第6章 gawk语言编程

awk是一种程序语言，对文档资料的处理具有很强的功能。 awk 名称是由它三个最初设计者的姓氏的第一个字母而命名的：Alfred V. Aho、 Peter J. Weinberger、 Brian W. Kernighan。awk最初在1977年完成。1985年发表了一个新版本的awk，它的功能比旧版本增强了不少。awk能够用很短的程序对文档里的资料做修改、比较、提取、打印等处理。如果使用 C 或 Pascal等语言编写程序完成上述的任务会十分不方便而且很花费时间，所写的程序也会很大。

awk不仅仅是一个编程语言，它还是 Linux系统管理员和程序员的一个不可缺少的工具。awk语言本身十分好学，易于掌握，并且特别的灵活。

gawk 是GNU计划下所做的 awk，gawk 最初在1986年完成，之后不断地被改进、更新。gawk 包含 awk 的所有功能。

6.1 gawk的主要功能

gawk 的主要功能是针对文件的每一行 (line)，也就是每一条记录，搜寻指定的格式。当某一行符合指定的格式时，gawk 就会在此行执行被指定的动作。gawk 依此方式自动处理输入文件的每一行直到输入文件档案结束。

gawk经常用在如下的几个方面：

- 根据要求选择文件的某几行，几列或部分字段以供显示输出。
- 分析文档中的某一个字出现的频率、位置等。
- 根据某一个文档的信息准备格式化输出。
- 以一个功能十分强大的方式过滤输出文档。
- 根据文档中的数值进行计算。

6.2 如何执行gawk程序

基本上有两种方法可以执行gawk程序。

如果gawk 程序很短，则可以将gawk 直接写在命令行，如下所示：

```
gawk 'program' input-file1 input-file2 ...
```

其中 program 包括一些 pattern 和action。

如果gawk 程序较长，较为方便的做法是将 gawk 程序存在一个文件中，

gawk 的格式如下所示：

```
gawk -f program-file input-file1 input-file2 ...
```

gawk 程序的文件不止一个时，执行gawk 的格式如下所示：

```
gawk -f program-file1 -f program-file2 ... input-file1 input-file2 ...
```

6.3 文件、记录和字段

一般情况下，gawk可以处理文件中的数值数据，但也可以处理字符串信息。如果数据没有存储在文件中，可以通过管道命令和其他的重定向方法给 gawk提供输入。当然， gawk只能处理文本文件（ASCII码文件）。

电话号码本就是一个gawk可以处理的文件的简单例子。电话号码本由很多条目组成，每一个条目都有同样的格式：姓、名、地址、电话号码。每一个条目都是按字母顺序排列。

在gawk中，每一个这样的条目叫做一个记录。它是一个完整的数据的集合。例如，电话号码本中的Smith John这个条目，包括他的地址和电话号码，就是一条记录。

记录中的每一项叫做一个字段。在gawk中，字段是最基本的单位。多个记录的集合组成了一个文件。

大多数情况下，字段之间由一个特殊的字符分开，像空格、 TAB、分号等。这些字符叫做字段分隔符。请看下面这个/etc/passwd文件：

```
tparker;t36s62hsh;501;101;Tim Parker;/home/tparker;/bin/bash
etreijs;2ys639dj3h;502;101;Ed Treijs;/home/etreijs;/bin/tcsh
ychow;1h27sj;503;101;Yvonne Chow;/home/ychow;/bin/bash
```

你可以看出/etc/passwd文件使用分号作为字段分隔符。 /etc/passwd文件中的每一行都包括七个字段：用户名；口令；用户 ID；工作组ID；注释；home目录；启始的外壳。如果你想要查找第六个字段，只需数过五个分号即可。

但考虑到以下电话号码本的例子，你就会发现一些问题：

```
Smith John 13 Wilson St. 555-1283
Smith John 2736 Artside Dr Apt 123 555-2736
Smith John 125 Westmount Cr 555-1726
```

虽然我们能够分辨出每个记录包括四个字段，但gawk却无能为力。电话号码本使用空格作为分隔符，所以gawk认为Smith是第一个字段， John 是第二个字段，13是第三个字段，依次类推。就 gawk而言，如果用空格作为字段分隔符的话，则第一个记录有六个字段，而第二个记录有八个字段。

所以，我们必须找出一个更好的字段分隔符。例如，像下面一样使用斜杠作为字段分隔符：

```
Smith/John/13 Wilson St./555-1283
Smith/John/2736 Artside Dr/Apt/123/555-2736
Smith/John/125 Westmount Cr/555-1726
```

如果你没有指定其他的字符作为字段分隔符，那么 gawk将缺省地使用空格或TAB作为字段分隔符。

6.4 模式和动作

在gawk语言中每一个命令都由两部分组成：一个模式（ pattern）和一个相应的动作(action)。只要模式符合， gawk就会执行相应的动作。其中模式部分用两个斜杠括起来，而动作部分用一对花括号括起来。例如：

```
/pattern1/{action1}
/pattern2/{action2}
/pattern3/{action3}
```

所有的gawk程序都是由这样的一对对的模式和动作组成的。其中模式或动作都能够被省略，但是两个不能同时被省略。如果模式被省略，则对于作为输入的文件里面的每一行，动作都会被执行。如果动作被省略，则缺省的动作被执行，既显示出所有符合模式的输入行而不做任何的改动。

下面是一个简单的例子，因为 gawk 程序很短，所以将 gawk 程序直接写在外壳命令行：

```
gawk '/tparker/' /etc/passwd
```

此程序在上面提到的/etc/passwd文件中寻找符合tparker模式的记录并显示（此例中没有动作，所以缺省的动作被执行）。

让我们再看一个例子：

```
gawk '/UNIX/{print $2}' file2.data
```

此命令将逐行查找file2.data文件中包含UNIX的记录，并打印这些记录的第二个字段。

你也可以在一个命令中使用多个模式和动作对，例如：

```
gawk '/scandal/{print $1} /rumor/{print $2}' gossip_file
```

此命令搜索文件 gossip_file中包括scandal的记录，并打印第一个字段。然后再从头搜索gossip_file中包括rumor的记录，并打印第二个字段。

6.5 比较运算和数值运算

gawk有很多比较运算符，下面列出重要的几个：

==	相等
!=	不相等
>	大于
<	小于
>=	大于等于
<=	小于等于

例如：

```
gawk '$4 > 100' testfile
```

将会显示文件testfile 中那些第四个字段大于100的记录。

下表列出了gawk中基本的数值运算符。

运 算 符	说 明	示 例
+	加法运算	2+6
-	减法运算	6-3
*	乘法运算	2*5
/	除法运算	8/4
^	乘方运算	3^2 (=9)
%	求余数	9%4 (=1)

例如：

```
{print $3/2}
```

显示第三个字段被2除的结果。

在gawk中，运算符的优先权和一般的数学运算的优先权一样。例如：

```
{print $1+$2*$3}
```

显示第二个字段和第三个字段相乘，然后和第一个字段相加的结果。

你也可以用括号改变优先次序。例如：

```
{print ($1+$2)*$3}
```

显示第一个字段和第二个字段相加，然后和第三个字段相乘的结果。

6.6 内部函数

gawk中有各种的内部函数，现在介绍如下：

6.6.1 随机数和数学函数

sqrt(x)	求x 的平方根
sin(x)	求x 的正弦函数
cos(x)	求x 的余弦函数
atan2(x, y)	求x/y的余切函数
log(x)	求x 的自然对数
exp(x)	求x 的e 次方
int(x)	求x 的整数部分
rand()	求0 和1之间的随机数
srand(x)	将 x 设置为 rand()的种子数

6.6.2 字符串的内部函数

- index(in, find) 在字符串 in 中寻找字符串 find 第一次出现的地方，返回值是字符串 find 出现在字符串 in 里面的位置。如果在字符串 in 里面找不到字符串 find，则返回值为 0。

例如：

```
print index("peanut", "an")
```

显示结果3。

- length(string) 求出 string 有几个字符。

例如：

```
length("abcde")
```

显示结果5。

- match(string, regexp) 在字符串 string 中寻找符合 regexp 的最长、最靠左边的子字符串。返回值是 regexp 在 string 的开始位置，即 index值。match 函数将会设置系统变量 RSTART 等于index的值，系统变量RLENGTH 等于符合的字符个数。如果不符合，则会设置 RSTART 为0、RLENGTH 为 -1。
- sprintf(format, expression1, ...) 和printf 类似，但是 sprintf 并不显示，而是返回字符串。

例如：

```
sprintf("pi = %.2f (approx.)", 22/7)
```

返回的字符串为pi = 3.14 (approx.)

- sub(regexp, replacement, target) 在字符串 target 中寻找符合 regexp 的最长、最靠左的地方，以字串 replacement 代替最左边的 regexp。

例如：

```
str = "water, water, everywhere"
sub(/at/, "ith", str)
```

结果字符串str会变成

```
wither, water, everywhere
```

- gsub(regexp, replacement, target) 与前面的sub类似。在字符串 target 中寻找符合 regexp的所有地方，以字符串 replacement 代替所有的regexp。

例如：

```
str="water, water, everywhere"
```

gsub(/at/, "ith", str)

结果字符串str会变成

wither, wither, everywhere

• substr(string, start, length) 返回字符串 string 的子字符串，这个子字符串的长度为 length，从第 start 个位置开始。

例如：

substr("washington", 5, 3)

返回值为ing

如果没有length ，则返回的子字符串是从第 start 个位置开始至结束。

例如：

substr("washington", 5)

返回值为ington。

• tolower(string) 将字符串string的大写字母改为小写字母。

例如：

tolower("MiXeD cAsE 123")

返回值为mixed case 123。

• toupper(string) 将字符串string的小写字母改为大写字母。

例如：

toupper("MiXeD cAsE 123")

返回值为MIXED CASE 123。

6.6.3 输入输出的内部函数

• close(filename) 将输入或输出的文件 filename 关闭。

• system(command) 此函数允许用户执行操作系统的指令，执行完毕后将回到 gawk程序。

例如：

BEGIN {system("ls")}

6.7 字符串和数字

字符串就是一连串的字符，它可以被gawk逐字地翻译。字符串用双引号括起来。数字不能用双引号括起来，并且gawk将它当作一个数值。例如：

gawk '$1 != "Tim" {print}' testfile

此命令将显示第一个字段和 Tim不相同的所有记录。如果命令中 Tim两边不用双引号，gawk将不能正确执行。

再如：

gawk '$1 == "50" {print}' testfile

此命令将显示所有第一个字段和 50这个字符串相同的记录。 gawk不管第一字段中的数值的大小，而只是逐字地比较。这时，字符串50和数值50并不相等。

6.8 格式化输出

我们可以让动作显示一些比较复杂的结果。例如：

gawk '$1 != "Tim" {print $1, $5, $6, $2}' testfile

将显示testfile文件中所有第一个字段和 Tim不相同的记录的第一、第五、第六和第二个字段。

进一步，你可以在print动作中加入字符串，例如：

```
gawk '$1 != "Tim" {print "The entry for ", $1, "is not Tim. ", $2}' testfile
```

print动作的每一部分用逗号隔开。

借用C语言的格式化输出指令，可以让gawk的输出形式更为多样。这时，应该用printf而不是print。例如：

```
{printf "%5s likes this language\n", $2}
```

printf中的%5s 部分告诉gawk 如何格式化输出字符串，也就是输出 5个字符长。它的值由printf 的最后部分指出，在此是第二个字段。 \n是回车换行符。如果第二个字段中存储的是人名，则输出结果大致如下：

```
Tim likes this language
Geoff likes this language
Mike likes this language
Joe likes this language
```

gawk 语言支持的其他格式控制符号如下：

- c 如果是字符串，则显示第一个字符；如果是整数，则将数字以ASCII 字符的形式显示。

例如：

printf “ %c ”, 65

结果将显示字母A。

- d 显示十进制的整数。
- i 显示十进制的整数。
- e 将浮点数以科学记数法的形式显示。

例如：

print “ $4.3e ”, 1950

结果将显示1.950e+03。

- f 将数字以浮点的形式显示。
- g 将数字以科学记数法的形式或浮点的形式显示。数字的绝对值如果大于等于 0.0001则以浮点的形式显示，否则以科学记数法的形式显示。
- o 显示无符号的八进制整数。
- s 显示一个字符串。
- x 显示无符号的十六进制整数。10至15以a至f表示。
- X 显示无符号的十六进制整数。10至15以A至F表示。
- % 它并不是真正的格式控制字符，%%将显示%。

当你使用这些格式控制字符时，你可以在控制字符前给出数字，以表示你将用的几位或几个字符。例如，6d表示一个整数有6位。再请看下面的例子：

```
{printf "%5s works for %5s and earns %2d an hour", $1, $2, $3}
```

将会产生类似如下的输出：

```
Joe works for Mike and earns 12 an hour
```

当处理数据时，你可以指定数据的精确位数

```
{printf "%5s earns $%.2f an hour", $3, $6}
```

其输出将类似于：

```
Joe earns $12.17 an hour
```

你也可以使用一些换码控制符格式化整行的输出。之所以叫做换码控制符，是因为 gawk对这些符号有特殊的解释。下面列出常用的换码控制符：

\a 警告或响铃字符。
\b 后退一格 。
\f 换页。
\n 换行。
\r 回车。
\t Tab。
\v 垂直的 tab。

6.9 改变字段分隔符

在gawk中，缺省的字段分隔符一般是空格符或 TAB。但你可以在命令行使用-F选项改变字符分隔符，只需在-F后面跟着你想用的分隔符即可。

```
gawk -F";"'/tparker/{print}' /etc/passwd
```

在此例中，你将字符分隔符设置成分号。注意： -F必须是大写的，而且必须在第一个引号之前。

6.10 元字符

gawk语言在格式匹配时有其特殊的规则。例如， cat能够和记录中任何位置有这三个字符的字段匹配。但有时你需要一些更为特殊的匹配。如果你想让 cat只和concatenate匹配，则需要在格式两端加上空格：

```
/ cat / {print}
```

再例如，你希望既和cat又和CAT匹配，则可以使用或（|）：

```
/ cat | CAT / {print}
```

在gawk中，有几个字符有特殊意义。下面列出可以用在 gawk格式中的这些字符：

• ^ 表示字段的开始。

例如：

```
$3 ~ /^b/
```

如果第三个字段以字符b开始，则匹配。

• $ 表示字段的结束。

例如：

```
$3 ~ /b$/
```

如果第三个字段以字符b结束，则匹配。

• . 表示和任何单字符m匹配。

例如：

```
$3 ~ /i.m/
```

如果第三个字段有字符i，则匹配。

• | 表示 “ 或 ”。

例如：

```
/cat|CAT/
```

和cat 或 CAT字符匹配。

• * 表示字符的零到多次重复。

例如：

/UNI*X/

和UNX、UNIX、UNIIX、UNIIIX等匹配。

• + 表示字符的一次到多次重复。

例如：

/UNI+X/

和UNIX、UNIIX等匹配。

• \{a，b\} 表示字符a次到b次之间的重复。

例如：

/UNI\{1，3\}X

和UNIX、UNIIX和UNIIIX匹配。

• ? 表示字符零次和一次的重复。

例如：

/UNI?X/

和UNX 和UNIX匹配。

• [] 表示字符的范围。

例如：

/I[BDG]M/

和IBM、IDM和IGM匹配

• [^] 表示不在[]中的字符。

例如：

/I[^DE]M/

和所有的以I开始、M结束的包括三个字符的字符串匹配，除了IDM和IEM之外。

6.11 调用gawk程序

当需要很多对模式和动作时，你可以编写一个gawk程序（也叫做gawk脚本）。在gawk程序中，你可以省略模式和动作两边的引号，因为在 gawk程序中，模式和动作从哪开始和从哪结束时是很显然的。

你可以使用如下命令调用gawk程序：

```
gawk -f script filename
```

此命令使gawk对文件filename执行名为script的gawk程序。

如果你不希望使用缺省的字段分隔符，你可以在 f选项后面跟着F选项指定新的字段分隔符（当然你也可以在gawk程序中指定），例如，使用分号作为字段分隔符：

```
gawk -f script -F";" filename
```

如果希望gawk 程序处理多个文件，则把各个文件名罗列其后：

```
gawk -f script filename1 filename2 filename3 ...
```

缺省情况下， gawk的输出将送往屏幕。但你可以使用 Linux的重定向命令使gawk的输出送往一个文件：

```
gawk -f script filename > save_file
```

6.12 BEGIN和END

有两个特殊的模式在gawk中非常有用。 BEGIN模式用来指明gawk开始处理一个文件之前

执行一些动作。BEGIN经常用来初始化数值，设置参数等。 END模式用来在文件处理完成后执行一些指令，一般用作总结或注释。

BEGIN 和END中所有要执行的指令都应该用花括号括起来。BEGIN 和END必须使用大写。请看下面的例子：

```
BEGIN { print "Starting the process the file" }
$1 == "UNIX" {print}
$2 > 10 {printf "This line has a value of %d", $2}
END { print "Finished processing the file. Bye!"}
```

此程序中，先显示一条信息： Starting the process the file，然后将所有第一个字段等于UNIX的整条记录显示出来，然后再显示第二个字段大于 10 的记录，最后显示信息：Finished processing the file. Bye!。

6.13 变量

在gawk中，可以用等号(=)给一个变量赋值：

```
var1 = 10
```

在gawk中，你不必事先声明变量类型。

请看下面的例子：

```
$1 == "Plastic" { count = count + 1 }
```

如果第一个字段是Plastic，则count的值加1。在此之前，我们应当给count赋予过初值，一般是在BEGIN部分。

下面是比较完整的例子：

```
BEGIN { count = 0 }
$5 == "UNIX" { count = count + 1 }
END { printf "%d occurrences of UNIX were found", count }
```

变量可以和字段和数值一起使用，所以，下面的表达式均为合法：

```
count = count + $6
count = $5 - 8
count = $5 + var1
```

变量也可以是格式的一部分，例如：

```
$2 > max_value {print "Max value exceeded by ", $2 - max_value}
$4 - var1 < min_value {print "Illegal value of ", $4}
```

6.14 内置变量

gawk语言中有几个十分有用的内置变量，现在列于下面：

NR	已经读取过的记录数。
FNR	从当前文件中读出的记录数 。
FILENAME	输入文件的名字。
FS	字段分隔符 （缺省为空格）。
RS	记录分隔符 （缺省为换行）。
OFMT	数字的输出格式（缺省为%g）。
OFS	输出字段分隔符 。
ORS	输出记录分隔符 。
NF	当前记录中的字段数 。

如果你只处理一个文件，则 NR 和FNR 的值是一样的。但如果是多个文件， NR是对所有的文件来说的，而FNR 则只是针对当前文件而言。

例如：

```
NR <= 5 {print "Not enough fields in the record"}
```

检查记录数是否小于5，如果小于5，则显示出错信息。

FS十分有用，因为FS控制输入文件的字段分隔符。例如，在 BEGIN格式中，使用如下的命令：

```
FS=":"
```

6.15 控制结构

6.15.1 if 表达式

if 表达式的语法如下：

```
if (expression){
commands
}
else{
commands
}
```

例如：

```
# a simple if loop
 (if ($1 == 0){
print "This cell has a value of zero"
}
else {
printf "The value is %d\n", $1
})
```

再看下一个例子：

```
# a nicely formatted if loop
 (if ($1 > $2){
print "The first column is larger"
}
else {
print "The second column is larger"
})
```

6.15.2 while 循环

while 循环的语法如下：

```
while (expression){
commands
}
```

例如：

```
# interest calculation computes compound interest
# inputs from a file are the amount, interest_rateand years
{var = 1
while (var <= $3) {
printf("%f\n", $1*(1+$2)^var)
var++
```

```
}
}
```

6.15.3 for 循环

for 循环的语法如下：

```
for (initialization; expression; increment) {
command
}
```

例如：

```
# interest calculation computes compound interest
# inputs from a file are the amount，interest_rateand years
{for (var=1; var <= $3; var++) {
printf("%f\n"，$1*(1+$2)^var)
}
}
```

6.15.4 next 和exit

next 指令用来告诉gawk 处理文件中的下一个记录， 而不管现在正在做什么。语法如下：

```
{ command1
command2
command3
next
command4
}
```

程序只要执行到next指令，就跳到下一个记录从头执行命令。因此，本例中， command4 指令永远不会被执行。

程序遇到exit指令后，就转到程序的末尾去执行END，如果有END的话。

6.16 数组

gawk语言支持数组结构。数组不必事先初始化。声明一个数组的方法如下：

```
arrayname[num]=value
```

请看下面的例子：

```
# reverse lines in a file
{line[NR] = $0 } # remember each line
END {var=NR # output lines in reverse order
while (var > 0){
print line[var]
var--
}
}
```

此段程序读取一个文件的每一行，并用相反的顺序显示出来。我们使用 NR作为数组的下标来存储文件的每一条记录，然后在从最后一条记录开始，将文件逐条地显示出来。

6.17 用户自定义函数

复杂的 gawk 程序常常可以使用自己定义的函数来简化。调用用户自定义函数与调用内部函数的方法一样。函数的定义可以放在 gawk 程序的任何地方。

用户自定义函数的格式如下：

```
function name (parameter-list) {
  body-of-function
}
```

name 是所定义的函数的名称。一个正确的函数名称可包括一序列的字母、数字、下标线(underscores)，但是不可用数字做开头。parameter-list 是函数的全部参数的列表，各个参数之间以逗点隔开。body-of-function 包含 gawk 的表达式，它是函数定义里最重要的部分，它决定函数实际要做的事情。

下面这个例子，会将每个记录的第一个字段的值的平方与第二个字段的值的平方加起来。

```
{print "sum =", SquareSum($1, $2)}
function SquareSum(x, y) {
sum=x*x+y*y
return sum
}
```

到此，我们已经知道了gawk的基本用法。gawk语言十分易学好用，例如，你可以用 gawk 编写一段小程序来计算一个目录中所有文件的个数和容量。如果用其他的语言，如 C语言，则会十分的麻烦，相反，gawk只需要几行就可以完成此工作。

6.18 几个实例

最后，再举几个gawk的例子：

```
gawk '{if (NF > max) max = NF}
  END {print max}'
```

此程序会显示所有输入行之中字段的最大个数。

```
gawk 'length($0) > 80'
```

此程序会显示出超过 80 个字符的每一行。此处只有模式被列出，动作是采用缺省值显示整个记录。

```
gawk 'NF > 0'
```

显示拥有至少一个字段的所有行。这是一个简单的方法，将一个文件里的所有空白行删除。

```
gawk 'BEGIN {for (i = 1; i <= 7; i++)
print int(101 * rand())}'
```

此程序会显示出范围是 0 到 100 之间的 7 个随机数。

```
ls -l files | gawk '{x += $4}; END {print "total bytes: " x}'
```

此程序会显示出所有指定的文件的总字节数。

```
expand file | gawk '{if (x < length()) x = length()}
END {print "maximum line length is " x}'
```

此程序会将指定文件里最长一行的长度显示出来。expand 会将 tab 改成 space，所以是用实际的右边界来做长度的比较。

```
gawk 'BEGIN {FS = ":"}
  {print $1 | "sort"}' /etc/passwd
```

此程序会将所有用户的登录名称，依照字母的顺序显示出来。

```
gawk '{nlines++}
  END {print nlines}'
```

此程序会将一个文件的总行数显示出来。

```
gawk 'END {print NR}'
```

此程序也会将一个文件的总行数显示出来，但是计算行数的工作由 gawk来做。

```
gawk '{print NR, $0}'
```

此程序显示出文件的内容时，会在每行的最前面显示出行号，它的函数与‘ cat -n ’类似。