和顺序读取下的语句（如 by 语句、where 语句）或者 where 数据集选项一同使用。

3.3.3 数据集观测操作

对观测进行筛选形成的新的变量和修改观测值是常见的数据集观测操作，前者也称为取子集，后者也称为观测编辑。

1．取子集（subsetting）

从已有数据集中筛选（filtering）观测形成新的数据集，是数据整理中常遇到的操作，可以通过取子集 if 语句（if statement subsetting）、where 语句和数据集选项来完成。

❋ **取子集 if 语句**

取子集 if 语句分为 if 和 if-delete。if 语句的格式为

if 表达式;

该语句为可执行语句，其功能是：如果表达式为真（取值1）则执行后续语句，到达DATA步结束边界时，PDV中变量当前值输出形成输出数据集的一条观测；如果表达式为假（取值0或者缺失值）则不执行后续语句，返回DATA步的开头进行新一次内置循环，由于没有达到DATA步结束边界，PDV中变量当前值不被输出。取子集if语句的客观效果是满足条件的观测被挑选出来（输出）形成新的数据集。程序3-15挑选sashelp.class中的男性观测形成数据集work.s，同时生成新的变量h。

程序 3-15

```
data s;
set sashelp.class;
if sex='男';
h=height*0.025;
run;
```

可采用 if-delete 语句进行观测筛选。if-delete 语句的格式为

if 表达式 then delete;

if-then-delete 语句为可执行语句，其功能是：如果表达式为假（取值 0 或缺失值）则执行后续语句，到达 DATA 步结束边界时，PDV 中变量当前值输出形成输出数据集的一条观测；如果表达式为真（取值 1）则不执行后续语句，返回 DATA 步的开头进行新一次内置循环，由于没有达到 DATA 步结束边界，PDV 中变量当前值不被输出。程序 3-16 挑选年龄小于 14 岁的男生形成数据集 work.s。

程序 3-16

```
data s;
set sashelp.class;
if sex='女' and age>=14 then delete;
h=height*0.025;
run;
```

❋ **取子集 where 语句**

采用 where 语句也可以实现观测筛选，语句格式为

where 表达式;

语句功能是将输入数据集中（不）符合条件的观测（不）读入 PDV，条件由表达式值确定，表达式值为真（取值 1）表明条件成立，为假（取值 0 或者缺失值）表明条件不成立。 程序 3-17 与 3-15 具有相同效果。

程序 3-17

```
data s;
set sashelp.class;
where sex='男';
h=height*0.025;
run;
```

where 语句为声明性语句，在 DATA 步编译阶段发挥作用，因此表达式中只能包含输入数据集中的变量。程序 3-18 中 where 表达式包含非数据集变量_n_而出现语法错误：

程序 3-18

```
data s;
set sashelp.class;
where _n_>=3;
run;
```

提交运行后信息窗口出现提示：ERROR: 变量 _n_ 不在文件“SASHELP.CLASS”中。

程序 3-19 用 height 定义新变量 h，然后在 where 语句中用 h 形成表达式 h=10,以此对读入 PDV 的观测进行筛选

程序 3-19

```
data h;
set sashelp.class;
h=height*0.25;
where h=10;
run;
```

提交运行后信息窗口出现提示：ERROR: 变量 h 不在文件“SASHELP.CLASS”中。

与取子集 if 语句相比，where 语句的有两个优势，一是在编译阶段起作用，不满足条件的观测不读入 PDV，运行效率高，二是有其特有的表达式运算符，表达式的书写很灵活，表现力强，细节可参考“SAS 表达式”中 “where 表达式”。程序 3-20 将数据集 sashelp.baseball 中名字包含字符’Bon’的队员观测挑出生成数据集 work.bon。

程序 3-20

```
data bon;
set sashelp.baseball;
where name ? 'Bon';
run;
```

2．数据集选项—观测控制

数据集选项（data set options）对 SAS 数据集实施观测和变量筛选或者其它操作。数据集选项的一般格式为

数据集名(选项)；

输入数据集选项和输出数据集选项的作用环节不相同，输入数据集选项对输入 PDV 的数据实施筛选和操作，而输出数据集选项则对 PDV 输出到输出数据集的数据实施筛选和操作。数据集选项不仅可以用在 DATA 步，也可以用在 PROC 步。这里讨论 DATA 步对观测筛选的数据集选项。

❋ 输入数据集观测范围选择—firstobs=n 和 obs=m

选项 firstobs=m 和 obs=n 通过观测号选择观测范围,m、n 为正整数。firstobs=m 选择输入数据集从第 m 个观测开始读入 PDV。obs=n 选择输入数据集第 n 个（包括第 n 个）之前的观测读入 PDV。两个选项只可用于输入数据集，可以单独使用，也可以一起使用，按观测号范围选择读入 PDV 的观测。程序 3-21 选择第 3 条到第 15 条观测之间、名字包含字符’Bon’的观测生成数据集 work.bon_3_15

程序 3-21

```
data bon;
set sashelp.baseball(firstobs=3 obs=15);
where name contains 'Bon';
run;
```

❋ **有条件选择观测——where=(表达式)**

数据集选项 where=(表达式)称为 where 数据集选项，通过条件选择观测，表达式成立（取值 1）的观测被选择，表达式不成立（取值 0 或者缺失值）的观测不被选择。用于输入数据集的 where=选项在 DATA 步每次内置循环开始时起作用，对读入 PDV 的观测进行选择，用于输出数据集的 where=选择在 DATA 步每次内置循环结束前起作用，对 PDV 读出到输出数据集的观测进行选择。where 数据集选项可以和 where 语句一起使用，SAS 忽略 where 语句对带 where 选项数据集的作用。where 数据集选项中的表达式可以使用 where 表达式运算符。程序 3-22 挑选 name（名字）首字母为'B'、Team（所属球队）名字首字母为'M'且 nhits（击中次数）大于 60 的球员的观测生成数据集 work.c1。

程序 3-22

```
data c1(where=(Team=:'M' and nhits>60));
set sashelp.baseball(where=(name like 'B%'));
run;
```

❋ **观测的数据集标记——in=变量名**

当采用 set、merge 等语句对多个 SAS 数据集进行操作时，往往需要知道处理的观测来自哪个数据集。数据集选项 in=vname 生成 DATA 步中的一个临时变量 vname（vname 由编程者给出），对 PDV 中的观测是否来自该数据集进行标记，取值 1（0）表示（不）来自该数据集。in=数据集选项只适用于输入数据集。in=定义的临时变量只存在于所在 DATA 步，不输出到输出数据集，但可用于 DATA 步编程。程序 3-23 首先生成数据集 work.A 和 work.B，然后将数据集叠放形成新数据集 work.c，新数据集包含变量 a 和 b 标识其观测是否来自数据集 A 或者 B。

程序 3-23

```
data A;
input shoole $ grade $ class $ name $;
datalines;
光明 初三 2 程辉
鲁迅 初二 3 刘晗
 ;
data B;
input shoole $ grade $ class $ name $;
datalines;
存志 初一 1 沈梦
复兴 初二 4 张倩
 ;
run;
data c;
set a(in=h) b(in=j);
a=h;b=j;
run;
```

3．修改观测值

用赋值语句可以对指定的观测进行修改。程序 3-24 将 sex 值为'男'的观测值修改为'M'。

程序 3-24

```
data class;
 set sashelp.class;
 if sex='男' then sex='M';
```

```
run;
```

4．输出节点控制—output 语句

默认情况下 DATA 步内置循环在遇到结束边界时输出 PDV 中变量值到输出数据集。一些情况下需要立即输出 PDV 中变量的值而不等 DATA 步结束边界处自动输出。output 语句的功能是将 PDV 中变量的当前值输出，称为显性输出（explicit output）。一般格式为

```
output <dsname>;
```

dsname 是输出数据集名字，是可选项，缺省情况下 output 的输出数据集为 data 语句中给出的数据集。DATA 步中包含 output 语句时，内置循环的隐性输出功能被屏蔽（overrided），遇到结束边界不再自动输出。程序 3-25 将 sashelp.class 的第 5 条观测输出生成数据集 work.obs_5。

程序 3-25

```
data class;
x=5;
set sashelp.class point=x;
output;
stop;
run;
```

没有 output 语句，DATA 步在遇到结束边界 run 语句之前执行 stop 语句而结束运行，PDV 中数据不被输出。没有 stop 语句，DATA 步数据指针始终停留在第 5 条观测，不能到达文件结束标志而形成死循环。

output 语句可以用输入数据集的一条观测形成输出数据集的多条观测。程序 3-26 用数据集 work.test1

VIEWTABLE: Work.Test1

	city	year1	year2	year3
1	上海	2017	2018	2019

生成数据集 work.test

VIEWTABLE: Work.Test

	city	year
1	上海	2017
2	上海	2018
3	上海	2019

程序 3-26

```
data test;
set test1;
year=year1;output;
year=year2;output;
year=year3;output;
keep city year;
run;
```

3.3.4 数据集变量操作

数据集变量的操作包括变量筛选、变量属性修改，以及 DATA 步内置循环中 PDV 变量值控制。

1． 变量筛选

变量筛选是从输入数据集中挑选变量形成新的数据集，是最为常用数据整理操作。变量筛选既可以用语句实现，也可以用数据集选项实现。

❋ **keep 和 drop 语句**

keep（drop）语句选择 DATA 步的 PDV 中哪些变量（不）输出到输出数据集。keep 语句和 drop 语句都是可执行语句，可放置在 DATA 步中的任何地方。其一般格式为

keep 变量名列表； drop 变量名列表；

其中变量名列表可以采用变量名缩略列示法。程序 3-27 用 sashelp.class 中的变量 name、sex、height 生成新数据集 work.c，两个程序段具有相同功能。

程序 3-27

```
data c;
 set sashelp.class;
 keep name sex height;
run;
data c;
 set sashelp.class;
 drop age weight;
run;
```

keep 语句和 drop 语句是在 PDV 数据读入输出数据集的环节起作用。

❋ **数据集选项 keep=和 drop=**

变量筛选也可以通过数据集选项 keep=和 drop=来实现。一般格式为

keep=变量名列表 和 drop=变量名列表

keep=（drop=）作用于输入数据集时，只有变量名列表中出的变量（不）在 PDV 分配单元并在 DATA 步内置循环中被处理中。keep=(drop=)作用于输出数据集时，只有变量名列表中的变量才（不）输出为输出数据集的一个变量，但所有变量都参与了 DATA 步内置循环的处理。变量名列表可以是 SAS 允许的任何方式，包括缩略列示。程序 3-28 与程序 3-27 具有相同的结果。

程序 3-28

```
data c(keep=name sex height);
 set sashelp.class;
run;
data c;
 set sashelp.class(drop=age weight);
run;
```

程序 3-29 运行时因 PDV 中没有变量 height，height 以及赋值语句得出的新变量 h 取缺失值，并在信息窗口提示“变量 height 未初始化”。注意，程序 3-29 没有语法错误（syntax error）。

程序 3-29

```
data c;
 set sashelp.class(drop=height);
 h=height*.025;
run;
```

keep(drop)语句与 keep=(drop=)数据集选项的区别主要体现在:(i)keep(drop)语句只对输出数据集起作用，并不对输入数据集读入 PDV 中的变量进行筛选，输入数据集所有变量都进入 DATA 步内置循环，而 keep=（drop=）数据集选项既可以作用于输出数据集，也可作用于输入数据集；(ii) keep=（drop=）对变量的筛选更为灵活，例如当有多个输出数据集时，keep（drop）语句不得不对所有输出数据集都筛选（去掉）相同的变量，而数据集选项 keep=（drop=）对每个数据集可以筛选不同的变量。

2． 变量属性修改

变量是 SAS 数据集的最基本元素，变量属性包括名字（name）、类型（type）、输入-输出格式（informat-format）、宽度（length）和标签（label）。对变量属性进行设定和修改是数据处理常见操作。

❋ 变量改名—rename

变量改名分为语句和数据集选项两种方式，其作用节点和用途不相同。

变量改名—rename 语句

rename 语句的一般格式为

rename 原名 1=新名 1 原名 2=新名 2 … 原名 n=新名 n;

将 PDV 写入输出数据集中的变量进行改名。rename 语句是声明性语句，在编译阶段起作用。程序 3-30 将 sashelp.class 中变量 height 改名为 h、weight 改名为 w 生成数据集 work.nc。

程序 3-30

```
data nc;
set sashelp.class;
rename height=h weight=w;
run;
```

rename 只对输出数据集变量起作用，不改变 DATA 步 PDV 中变量名。

rename 和 keep 语句出现在同一 DATA 步时，SAS 先编译 keep 语句，后编译 rename 语句，要避免 rename 语句修改 keep 语句中没有包含的变量。运行程序 3-31

程序 3-31

```
data null;
set sashelp.class;
rename height=h;
keep name age sex;
run;
```

后信息窗口发出警告(warning)：“从未引用过 keep、drop 或 rename 列表中的变量 height”。drop 语句和 rename 语句具有类似的协调关系。

变量改名—数据集选项 rename=

数据集选项 rename=的一般格式为

rename=(原名 1=新名 1 原名 2=新名 2 … 原名 n=新名 n)

rename=选项作用于输入数据集时，将输入数据集变量改名后读入 PDV，经历 DATA 步内置循环；作用于输出数据集时与 rename 语句一样，将 PDV 写入输出数据集中的变量进行改名，PDV 中变量不受影响。

在同一 DATA 步内有多个 set 语句时，rename=选项用于 set 语句的输入数据集避免读入 PDV 同名变量时先读入的变量值被后读入的变量值覆盖。程序 3-32 用 sashelp.class 生成新数据集 work.dc，新数据集有变量 name 和 name1 取值相同、age 和 age1 取值相同。

程序 3-32

```
data dc;
set sashelp.class(rename=(name=name1));
set sashelp.class(rename=(age=age1));
run;
```

第一个 set 语句将 sashelp.class 的观测读入 PDV 时，将变量 name 改名为 name1，第二个 set 语句将 sashelp.class 的观测读入 PDV 时，变量 name 的值不会覆盖 name1

的值，输出数据集有两个变量 name 和 name1。变量 age 和 age1 的生成机理类似。其它变量没有改名,第二个 set 语句读入的观测值覆盖第一个 set 语句读入的同名变量观测值。

rename 语句、rename=数据集选项与 keep（drop）语句以及 keep=（drop=）数据集选项也存在协调性问题。运行程序 3-33

程序 3-33

```
data dc;
set sashelp.class(rename=(sex=xb));
keep sex;
run;
```

会出现类似运行程序 3-31 时信息窗口的警告信息。

❋ 变量数据类型转换

根据存放的数据类型，SAS 数据集变量分为字符型变量和数值型变量。运算和操作中要求数据匹配，需要对数据集中变量原来的数据类型进行转换。例如，如果变量 stkcd（股票代码）为字符变量，用 where 语句挑选特定股票（600258 和 600743）观测值的语句

```
where stkcd in (600258 600743);
```

中的比较运算中，字符变量 stkcd 的类型与括号的数值常量不匹配，需要对 stkcd 进行类型转换。

字符转数值

有些情况下 SAS 会对数据类型不匹配的变量自动进行转换，然后再进行有关操作。如果自动转换不可行，则可以用 input()函数进行转换。

自动转换

有些情形下，SAS 尝试将字符变量（或者字符常量）转换为数值变量（或者数值常量），转换成功时会在信息窗口给出提示信息，转换不成功则在信息窗口给出错误提示信息，并将 PDV 中自动变量_ERROR_的值置为 1。自动转换的情形包括：

- 将字符常量或者字符变量赋值给数值型变量；
- 算数运算中使用字符变量；
- 在比较运算中将字符变量和数值常量或者数值变量）进行比较（仅限于 if 语句，where 语句和数据集选项 where=的表达式中不进行自动转换）；
- 在应该为数值型变量（或者常数）的函数自变量位置引用字符变量（或者字符常数）。

程序 3-34 将字符变量 x 赋值给数值变量 y，SAS 对 x 实施自动类型转换。

程序 3-34

```
data dc;
y=1;
x='2';
y=x;
run;
```

但如果定义变量 x 的语句是 x='a';，则自动转换失败，并导致 y 取缺失值。

用函数 input()转换

一般格式为

input(源数据,输入格式)

功能是按指定的输入格式读取字符型源数据。输入格式决定了函数的返回值的数值类型：输入格式为数值型（字符型），则 input 以数值型（字符型）格式读取字符型源数据，函数返回值为数值型（字符型）数据。将输入格式设定为数值型，字符型源数据就被转换为数值型数据。源数据必须是字符型数据，可以是变量、常数或者表达式。程序 3-35 中第一个 DATA 步生成数据集 work.w 时，将每小时工资 rate 设置为字符变量，第二个 DATA 步计算应付工资 pay 时,SAS 对变量 rate 实施自动类型转换,第三个 DATA 步采用 input 函数将 rate

转换为数值变量。

程序 3-35

```
data w;
input name $ rate $ hours;
cards;
Jack 10 88
Simon 8 200
Edward 40 208
;
data pay1;
set w;
pay=rate*hours;
data pay2;
set w;
pay=input(rate,2.)*hours;
run;
```

数值转字符

同字符转数值类似，数值转字符也分自动转换和函数转换。

自动转换

自动转换的情形包括：

- 将数值常量或者变量赋值给字符型变量；
- 字符运算中使用数值变量或者常数；
- 在比较运算中将数值变量和字符（或者字符变量）进行比较（仅限于 if 语句，where 语句和数据集选项 where=的表达式中不进行自动转换）；
- 在应该为字符型变量（或者常数）的函数自变量位置引用数值变量（或者数值常数）。

用函数 put()转换

函数一般格式为

put(源数据,输出格式)

功能是按输出格式将源数据转换为字符。源数据可以是任何类型的数据，可以是常数也可以是变量。put 函数总是返回字符数据。函数中的输出格式类型必须与源数据的类型一致。如果源数据为数值型（字符型），则函数中的输出格式也必须是数值型（字符型）。数值型（字符型）输出格式得出的返回值是右（左）对齐的。使用 put 函数时，对未设定长度的目标变量其长度与格式规定的宽度一致。

程序 3-36 中第一个 DATA 步生成数据集 work.p 时，将 code（区号）设置为数值型变量，第二个 DATA 步将 code 用左右小括号括住后和 number（号码）连接，SAS 将 code 自动转换为字符变量，第三个 DATA 步采用 put 函数将 code 转换为字符变量后进行字符连接。

程序 3-36

```
data p;
input code number $;
cards;
303 393-0956
919 770-8292
301 449-5239
;
data phone;
set phone;
phone='('||code||')'||number;
run;
data phone;
set phone;
```

```
phone='('||put(code,3.)||')'||number;
run;
```

❋ **输入输出格式设置**

输入-输出格式是数据集变量的重要属性，informat 和 format 语句可以设置变量的输入-输出格式。informat 和 format 语句都是声明性语句。

输入格式设置—informat 语句

DATA 步中的 informat 语句对 input 语句定义的数据集变量输入格式进行设定。也可以在 input 语句中直接定义变量的输入格式。informat 语句的一般形式为

informat 变量名列表 <输入格式>;

如果没有输入格式选项，则取消以前定义的输入格式。语句

informat default=输入格式;

用于定义所在 DATA 步中变量的临时缺省输入格式。程序 3-37 将 input 语句中变量 name 的输入格式定义为$char4.，即宽度为 4 的字符变量。

程序 3-37

```
data a;
informat x1-x5 3.1 name $char4.;
input x1-x5 name;
datalines;
11 22 33 44 100 John
;
run;
```

DATA 步中 informat 语句对输入 SAS 数据集变量不起作用。

输出格式设定—format 语句

format 语句对变量的输出格式进行设定，包括 SAS 数据集变量输出格式重设。语句格式为

format 变量名列表 <输出格式>;

如果没有输入格式选项，则取消以前定义的输出格式。语句

format default=输出格式;

规定所在 DATA 步变量的临时缺省输出格式。程序 3-38 用赋值语句定义变量 w、x、y、和 z，用 format 语句定义变量输出格式、数值变量临时缺省输出格式和字符变量临时输出格式。

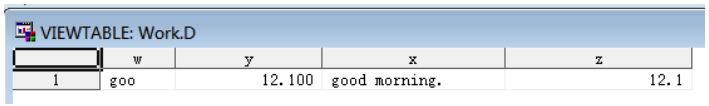
程序 3-38

```
data d;
format w $3. y 10.3 default=8.2 default=$8.;
w='good morning.';
x='good morning.';
y=12.1;
z=12.1;
put w/x/y/z;
run;
```

put 语句执行后（符号/的功能是分行显示）在 log 窗口显示结果为

```
goo
good mor
12.100
122.10
```

而打开数据集后显示



两种输出结果不一样。原因在于 put 是 DATA 步内的输出命令，default=8.2 和 default=$8.对没有定义输出格式的变量 x 和 z 发挥作用，而点击打开数据集是 DATA 步外的输出，临时默认输出格式不起作用。

用 length 语句和 label 语句可以设定变量的宽度和标签，详情可以参考 SAS 帮助。

❋ **取值保留—retain 语句**

在 DATA 步执行阶段，新的一次内置循环开始时，赋值语句和 input 语句定义的变量在 PDV 中的值重新初始化为缺失,本次循环不能采用上次循环的变量值。程序 3-39 在 DATA 步中用函数 max()定义变量 maxh 用于计算 sashelp.class 中学生的最高身高。

程序 3-39

```
data hightest;
set sashelp.class;
maxh=max(maxh,height);
run;
```

运行结果显示 maxh 的变量值与变量 height 值相等。编译后的第一次内置循环 PDV 中 maxh 的值为缺失值，与第一条观测 height 值取最大后等于 height 值。第二轮内置循环开始时，maxh 的值初始化为缺失，与第二条 height 值取最大后仍然等于 height 值，…。

retain 语句将赋值语句和 input 语句定义的变量的值在新一轮内置循环开始时得以保留。retain 语句为声明性语句，格式为

retain <变量名列表>;

如果没有变量名列表,retain 语句默认对所有赋值语句和 input 语句定义的变量起作用。程序 3-40 语句 retain 保留 maxh 上次循环值，

程序 3-40

```
data hightest;
set sashelp.class;
retain maxh;
maxh=max(maxh,height);
run;
```

retain 语句的另外一个功能是给_N_和_ERROR_以外的变量赋初始值，即在第一次内置循环开始时将变量的值由编译阶段的缺失值置为初始值。语句格式为

retain 变量名 1 初始值 1 <变量名 2 初始值 2 …>;

变量名可以是多个变量名列表，初始值的列表值与之匹配。程序 3-41 为 sashelp.class 中变量 age 赋初值 100。

程序 3-41

```
data cz;
put _all_;
set sashelp.class;
retain age 100;
run;
```

从 put 语句输出到信息窗口的信息可以看出,在 set 语句执行之前 age 的取值已经是 100，而其它数据集变量的值为缺失值。

❋ **变量值累加—累加语句**

SAS 的累加语句采用更加紧凑的形式，一般格式为

累加变量+累加值

累加变量定义一个累加器（accumulator）变量，每次循环将累加值累加到累加变量上，累加器变量是数值型变量，累加值是一个表达式，表达式可以是数值型的，也可以是可转换为数值型的字符型表达式。SAS 自动给累加变量赋初始值 0，并在每次 DATA 步内置循环开始时保留上次循环的值，相当于执行了语句 retain 累加变量 0;。累加语句是可执行语句。程序 3-42 的累加变量 ta 计算了 sashelp.class 中学生的年龄平方和。

程序 3-42

```
data ta;
set sashelp.class;
ta+age**2;
run;
```

如果累加变量的初值不是 0，需要用 retain 语句给累加变量赋初值。

DATA 步中的累加计算也可以采用 x=x+Δ（或者 x=sum(x,Δ)）的赋值语句形式，关键是给累加变量 x 赋初值。程序 3-43 程序 3-42 有相同的结果

程序 3-43

```
data ta;
set sashelp.class;
retain ta 0;
ta=ta+age**2;
run;
```

但如果没有 retain 语句给变量 ta 赋初值，则 ta 的初始值为编译阶段的缺失值，导致运行结果观测值都是缺失值。

累加语句和求和函数 sum()有两点不同：(1)用 sum()函数定义的累加变量不具有自动保留功能，要实现累加需要用 retain 语句；更为重要的是（2）sum()是统计函数，只对自变量中的非缺失值进行计算，而累加语句是表达式，SAS 将运算结果为缺失值表达式作为 0 处理。

❋ **变量次序调整**

变量在 SAS 数据集中的次序在数据集创建时确定，例如将 Excel 文件导入为 SAS 数据集时保留了原表格中变量的次序，用 input 语句读入外部数据生成的数据集中变量按在 input 语句中出现的顺序排列。用 retain 语句可以重新安排数据集中变量的次序。程序 3-44 对 sashelp.class 中变量次序进行重排。

程序 3-44

```
data cp;
retain weight height sex;
set sashelp.class;
run;
```

需要注意的是，retain 语句一定要在 set 语句之前，先编译 retain 语句确定语句中变量在 PDV 中的位置，后编译 set 语句再确定其余变量在 PDV 中的位置。将 retain 语句放在 set 语句之后起不到重排变量次序作用。

也可以用多个 set 语句结合 drop=数据集选项实现变量次序重排，如程序 3-45

程序 3-45

```
data cp1;
set sashelp.class(drop=name sex);
set sashelp.class;
run;
```

3.4 DATA 步—执行流程控制

默认情况下 DATA 步每次内置循环按语句的自然顺序自上而下执行，这种执行方式称为顺序处理。顺序处理与分支控制和循环控制形成 DATA 步执行流程控制的三种主要方式。

3.4.1 分支控制

分支控制语句通过判断条件是否成立确定后续语句执行的顺序。DATA 步的分支控制语句分为 IF-THEN 语句、IF-THEN-ELSE 语句和 SELECT-WHEN 语句。

1．IF-THEN 语句

IF-THEN 语句为单分支控制语句，一般格式为

If 表达式 then 语句；

当表达式的值为非零、非缺失时，执行 then 后面的语句，为零或者缺失值不执行。程序 3-46 将性别为男的观测输出生成新数据集 work.m。

程序 3-46

```
data m;
set sashelp.class;
if sex='男' then output;
h=height*.025;
run;
```

尽管程序 3-46 中的分支控制语句与程序 3-15 中取子集 IF 语句具有同样的效果，但执行顺序并不一样。IF-THEN 语句不影响后续语句的执行，不管 sex='男'的比较结果如何，h=h*.025 都会执行，而取子集 IF 语句中的条件不成立的话后续语句不被执行。IF-THEN 语句为可执行语句。

程序 3-47

```
data cp1;
if 0 then set sashelp.class(drop=name sex);
set sashelp.class;
run;
```

程序 3-47 的第一个 set 语句并没有执行，在编译阶段只给 age、weight 和 height 按先后次序在 PDV 中开辟内存单元，结合第二个 set 语句的执行对数据集变量次序进行重排，和程序 3-45 具有同样的效果。

2．IF-THEN-ELSE 语句

IF-THEN-ELSE 语句为双分支控制语句，一般格式为

If 表达式 then 语句 1；
else 语句 2；

表达式为真（非零值、非缺失值）执行语句 1，否则执行语句 2。IF-THEN-ELSE 语句具有两个分支，表达式为真或者假执行不同的语句，而 IF-TEHN 语句为单分支语句，表达式为假不执行任何语句。程序 3-48 将数据集 sashelp.class 按性别拆分为两个数据集。

程序 3-48

```
data m f;
set sashelp.class;
if sex='男' then output m;
else output f;
run;
```

IF-THEN 语句和 IF-THEN-EELSE 语句可以嵌套。程序 3-49 根据学生体重的范围定义变量 WtGrade。

程序 3-49

```
data wg;
set sashelp.class;
if weight<90 then WtGrade='Light';
else if weight<110 then WtGrade='Middle';
else WtGrade='Heavy';
run;
```

嵌套时要注意不同的 IF-THEN 和 ELSE 的对应关系。

要注意，delete 语句的功能决定了 if-then-delete 语句在 if 条件成立时返回 DATA 步开头，进行下一次循环，实现 DATA 步“短路”。

3．DO-END 语句组与分支控制

分支控制语句中， THEN 和 ELSE 后面只能出现一条语句，功能受到限制。DATA 步采用 DO-END 语句将多条语句封装为 DO-END 语句组（DO-EDN group）。DO-EDN 语句组放在 THEN 或者 ELSE 关键词之后作为一个整体，使分支流程可以执行一组语句，拓展了流程控制的功能。语句格式为

```
DO;
语句 1
⋮
语句 n
EDN;
```

语句组以 Do;为开始边界、END;为结束边界。程序 3-50 生成体重分类变量 WtGrade 的同时，用体重和身高数据生成体型变量 Figure。

程序 3-50

```
data F;
set sashelp.class;
if weight<90 then
do;
WtGrade='Light';
if height>60 then figure='Slim';
else figure='Moderate';
end;
else if weight<110 then WtGrade='Middle';
else
do;
WtGrade='Heavy';
if height<60 then figure='Fat';
else figure='Moderate';
end;
run;
```

4．SELECT-WHEN 语句

SELECT-WHEN 为多分支控制语句，可以实现多条件下的分支控制。一般格式为

```
select <表达式>;
When (表达式 1 <,表达式 2> <,…>) 语句 1;
<When (表达式 1 <,表达式 2> <,…>) 语句 2;>
<…>
<Otherwise 语句;>
End;
```

如果 select 语句中有表达式，则将表达式的值与 when 语句中表达式的值进行比较：首先与第一个 when 语句中的第一个表达的值比较，如果二者相等，执行语句 1 并跳出 select-when 分支，否则与第二个表达式比较，… ，如果与第一个 when 语句中所有表达式值都与不相等，则与第二个 when 语句中表达式值进行比较，…，只要与某个 when 表达

式中某个表达式值相等，则执行该 when 语句中的语句并跳出分支，如果与所有 when 语句的所有表达式值都不相等且分支中包含 otherwise 语句，则执行 otherwise 语句中的语句并跳出分支，如果没有 otherwise 语句则停止 DATA 步执行，并发出错误信息。程序 3-51 模拟抛一枚骰子的结果，筛子的六个面分别标注数字 1、2、…、6，并且出现的概率均为 1/6。随机数生成函数 rand('UNIFORM')生成(0,1)上均匀分布的随机数，ceil(x)为上取整函数，函数值是大于等于 x 的整数。

程序 3-51

```
data _null_;
x=ceil(rand('UNIFORM')*6);
select(x);
when(1) put '1';
when(2) put '2';
when(3) put '3';
when(4) put '4';
when(5) put '5';
when(6) put '6';
otherwise put '*';
end;
run;
```

如果 select 语句中没有表达式，分支语句的执行依赖 when 语句中表达式是否为真。when 语句中表达式的设置可以更加灵活。程序 3-52 模拟抛掷两枚筛子的可能结果。

程序 3-52

```
data _null_;
x=ceil(rand('UNIFORM')*6);
y=ceil(rand('UNIFORM')*6);
select;
when(x<=3 and y<=3) put '(小，小)';
when(x<=3 and y>3) put '(小，大)';
when(x>3 and y<=3) put '(大，小)';
when(x>3 and y>3) put '(大，大)';
end;
run;
```

3.4.2 循环控制

一种操作重复进行时，需要循环语句来实现。DATA 步中的循环语句都是以 DO 语句开始 END 语句结束，因此又称 DO-EDN 循环（注意和 DO-END 语句组区别），简称 DO 循环。要注意 DO-END 循环和 DATA 步内置循环的区别和联系。

1．指定次数循环

定义指标变量(index-variable)并通过变量取值的变化控制循环。一般格式为

```
Do 指标变量=初值 To 终值 <by 步长>;
SAS 语句;
End;
```

指标变量从初值开始取值，每次循环开始时判断指标变量是否越出终值规定的范围，没有越出则开始循环。每次循环执行 DO 语句和 END 语句中间的语句（称为循环体），END 为循环体下边界，循环体语句执行完之后指标变量按 by 语句给定的步长增加，如果没有 by 关键词则步长默认为 1，并判断指标变量新的取值是否越出终值规定的范围确定是否进行新一轮循环。当指标变量取值越出终值规定的范围，循环结束，DATA 步继续执行 END 语句的后续句。指标变量作为新定义变量输出到输出数据集，可用 drop 语句去掉。

程序 3-53 用 DO 循环语句生成 10 个标准正态分布样本值，每次循环随机数生成函数 rannorm()产生一个标准正态分布随机数赋值给变量 x。

程序 3-53

```
data rnor;
do i=1 to 10;
x=rannorm(123);
end;
run;
```

数据集 work.rnor 中有两个变量 i 和 x，但只有一条观测，况且 i 的值是 11 而不是 10！从 DATA 语句到 RUN 语句的 DATA 步内置循环只运行了一次，在这一次内置循环中 DO 循环运行了 10 次，每次 DO 循环都在碰到循环体下边界 end 语句时返回到循环体开始，并没有运行 end 下面的 run 语句，因此 PDV 中的变量值没有输出。当第 10 次循环完成后，指标变量 i 的变为 11，越出了终值规定的范围，跳出循环体运行 run 语句并输出 PDV 中变量的值。因此数据集中 i 的值是 11，x 的值是第 10 次 DO 循环得到的随机数。

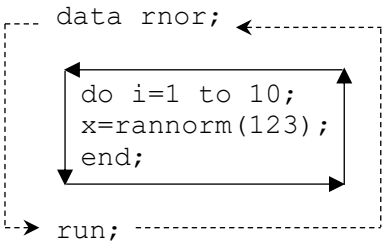


图 3.5 DO 循环和内置循环的关系

循环体内增加 output 语句可将 DO 循环中产生的随机数及时输出。运行程序 3-54 能得到期望的结果。

程序 3-54

```
data rnor;
do i=1 to 10;
x=rannorm(12345);
output;
end;
drop i;
run;
```

DO 循环中指标变量的初值、终值和步长都可以是表达式。程序 3-55 首先生成三家银行存款利率和存款年数数据集 rate，然后计算每年存入$5000 最终的本利和。

程序-55

```
data cdrate;
input bankname $ rate years;
datalines;
HSBC 0.0817  3
BARKLY 0.0814 5
CityBank 0.0806 1
;
data compare;
set cdrate;
investment=5000;
do i=1 to years;
investment+investment*rate;
end;
run;
```

2. 指定内容循环

也可以将指标变量的取值列出，每次 DO 循环指标变量取一个值直到取遍所有值结束循环。具体格式为

Do 指标变量=取值列表;

SAS 语句;

End;

取值列表中各个取值之间逗号间隔。程序 3-56 中第一个 DATA 步生成数据集 work.birth，第二个 DATA 步从中挑选特殊日期的观测值。

程序 3-56

```
data birth;
do i='01JAN1988'd to '01sep2019'd;
birthday=i;
output;
end;
data spebirth;
set birth;
do i='03APR1999'd,'25MAR2004'd,'12JUN2010'd;
if birthday=I then output;
end;
run;
```

3. 指定条件循环—DO Until 和 DO While

除了通过指标变量对循环语句的执行进行控制之外，还可以通过设置条件进行循环控制。指定条件循环分为 Do-Until 循环和 Do-While 循环，语句格式为

Do Until(表达式);

SAS 语句;

End;

和

Do Whilel(表达式);

SAS 语句;

End;

Do Untile 在表达式不成立（为 0 或者缺失）时执行 Do-End 循环体中的语句，直至表达式成立。Do While 语句在表达式成立（表达式值不为 0 也不为缺失值）时重复执行 Do-End 循环体中的语句。程序 3-57 计算年利率 0.05、每年存款￥2000，本利和达到￥50000 需要多少年。

程序 3-57

```
data capital;
do until(capital>=50000);
capital+2000;
capital+capital*.05;
year+1;
end;
run;
```

Do While 在每次循环开始时验证条件，而 Do Until 在每次循环结束验证条件。使用 Do Until 时，不管表达式是否成立，循环至少执行一次，而 Do While 则未必。

指定次数循环和指定条件循环可以一起使用。 程序 3-58 计算年利率 0.05、每年存款￥5000 且存款在 10 年以内本利和达到￥50000 需要多少年。

程序 3-58

```
data invest;
do i=1 to 10 while(capital<50000);
capital+5000;
```

```
capital+capital*.05;
year+1;
end;
run;
```

运行结果表明，在 i=9（8 年末）时本利和已经超过￥50000。

DO 循环程序段（循环体）可以作为 DO-END 语句块出现在分支语句中，包括 if-then 分支语句和 select-where 分支语句。

4．DO 循环的“短路”和“跳出”—CONTINUE 和 LEAVE 语句

Continue 语句用于 DO 循环中，终止当次循环返回 DO 循环开头开始下一次循环，实现循环“短路”。程序 3-59 用 input 语句将内嵌数据读入变量 name、idno 和 status，当 status 为'PT'（part time）时不执行后续的 input 命令，对应数据行不读入。

程序 3-59

```
data new_emp;
do i=1 to 5;
input name $ idno status $;
if status='PT' then continue;
input benefits $10.;
output;
end;
datalines;
Jones 9011 PT
Thomas 876 PT
Richards 1002 FT
Eye/Dental
Kelly 85111 PT
Smith 433 FT
HMO
;
run;
```

程序 3-59 中 DATA 步内置循环 1 次，DO 循环 5 次，在 status 取值'PT'的循环中，continue 的后续语句不被执行。

DO 循环中 Leave 语句使 DATA 步结束 DO 循环继续执行后续语句，实现 DO 循环的“跳出”。常用于基于条件的分支语句。程序 3-60 按每年 50 美元计算津贴，津贴总和超过 500 不再增加。

程序 3-60

```
data week;
input name $ start_yr 4. status $;
bonus=0;
do year= start_yr to 1991;
if bonus ge 500 then leave;
bonus+50;
end;
datalines;
赵璐  1990 PT
李刚 1976 PT
程昱 1991 FT
范海华 1975 FT
欧阳德生 1990 FT
;
run;
```

注：Leave 语句也用于 SELECT-WHEN 语句组，跳出当前的多分支语句组。

3.4.3 特殊流程控制

除了分支控制和循环控制外，DATA 还有一些特殊的流程控制语句，包括跳转语句 GOTO 和 LINK 以及 DATA 步终止语句 STOP 和 ABORT。

1．跳转语句 GOTO-标号

跳转语句将 DATA 步语句执行顺序引向标号标识的语句，常用在由条件控制的分支语句中，具体格式如下

GOTO 标号；

GOTO 语句必须和标号语句配合使用，标号语句定义一个符号，为 GOTO 语句指定跳转的目标语句。标号语句以标号:的形式放置在目标语句之前，与目标语句空格分隔：

标号：目标语句；

GOTO 语句与标号语句必须处于同一 DATA 步中。程序 3-61 用 input 语句将内嵌数据行数据读入变量 x，并对不同范围的 x 值进行求和。

程序 3-61

```
data a;
input x@@;
if 1<=x<=5 then goto ok;
put x;count+1;
ok:sum+x;
datalines;
1 2 7 2 12 32 11
;
run;
```

行指针控制符@@要求 input 语句下一次读取数据时不换行。标号语句为 OK:，语句跳转的结果是当 x>5 时 put 语句和累加语句 count+1 被跳过，而累加语句 sum+x 则是对 x 的所有取值进行了累加。

2．跳转语句 LINK-RETURN

Link 语句与 return 语句配合使用，使 DATA 步跳转到目标语句，执行目标语句及其后续语句直到 return 语句，return 语句使执行流程返回到 Link 语句的后续语句。程序 3-62 中有三处 link 语句，跳转到并执行目标语句将 test 取'E'的值替换为'F'，然后遇到 return 语句返回 link 语句的后续语句。

程序 3-62

```
data test;
input id test1 $ test2 $ test3 $;
test=test1;link recode;
test1=test;
test=test2;link recode;
test2=test;
test=test3;link recode;
test3=test;
recode:if test='E' then test='F';
return;
datalines;
1 A B E
2 E F C
3 C D A
4 A B A
5 B E E
;
```

```
run;
```

注：goto 语句也可以和 return 语句组合使用，但 return 语句使 SAS 返回到 DATA 步开头开始新的一次循环，而不是返回到 GOTO 语句的后续语句。程序 3-63 中的累加语句没有对 x>5 的值进行累加。

程序 3-63

```
data a;
input x@@;
if 1<=x<=5 then goto ok;
put x;count+1;
return;
ok:sum+x;
datalines;
1 2 7 2 12 32 11
;
run;
```

3．DATA 步的终止—STOP 和 ABORT 语句

STOP 语句结束所在 DATA 步的执行。程序 3-64 从数据集 sashelp.class 中随机（有放回）抽取 5 个观测。

程序 3-64

```
data class_5;
do i=1 to 5;
s=ranuni(123456)*19;
set sashelp.class point=s;
output;
end;
stop;
run;
```

stop 语句避免了 set 语句中 point=选项导致的死循环。

DATA 步中的 Abort 语句与 Stop 语句具有相同的功能，区别在于 Abort 执行后会使自动变量_ERROR_的值置为 1，并在信息窗口发布错误提示信息。

3.5 SAS 数据集排序

SAS 数据集的排序需要用到过程步 PROC SORT。排序数据集在 DATA 步打开后产生的临时变量能极大方便观测操作，况且数据集的匹配合并中要求合并数据集提前排序，这里提前对 PROC SORT 进行介绍。

3.5.1 排序过程—Proc sort

Proc sort（排序过程）的一般格式为

```
proc sort data=数据集 <out=数据集>;
by <descending> 变量 1 变量 2 …;
run;
```

功能是对 data=给出的数据集按 by 语句中的变量进行升序或降序排序，排序后的数据集存为 out=指定的数据集，如果没有给出 out=选项，则用排序后的数据集覆盖原数据集。程序 3-65 按 sex 降序对数据集 sashelp.class 进行降序（decending）排序，排序后的数据集为 work.class。

程序 3-65

```
proc sort data=sashelp.class out=class;
```

```
by descending sex;
run;
```

3.5.2 DATA 步中的排序数据集

要理解排序给 data 步处理数据带来的灵活性和便利性，需要首先理解 by 变量、by 值和 by 组的概念。

1．by 变量、by 值和 by 组

by 变量是指排序所依据的关键变量（key variable），即过程步 proc sort 中位于 by 语句之后的变量。by 值是指 by 变量的取值。by 组是指具有相同 by 值的观测形成的组。例如，数据集 class 按 sex 排序后，sex 为 by 变量，'男'和'女'为 by 值，男性观测和女性观测形成两个 by 组。

data 步中用 set 语句打开排序数据集并用 by 语句激活排序后，会产生两个临时变量 first.var 和 last.var 来标识 by 组，var 为 by 变量名。每个 by 组中第（最后）一个观测对应的 first.var（last.var）值为 1，其余观测对应的值为 0。变量 first.var 和 last.var 仅存在于所在 DATA 步内，并不输出到输出数据集，但可用于编程语句。程序 3-66 首先生成 3 只股票一周交易的数据集 work.test，然后用 proc sort 按 stkcd(股票代码)排序,最后在 DATA 步内用 set 语句打开数据集并用 by 语句激活排序，并用赋值语句将 first.stkcd 和 last.stkcd 的值赋值给变量 f 和 l。

程序 3-66

```
data test;
   input stkcd $ date yymmdd8. clp 6.2 @@;
   format date yymmddn8.;
   datalines;
      600013 20030310 5.23  600013 20030311 5.25
      600013 20030312 5.20  600013 20030313 5.40
      600013 20030314 5.10  600059 20030310 7.23
      600059 20030311 7.25  600059 20030312 7.20
      600059 20030313 7.40  600059 20030314 7.10
      600582 20030310 12.23 600582 20030311 12.25
      600582 20030312 12.20 600582 20030313 12.40
      600582 20030314 12.10
   ;
run;
proc sort;
   by stkcd;
run;
    data test1;
    set test;
    by stkcd;
    f=first.stkcd;l=last.stkcd;
    run;
```

数据集 test1 中的变量 f 和 l 标识出三个 by 组，见图 3.6。

stkcd	date	clp	f	l	
600013	20030310	5.23	1	0	by 组 1
600013	20030311	5.25	0	0	by 值 600013
600013	20030312	5.2	0	0	
600013	20030313	5.4	0	0	
600013	20030314	5.1	0	1	
600059	20030310	7.23	1	0	by 组 2
600059	20030311	7.25	0	0	by 值 600059
600059	20030312	7.2	0	0	
600059	20030313	7.4	0	0	
600059	20030314	7.1	0	1	
600582	20030310	12.23	1	0	by 组 3
600582	20030311	12.25	0	0	by 值 600582
600582	20030312	12.2	0	0	
600582	20030313	12.4	0	0	
600582	20030314	12.1	0	1	

图 3.6 排序数据集 by 组的标识

排序数据集的 by 组标识变量能给数据处理带来方便。程序 3-67 计算 sashelp.class 数据集中男女生体重总和（tw）和身高总和（th）并输出。

程序 3-67

```
proc sort data=sashelp.class;
by sex;
data c;
set sashelp.class;
by sex;
if first.sex then
do;
tw=0;th=0;
end;
tw+weight;th+height;
if last.sex then output;
keep sex tw th;
run;
```

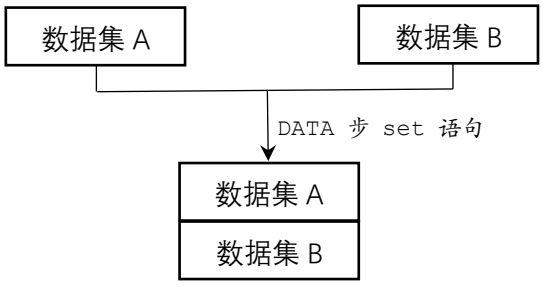
当first.var与last.var均取值为1时，表明by组只有一条观测。据此可以判断数据集内是否有相同的观测。

3.6 SAS 数据集合并

数据集的拆分和合并是常见的数据整理操作。数据集的拆分通过 3.3 节观测取子集和变量筛选来实现。本节主要对数据集的合并进行介绍，分为数据集的上下串接合并和左右并接合并。

3.6.1 数据集串接合并

数据集的串接是将两个或多个数据集的观测（行）上下"摞起来"的一种合并方式，用 set 语句实现，如下图所示。



串接合并分为非匹配合并和匹配合并。

❋ **非匹配合并**

非匹配合并（concatenating）按 set 语句中数据集出现的顺序，将后面的数据集观测续接在前面数据集观测下面，是一种按观测序号摆放观测的合并方法。set 语句的格式为

```
DATA 数据集;
  SET 数据集名列表;
RUN;
```

数据集名列表可采用缩略列示法。采用 SET 语句串接数据集时

- 编译阶段，SET 语句读入所有数据集描述部分，在 PDV 中安置所有变量，数据集变量按语句中数据集的次序排列；
- 执行阶段，DATA 步每次内置循环中，SET 语句依次读入各个输入数据集的观测。首先读

入第一个数据集的各条记录，读取一条新的记录时，PDV 中的数据集变量保留上次循环值。遇到第一个数据集的文件结束标志（end-of-file indicator）时，SAS 先将 PDV 中变量的值初始化为缺失，SAS 开始读取第二个数据集的观测，直到读取完所有数据集观测为止。

程序 3-68 先生成数据集 A 和 B，然后串接合并成数据集 C。

程序 3-68

```
data A(drop=i);
year=1991;
output;
do i=1 to 4;
year+1;
output;
end;
data B(drop=i);
year=1992;
output;
do i=1 to 4;
year+1;
output;
end;
data C;
set A B;
run;
```

此时 A 和 B 具有相同的变量 year，B 变量 year 的观测值直接在 A 中观测值下面。程序 3-69 将 A 和 B 中的变量 year 分别改名为 year1 和 year2 生成数据集 D 和 F，然后串接合并成 E。

程序 3-69

```
data D;
set A;
rename year=year1;
data E;
set B;
rename year=year2;
run;
data F;
set D E;
run;
```

D和E中具有不同的变量year1和year2，读取D的观测时，PDV中year2值为缺失值，读取E的观测时PDV中year1为缺失值。

A

year
1991
1992
1993
1994
1995

B

year
1992
1993
1994
1995
1996

C

year
1991
1992
1993
1994
1995
1992
1993
1994
1995
1996

D

year1
1991
1992
1993
1994
1995

E

year2
1992
1993
1994
1995
1996

F

year1	year2
1991	.
1992	.
1993	.
1994	.
1995	.
.	1992
.	1993
.	1994
.	1995
.	1996

❋ 匹配合并

匹配合并也称为穿插（interleaving）合并，是将多个数据集观测按 by 变量（key variable）值进行匹配的合并方式。匹配合并的数据集需要有同名变量，并按同名变量排序，同名变量必须是相同数据类型。语句格式为

```
DATA 数据集;
   SET 数据集名列表;
   BY 变量 1 变量 2 … ;
RUN;
```

生成数据集中的观测按 by 值排序，同一 by 组的观测按其数据集在 set 语句中的顺序排列。DATA 步流程为

■ 编译阶段

第一步：SET 语句读入所有数据集描述部分，在 PDV 中安置所有变量，数据集变量按语句中数据集的次序排列；

第二步：为每个 by 变量创建临时变量 First.nvar 和 Last.nvar。

■ 执行阶段

第一步：比较 set 语句中所有数据集的第一个观测值，确定哪个数据集 by 组首先输出到生成的数据集中，并读取 by 组所有观测。如果 by 组出现在多个数据集，则按数据集在 set 语句中的顺序读取所有 by 组观测。当开始读取新的数据集或者新的 by 组时，PDV 中变量值初始化为缺失值。

第二步：比较所有数据集的下一条观测确定下一个读取的 by 组，并读取 by 组所有观测。以此操作直到读取完所有数据集的所有观测。

程序 3-70 将数据集 animal 和 plant 进行匹配合并，两个数据集都有变量 common 并按此变量排序。

程序 3-70

```
data comb;
set animal plant;
by common;
run;
```

合并前后的数据集见下图

Animal

common	animal
a	Ant
a	dog
b	Bird
c	Cat

Plant

common	plant
a	Apple
b	Banana
c	coconut
f	Elggplan

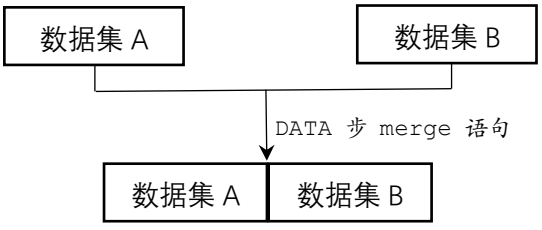
Comb

common	animal	plant
a	Ant	
a	dog	
a		Apple
b	Bird	
b		Banana
c	Cat	
c		coconut
f		Elggplan

比较两个数据集第一条观测时确定读入 common 值（by 值）为'a'的 by 组，先读入 Animal 中观测（两条），然后读入 Plant 中的观测（一条）。以此读取全部观测，实现穿插合并。

3.6.2 数据集并接合并

数据集并接（merge）合并将多个数据集的变量（列）横向合并成新的数据集，通过 DATA 步的 merge 语句实现。示意图如下



并接合并分为非匹配合并和匹配合并。

1. 非匹配合并

非匹配合并也称为一对一合并（one-to-one merging），语句格式为

```
DATA 数据集;
   MERGE 数据集名列表;
RUN;
```

DATA 步流程为

- 编译阶段

  merge 语句读入所有数据集描述部分，在 PDV 中安置所有变量，数据集变量按语句中数据集的次序排列；

- 执行阶段

  SAS 顺次读取每个数据集的第一条观测。若两个数据集中有同名变量，后读入的变量值覆盖先前读入的变量值。同名变量的数值类型必须相同，长度按第一次遇到的变量为准。合并生成的数据集观测个数与 merge 语句中观测个数最多的数据集相等。读取到观测个数少的数据集文件结束标志后，PDV 中对应的变量值取缺失值。

数据集 gdp 和 cpi 分别是美国 1947 年 3 月到 2010 年 9 月的名义国内生产总值和消费价格指数数据，两个数据集都包含变量 date，gdp 中变量 date 是每个季度第一个月的 1 号，cpi 中的变量 date 是每个季度最后一个月的 1 号。程序 3-71 将两个数据集非匹配并接合并为 gdp_cpi。

程序 3-71

```
data gdp_cpi;
merge gdp cpi;
run;
```

后读入的 cpi 中 date 变量值覆盖先读入的 gdp 中 date 变量值。

2. 匹配合并

用 merge 语句匹配合并多个数据集时，数据集需要有同名变量，且按该变量排序，同名变量必须是相同数据类型。命令一般形式：

```
DATA 数据集;
   MERGE 数据集名列表;
   BY 变量 1 变量 2 … ;
RUN;
```

SAS 的处理步骤为：

- 编译阶段

  MERGE 语句读入所有数据集描述部分，将所有变量置于 PDV。若不同的数据集中有同名变量，长度按第一次遇到的变量为准；为每个 by 变量创建临时变量 First.vname 和 Last.vname。

- 执行阶段

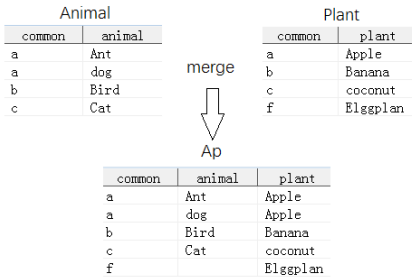
  检查每个数据集的第一个 by 组以确定首先读取哪个 by 组，然后读入每个数据集中该 by 组的第一条观测，按数据集名列表顺序读取。如果数据集在该 by 组中没有观测，则其变量在 PDV 中取缺失值。读取完第一条观测并执行完其它 DATA 语句后，SAS 将 PDV 中变量值输出到新生成数据集并开始下一次循环，此时所有数据集变量值在 PDV 中的值保留。以此操作读取该 by 组全部观测后将 PDV 中数据集变量值初始化为缺失。依次读取下一个 by 组，直至读取完所有数据集观测。

  程序 3-72 以匹配方式并接合并数据集 animal 和 plant

  程序 3-72

```
proc sort data=course.animal;
by common;
proc sort data=course.plant;
by common;
run;
data ap;
merge course.animal course.plant;
by common;
run;
```

生成数据集 ap 如下图



读取第一个 by 组第一条观测后，变量 plant 在 PDV 中的值' Apple'得以保留，第二次循环 animal 读取值'dog'时，plant 值仍然为'Apple'。读取 by 值为'f'的 by 组时，Animal 中变量 animal 没有观测值，PDV 中取缺失值。

3.6.3 用多个 set 语句合并

DATA 步中的多个 set 语句在同一个内置循环中将独立打开各自的数据集，并且具有各自的记录指针。用多个 set 语句及其指针控制等选项，可以更为灵活地合并数据集。

1. 一对一合并

采用多个 set 语句可以实现数据集的非匹配并接合并。程序 3-73 将数据集 gdp 和 cpi 进行并接合并。

程序 3-73

```
data gdp_cpi_1;
set course.gdp;
set course.cpi;
run;
```

与程序 3-71 合并结果比较，会发现多个 set 语句合并和非匹配 merge 语句合并的区别。只要有一个 set 语句读取到其数据集的文件结束标志，DATA 就结束循环，形成的数据集观测个数等于合并数据集中观测最少的数据集。

2．一对多合并

采用条件语句控制观测读取，可以实现数据集观测一对多合并。数据集 c_Num 有一个变量 No，表示学生的班级编号。程序 3-74 将变量 No 的第一个观测值与 sashelp.class 中的所有观测进行合并。

程序 3-74

```
data c_Num(drop=i);
no=0;
do i=1 to 7;
no=i;
output;
end;
data class_num;
if _n_=1 then set c_num;
set sashelp.class;
run;
```

第二个 DATA 步中，第一个 set 语句只读取了 c_num 的第一条观测，且在每次 DATA 步内置循环中保留了该值，实现了与 sashelp.class 每个观测的匹配。

3．多对多合并

采用数据集指针控制选项 point=选项和 if 语句，可以实现一个数据集的子集观测和另一个数据集观测的匹配并接。程序 3-75 将 sashelp.class 中 name 以'罗'开头的观测与 c_num 中 no 变量的第 7、8 和 10 个观测匹配，并接两个数据集。

程序 3-75

```
data class_num1;
set sashelp.class;
if name=:'罗';
set c_num point=_n_;
run;
```

解读：if 语句条件不满足时，后面的语句不再执行，DATA 步"短路"！在前 6 次循环中第二个 set 语句均未执行，第 7、8 次循环执行，直到第 18 次循环前均未执行，第 18 次执行，第 19 次未被执行。但是第 7 次循环后_n_>7，第二个 set 语句选项 point=_n_指向的观测序号已超出 c_mum 观测序号范围，不能读取数据，PDV 中对应的变量值不被更新，并保留至（retain）下一次循环。

将 if 语句的位置下移后会得出不同结果（见作业 23），据此体会 if 语句的功能。

习题

1.表达式 3+'2'、3||'2'正确吗？可行吗？表达式 3+'a'正确吗？可行吗？'2018/05/03'd 是正确的日期常量表达吗？'3Feb19'd 是正确的日期常量表达吗？SA 将其中的'19'识别为世纪还是年份？

2.如果 DATA 步中出现赋值语句 y=sim(x);，信息窗口会出现什么错误提示？SAS 将 sim(x)识别为一个变量还是函数？为什么？

3.通过查阅帮助,详细说明函数 logistic()、fact()、substr()=、exist()和 Yrdif()的功能和使用方法。

4.将 SAS 数据集 sashelp.class 中 name 以'罗'开头的观测挑选出来,if 语句如何写？if name='罗';可以吗？where 语句如何写？还有其它方法吗？语句 where name not like '%德';选出的是哪些观测？

5.什么是空语句？空语句能作为 DATA 步的结束边界吗？

6.DATA 步中的自动变量_Error_和_N_的初次赋值（_Error=0_,_N_=1）是在编译阶段还是在执行阶段？你能编程序验证码？

7.什么是 DATA 步中的声明性语和执行性语句？举出三个声明性语句？

8.指出 DATA 步程序

```
Data test;
set sashelp.class(keep=name sex);
h=height*0.025;
run;
```

中存在的错误并进行修改。指出 DATA 步程序

```
Data test;
set sashelp.class;
h=height*0.025;
where h<10;
run;
```

中存在的错误并进行修改。

9.程序

```
data s;
if sex='男';
```

```
set sashelp.class;
h=height*0.025;
run;
```

存在语法错误吗？将取子集 if 语句置于 set 语句之前和之后得到的结果有什么区别？为什么？

10.如何用 input 函数将字符'03APR2019'转换为日期常量？能将'03APR2019'd 转为日期常量吗？如何将'03APR2019'd 转换为字符常量？

11.写出 DATA 步程序用数据集 work.test

VIEWTABLE: Work.Test

	city	year
1	上海	2017
2	上海	2018
3	上海	2019

生成数据集 work.test1

VIEWTABLE: Work.Test1

	city	year1	year2	year3
1	上海	2017	2018	2019

12.在程序不同位置添加语句 put _all_可以查看程序执行不同阶段 PDV 中变量取值的变化。运行程序

```
data test;
put _all_;
set sashelp.class;
put _all_;
run;
```

查看信息窗口的内容，解释

(1)每条观测都出现两次，两次的内容有什么不同？

(2)为什么最后一条观测的第二次显示中，_N_的值是 20？DATA 步内置循环是 20 次吗？如果是 20 次为什么只有 19 条记录输出？你能据此判断在从本次循环到下次循环的过程中，_N_的值是在哪个环节增加的吗？如果将程序改为

```
data test;
put _all_;
set sashelp.class;
_n_=3;
put _all_;
run;
```

怎么解释信息窗口看到的_N_值的变化？

13.提交并运行程序

```
data _null_;
set sashelp.class;
rename height=h;
keep name age sex;
run;
```

信息窗口为什么没有出现运行程序 3-31 的警告（warning）信息？

14.用 output 语句在数据集 sashelp.class 第一条观测前添加一条空白记录。用 output 语句和 if-then-语句在第 5 条观测后添加一条空白记录（提示：可使用自动变量_n_或者 set 语句选项 curobs=）。

15.文件 sashelp.class 中有 19 条观测，对应一个班级中 19 位学生的信息，其中变量 age 表示学生年龄。针对每位学生，计算班级中年龄大于该生的学生人数。提示：采用 do-end 循环语句、output 语句、rename=()数据集选项以及带 point=选项和 nobs=选项的 set 语句等。

16.用 output 语句和 DO 循环语句在数据集 sashelp.class 第 5 条观测前添加一条空白记录生成数据集 work.c_5。为什么 work.c_5 中变量 i 的值从第 7 条记录开始为缺失值？

17.为理解 stop 语句和 set 语句 point=选项，运行程序

```
data x;
do i=1 to 25;
set sashelp.class point=i;
output;
end;
stop;
run;
```

（1） 如果不带 stop 语句将会如何？

（2） 为什么数据集 x 的第 19 到第 25 条观测完全相同？

18. 运行程序

```
data x1;
n=23;
set sashelp.class point=n;
stop;
run;
```

为什么数据集 x1 中没有观测？如果在 set 语句添加语句 output 会得出什么步同结果？为什么？

19.执行程序

```
data test;
if 0 then
do;
set sashelp.class(drop=name sex);
retain sex name;
end;
set sashelp.class;
run;
```

（1） test 数据集与 sashelp.class 由什么区别？

（2） 程序中的 do-end 程序块中的语句执行了吗？这些语句的作用是什么？

（3） 为什么需要第二个 set 语句？

20.对程序 3-66 进行修改，在数据集 work.test 中每只股票的最后一条观测后和下一只股票第一条观测前添加一条空白记录。

21.将数据集 rate 和 GDP、cpi 合并，要求

（1）从 cpi 中抽取 3/6/9/12 月份的观测作为季度观测值；

（2）从 rate 中抽取每个月份最后一个交易日数据作为月份交易数据，再从月份数据中抽取 3/6/9/12 月份的观测作为季度观测值

（3）实施合并。

22.将 Animal 和 Plant 进行非匹配并接合并（merge），将结果与程序-70 的合并结果比较，据此说明两种并接合并的区别。

23.将数据集c_Num变量No的最后一个观测值和sashelp.class的所有观测匹配合并。

24.将程序 3-75 中的 if 语句的位置改变为

```
data class_num2;
set sashelp.class;
set c_num point=_n_;
if name=:'罗';
run;
```

会得出什么结果，为什么？