第十章 格式化输出

by flamephoenix

我们已经见过用print函数将原始的未格式化的文本输出到文件，本章讲述如何用函数write和打印格式来生成格式化的输出。

二、显示打印格式

打印格式的显示有两步：

1、将系统变量$~设成所要使用的格式

2、调用函数write

例如：

```
#!/usr/local/bin/perl

$~ = "MYFORMAT";
write;

format MYFORMAT =
===================================
Here is the text I want to display.
===================================
.
```

结果输出如下：

```
$ program
===================================
Here is the text I want to display.
===================================
$
```

如果不用$~指定打印格式，Perl解释器就假定要使用的格式名与要写入的文件变量同名，在本例中，如果不指定使用MYFORMAT，则Perl解释器试图使用名为STDOUT的打印格式。

三、在打印格式中显示值

我们使用打印格式的主要原因当然是格式化存贮在简单变量或数组变量中的值从而生成可读性好的输出，这一目的用“值域”来实现。每个值域指定一个值，如变量或表达式，调用write函数时，该值就以值域指定的格式显示。

1、通用的打印格式

打印格式的一个缺点是定义中包含了变量名，例如：

```
format MYFORMAT =
===========================================================
```

```
The winning number is @<<<<<<!
$winnum
==========================================================
.
```

当调用write输出此格式时，必须记着它使用了变量$winnum。用子程序和局域变量就可以创建更通用的打印格式。下例从STDIN输入一个文件并输出五个出现频率最高的字母及出现次数。

```
                1 : #!/usr/local/bin/perl
2 :
3 : while ($line = ) {
4 :   $line =~ tr/A-Z/a-z/;
5 :   $line =~ s/[^a-z]//g;
6 :   @letters = split(//, $line);
7 :   foreach $letter (@letters) {
8 :     $lettercount{$letter} += 1;
9 :   }
10: }
11:
12: $~ = "WRITEHEADER";
13: write;
14: $count = 0;
15: foreach $letter (reverse sort occurrences
16:       (keys(%lettercount))) {
17:   &write_letter($letter, $lettercount{$letter});
18:   last if (++$count == 5);
19: }
20:
21: sub occurrences {
22:   $lettercount{$a} <=> $lettercount{$b};
23: }
24: sub write_letter {
25:   local($letter, $value) = @_;
26:
27:   $~ = "WRITELETTER";
28:   write;
29: }
30: format WRITEHEADER =
31: The five most frequently occurring letters are:
32: .
33: format WRITELETTER =
34:   @: @<<<<<<
35:   $letter, $value
36: .
```

运行结果如下:

```
            $ program
This is a test file.
This test file contains some input.
The quick brown fox jumped over the lazy dog.
^D
The five most frequently occurring letters are:
         t: 10
         e: 9
```

```
i:  8
s:  7
o:  6
$
```

2、格式和局域变量

在上例中，你可能已经注意到子程序write_letter调用write输出字母及其出现次数，即使格式定义在子程序外部仍能正常工作。在第17行中将字母及其出现次数传递给该子程序，在子程序中，打印格式使用局域变量$letter和$value，这样保证了在foreach循环中每次输出当前的字母和值。

然而要注意的是，使用my定义的局域变量要求格式定义在子程序内部，否则就不会输出，因此，用write输出的局域变量一定要用local定义。（local和my详见《子程序》一章）

注：Perl4中没有my函数，故不会有此问题。

3、选择值域格式

我们已经知道了打印格式和write函数怎么工作，现在来看看值域的格式，见下表：

格式	值域含义
@<<<	左对齐输出
@>>>	右对齐输出
@|||	中对齐输出
@##.##	固定精度数字
@*	多行文本

每个值域的第一个字符是行填充符，当使用@字符时，不做文本格式化。对文本的格式化稍后来讲。

在上表中，除了多行值域@*，域宽都等于其指定的包含字符@在内的字符个数，例如：

@###.##

表示七个字符宽，小数点前四个，小数点后两个。

4、输出值域字符

在打印格式里，特定字符如@、<和>被看作值域定义，那么如何将它们输出呢？方法如下：

```
format SPECIAL =
This line contains the special character @.
"@"
.
```

四、输出到其它文件

缺省地，函数write将结果输出到标准输出文件STDOUT，我们也可以使它将结果输出到任意其它的文件中。最简单的方法就是把文件变量作为参数传递给write，如：

```
write (MYFILE);
```

这样，write就用缺省的名为MYFILE的打印格式输出到文件MYFILE中，但是这样就不能用$~变量来改变所使用的打印格式。系统变量$~只对缺省文件变量起作用，我们可以改变缺省文件变量，改变$~，再调用write，例如：

```
select (MYFILE);
$~ = "MYFORMAT";
write;
```

当select改变缺省文件变量时，它返回当前缺省文件变量的内部表示，这样我们就可以创建子程序，按自己的想法输出，又不影响程序的其它部分，如下：

```
sub write_to_stdout {
    local ($savefile, $saveformat);
    $savefile = select(STDOUT);
    $saveformat = $~;
    $~ = "MYFORMAT";
    write;
    $~ = $saveformat;
```

```
    select($savefile);
  }
```

五、分页

在输出到打印机时，可以在每页顶部输出相应的信息，这样的特殊文本叫页眉。定义页眉实际上就是定义名为filename_TOP的打印格式，例如给标准输出文件定义页眉如下：

```
format STDOUT_TOP =
Consolidated Widgets Inc. 1994 Annual Report
.
```

在页眉的定义中也可以包含值域，页眉中经常使用的一个特殊值是当前页码，存贮在系统变量$%中，如：

```
format STDOUT_TOP =
Page @<<.
$%
.
```

我们也可以通过改变系统变量$^改变定义页眉的打印格式名，与$~一样，$^只对当前缺省文件起作用，因此可以与select函数结合使用。

缺省情况下，每页长度为60行，可以通过改变$=来改变页长，如：

```
$= = 66; #页长设为66行
```

此赋值语句必须出现在第一个write语句前。

注：一般使用分页机制时不用print函数，因为当用write输出时，Perl解释器跟踪每页的当前行号。如果必须使用print而又不打乱页计数，可以调整系统变量$-。$-的含义是当前行到页末之间的行数，当$-达到零时，就开始新的一页，调整方法如：

```
print ("Here is a line of output\n");
$- -= 1;
```

六、格式化长字符串

我们已经学过值域@*可以输出多行文本，但它完全将字符串原样输出，不加以格式化。在Perl中对长字符串（包含换行）进行格式化的值域定义很简单，只需把打头的@字符换成^就行了，这种文本格式化中，Perl解释器在一行中放置尽可能多的单词。每当输出一行文本，被输出的子串就从变量中删除，再次在域值中使用该变量就把剩下的字符串继续按格式输出。当内容已输出完毕，该变量就成了空串，再输出就会输出空行，为避免输出空行，可以在值域格式行首加一个~字符。见下例：

```
                1 : #!/usr/local/bin/perl
  2 :
  3 : @quotation = <STDIN>;
  4 : $quotation = join("", @quotation);
  5 : $~ = "QUOTATION";
  6 : write;
  7 :
  8 : format QUOTATION =
  9 : Quotation for the day:
 10: ----------------------------
 11: ~  ^<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
 12: $quotation
 13: ~  ^<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
 14: $quotation
 15: ~  ^<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
 16: $quotation
 17: ----------------------------
 18: .
```

运行结果如下：

```
                $ program
  Any sufficiently advanced programming
  language is indistinguishable from magic.
```

```
^D
Quotation for the day:
---------------------------
  Any sufficiently advanced programming language is
  indistinguishable from magic.
---------------------------
$
```

如果把打印格式中行首的~字符去掉，上面的输出结果中就会多一行空行。很明显，当字符串长度不明确时，这种用法很不方便，原因就在于它指明了输出的行数上限，超过这一上限的字符就不会被输出，解决方法很简单，就是在域值格式行首加两个~字符，这样就会持续按格式输出文本直到输出完毕，用此方法把上述程序改写如下：

```
1 : #!/usr/local/bin/perl
2 :
3 : @quotation = <STDIN>;
4 : $quotation = join("", @quotation);
5 : $~ = "QUOTATION";
6 : write;
7 :
8 : format QUOTATION =
9 : Quotation for the day:
10: ---------------------------
11: ~~ ^<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
12: $quotation
13: ---------------------------
14: .
```

这样运行结果相同。

七、用printf格式化输出

还有一种容易掌握和使用的格式化输出函数，那就是printf，它与C语言中的printf基本上是相同的。printf有两个参数，一个是字符串，其中含有一个或多个域值形式，另一个是与各域值相对应的变量值按一定格式替换，如：

```
printf("The number I want to print is %d.\n", $number);
```

各种域值形式如下表：

域值	含义
%c	单个字符
%d	十进制整数
%e	科学计数法形式的浮点数
%f	普通形式（定点）浮点数
%g	紧缩形式浮点数
%o	八进制整数
%s	字符串
%u	无符号整数
%x	十六进制整数

一些使用细节如下：

1、在格式d、o、u或x中，如果整数值较大或可能较大，可加个l字符，意为长整型，如%ld。

2、%字符后加正整数表示该域的最小宽度，如果输出结果宽度不足，则向右对齐，前面用空格补足，如果该正整数以数字0打头，则补足字符为0。若%字符后为负整数，则结果向右对齐。

3、浮点数域值(%c、%f和%g)中可以指定小数点前后的宽度，如%8.3f意为总宽度为8个字符，

小数点后（即小数部分）为3个字符，多出的小数部分四舍五入。
4、在整数、字符或字符串的值域中使用如上的小数形式n.m，整数部分n为总宽度，小数部分m为输出结果的最大宽度，这样就保证了输出结果前至少有n-m个空格。