第十一章　文件系统

by flamephoenix

本章所讲的函数多数使用了UNIX操作系统的特性，在非UNIX系统中，一些函数可能没有定义或有不同的工作方式，使用时请查看Perl联机文档。

一、文件输入/输出函数

本节讲述从文件中读取信息和向文件写入信息的内置库函数。

1、基本I/O函数

一些I/O函数在前面的章节中已有讲述，如

- open：允许程序访问文件
- close：终止文件访问
- print：文件写入字符串
- write：向文件写入格式化信息
- printf：格式化字符串并输出到文件

这里简单回顾一下，再讲一些前面未提到的函数。

1)open函数

open函数将文件变量与某文件联系起来，提供访问文件的接口，例如：open(MYVAR, "/u/file"); 如果文件打开成功，则返回非零值，否则返回零。缺省地，open打开文件用以读取其内容，若想打开文件以写入内容，则在文件名前加个大于号：open(MYVAR, ">/u/file"); 向已有的文件末尾添加内容用两个大于号：open(MYVAR, ">>/u/file"); 若想打开文件作为数据导向的命令，则在命令前加上管道符(|)：open(MAIL, "|mail dave");

2)用open重定向输入

可以把打开的文件句柄用作向程序输入数据的命令，方法是在命令后加管道符(|)，如：

```
open(CAT, "cat file*|");
```

对open的调用运行命令cat file* ，此命令创建一个临时文件，这个文件的内容是所有以file打头的文件的内容连接而

成，此文件看作输入文件，可用文件变量CAT访问，如：

```
$input = ;
```

下面的例子使用命令w的输出来列出当前登录的所有用户名。

```
                1 : #!/usr/local/bin/perl
    2 :
    3 : open (WOUT, "w|");
    4 : $time = <WOUT>;
    5 : $time =~ s/^ *//;
    6 : $time =~ s/ .*//;
    7 : ; # skip headings line
    8 : @users = ;
    9 : close (WOUT);
    10: foreach $user (@users) {
    11:   $user =~ s/ .*//;
    12: }
    13: print ("Current time: $time");
    14: print ("Users logged on:\n");
    15: $prevuser = "";
    16: foreach $user (sort @users) {
    17:   if ($user ne $prevuser) {
    18:     print ("\t$user");
    19:     $prevuser = $user;
    20:   }
    21: }
```

结果输出如下：

```
              Current time: 4:25pm
    Users logged on:
      dave
      kilroy
      root
      zarquon
```

w命令列出当前时间、系统负载和登录的用户，以及每个用户的作业时间和当前运行的命令，如：

```
              4:25pm  up 1 day,  6:37,  6 users,  load average: 0.79, 0.36, 0.28
    User     tty        login@  idle   JCPU   PCPU what
    dave     ttyp0      2:26pm            27      3 w
    kilroy   ttyp1      9:01am  2:27   1:04     11 -csh
    kilroy   ttyp2      9:02am    43   1:46     27 rn
    root     ttyp3      4:22pm     2                -csh
    zarquon  ttyp4      1:26pm     4      43     16 cc myprog.c
    kilroy   ttyp5      9:03am          2:14     48 /usr/games/hack
```

上例中从w命令的输出中取出所需的信息：当前时间和登录的用户名。第3行运行w命令，此处对open的调用指定w的输出用作程序的输入，用文件变量WOUT来访问该输入。第4行读取第一行信息，即：

4:25pm up 1 day, 6:37, 6 users, load average: 0.79, 0.36, 0.28

接下来的两行从这行中抽取出时间。首先，第5行删除起始的空格，然后第6行删去除时间和结尾换行符之间的所有字符，存入变量$time。

第7行从WOUT读取第二行，这行中无有用信息，故不作处理。第8行把剩下的行赋给数组@users，然后第9行关闭WOUT，终止运行w命令的进程。

@users中的每个元素都是一行用户信息，因为本程序只需要每行的第一个单词，即用户名，故10~12行去掉除换行符外的其它字符，这一循环结束后，@users中只剩下用户名的列表。

第13行输出存贮在$time中的时间，注意这时print不需要加上换行符，因为$time中有。16~21行对@users中的用户名排序并输出。因为同一个用户可以多次登录，所以用$preuser存贮输出的最后一个用户名，下次输出数组元素$user时，如果其与$preser相等，则不输出。

3)文件重定向

许多UNIX shell可以把标准输出文件(STDOUT)和标准错误文件(STDERR)都重定向到同一个文件，例如在Bourne Shell（sh）中，命令

```
$ foo > file1 2>&1
```

运行命令foo并把输出到标准输出文件和标准错误文件的内容存贮到文件file1中。下面是用Perl实现这一功能的例子:

```
1: #!/usr/local/bin/perl
2:
3: open (STDOUT, ">file1") || die ("open STDOUT failed");
4: open (STDERR, ">&STDOUT") || die ("open STDERR failed");
5: print STDOUT ("line 1\n");
6: print STDERR ("line 2\n");
7: close (STDOUT);
8: close (STDERR);
```

运行后，文件file1中的内容为:

```
line 2
line 1
```

可以看到，这两行并未按我们想象的顺序存贮，为什么呢？我们来分析一下这段程序。

第3行重定向标准输出文件，方法是打开文件file1将它与文件变量STDOUT关联，这也关闭了标准输出文件。第4行重定向标准错误文件，参数>&STDOUT告诉Perl解释器使用已打开并与STDOUT关联的文件，即文件变量STDERR指向与STDOUT相同的文件。第5、6行分别向STDOUT和STDERR写入数据，因为这两个文件变量指向同一个文件，故两行字符串均写到文件file1中，但顺序却是错误的，怎么回事呢？

问题在于UNIX对输出的处理上。当使用print（或其它函数）写入STDOUT等文件时，UNIX操作系统真正所做的是把数据拷贝到一片特殊的内存即缓冲区中，接下来的输出操作继续写入缓冲区直到写满，当缓冲区满了，就把全部数据实际输出。象这样先写入缓冲区再把整个缓冲区的内容输出比每次都实际输出所花费的时间要少得多，因为一般来说，I/O比内存操作慢得多。

程序结束时，任何非空的缓冲区都被输出，然而，系统为STDOUT和STDERR分别维护一片缓冲区，并且先输出STDERR的内容，因此存贮在STDERR的缓冲区中的内容line 2出现在存贮在STDOUT的缓冲区中的内容line 1之前。

为了解决这个问题，可以告诉Perl解释器不对文件使用缓冲，方法为:

1、用select函数选择文件

2、把值1赋给系统变量$|

系统变量$|指定文件是否进行缓冲而不管其是否应该使用缓冲。如果$|为非零值则不使用缓冲。$|与系统变量$~和$^协同工作，当未调用select函数时，$|影响当前缺省文件。下例保证了输出的次序:

```
1 : #!/usr/local/bin/perl
2 :
3 : open (STDOUT, ">file1") || die ("open STDOUT failed");
4 : open (STDERR, ">&STDOUT") || die ("open STDERR failed");
5 : $| = 1;
6 : select (STDERR);
7 : $| = 1;
8 : print STDOUT ("line 1\n");
9 : print STDERR ("line 2\n");
10: close (STDOUT);
11: close (STDERR);
```

程序运行后，文件file1中内容为:

```
line 1
line 2
```

第5行将$|赋成1，告诉Perl解释器当前缺省文件不进行缓冲，因为未调用select，当前的缺省文件为重定向到文件file1的STDOUT。第6行将当前缺省文件设为STDERR，第7行又设置$|为1，关掉了重定向到file1的标准错误文件的缓冲。由于STDOUT和STDERR的缓冲均被关掉，向其的输出立刻被写到文件中，因此line 1出现在第一行。

4)指定读写权限

打开一个既可读又可写的文件方法是在文件名前加上"+>"，如下:

```
open (READWRITE, "+>file1");
```

此语句打开既可读又可写的文件file1，即可以重写其中的内容。文件读写操作最好与库函数seek和tell一起使用，这样可以跳到文件任何一点。

注: 也可用前缀"+<"指定可读写权限。

5)close函数

用于关闭打开的文件。当用close关闭管道，即重定向的命令时，程序等待重定向的命令结束，如:

```
open (MYPIPE, "cat file*|");
close (MYPIPE);
```

当关闭此文件变量时，程序暂停运行，直到命令cat file*运行完毕。

6)print, printf和write函数

print是这三个函数中最简单的，它向指定的文件输出，如果未指定，则输出到当前缺省文件中，如：

```
print ("Hello, there!\n");
print OUTFILE ("Hello, there!\n");
```

第一句输出到当前缺省文件中，若未调用select，则为STDOUT。第二句输出到由文件变量OUTFILE指定的文件中。

printf函数先格式化字符串再输出到指定文件或当前缺省文件中，如：

```
printf OUTFILE (“You owe me %8.2f", $owing);
```

此语句取出变量$owing的值并替换掉串中的%8.2f，%8.2f是域格式的例子，把$owing的值看作浮点数。

write函数使用输出格式把信息输出到文件中，如：

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

关于printf和write，详见《第x章 格式化输出》。

7)select函数

select函数将通过参数传递的文件变量指定为新的当前缺省文件，如：

```
select (MYFILE);
```

这样，MYFILE就成了当前缺省文件，当对print、write和printf的调用未指定文件时，就输出到MYFILE中。

8)eof函数

eof函数查看最后一次读文件操作是否为文件最后一个记录，如果是，则返回非零值，如果文件还有内容，返回零。

一般情况下，对eof的调用不加括号，因为eof和eof()是等效的，但与<>操作符一起使用时，eof和eof()就不同了。现在我们来创建两个文件，分别叫做file1和file2。file1的内容为：

```
This is a line from the first file.
Here is the last line of the first file.
```

file2的内容为：

```
This is a line from the second and last file.
Here is the last line of the last file.
```

下面就来看一下eof和eof()的区别，第一个程序为：

```
1: #!/usr/local/bin/perl
2:
3: while ($line = <>) {
4:   print ($line);
5:   if (eof) {
6:     print ("-- end of current file --\n");
7:   }
8: }
```

运行结果如下：

```
$ program file1 file2
This is a line from the first file.
Here is the last line of the first file.
-- end of current file --
This is a line from the second and last file.
Here is the last line of the last file.
-- end of current file --
$
```

下面把eof改为eof()，第二个程序为：

```
1: #!/usr/local/bin/perl
2:
3: while ($line = <>) {
4:   print ($line);
5:   if (eof()) {
6:     print ("-- end of output --\n");
7:   }
8: }
```

运行结果如下：

```
$ program file1 file2
This is a line from the first file.
```

```
Here is the last line of the first file.
This is a line from the second and last file.
Here is the last line of the last file.
-- end of output --$
```

这时，只有所有文件都读过了，eof()才返回真，如果只是多个文件中前几个的末尾，返回值为假，因为还有要读取的输入。

9)间接文件变量

对于上述各函数open, close, print, printf, write, select和eof，都可以用简单变量来代替文件变量，这时，简单变量中所存贮的字符串就被看作文件变量名，下面就是这样一个例子，此例很简单，就不解释了。需要指出的是，函数open, close, write, select和eof还允许用表达式来替代文件变量，表达式的值必须是字符串，被用作文件变量名。

```
                1: #!/usr/local/bin/perl
2:
3: &open_file("INFILE", "", "file1");
4: &open_file("OUTFILE", ">", "file2");
5: while ($line = &read_from_file("INFILE")) {
6:   &print_to_file("OUTFILE", $line);
7: }
8:
9: sub open_file {
10:   local ($filevar, $filemode, $filename) = @_;
11:
12:   open ($filevar, $filemode . $filename) ||
13:     die ("Can't open $filename");
14: }
15: sub read_from_file {
16:   local ($filevar) = @_;
17:
18:   <$filevar>;
19: }
20: sub print_to_file {
21:   local ($filevar, $line) = @_;
22:
23:   print $filevar ($line);
24: }
```

2、跳过和重读数据

函数名	seek
调用语法	seek (filevar, distance, relative_to);
解说	在文件中向前/后移动，有三个参数: 1、filevar，文件变量 2、distance，移动的字节数，正数向前移动，负数往回移动 3、reletive_to，值可为0、1或2。为0时，从文件头开始移动，为1时，相对于当前位置（将要读的下一行）移动，为2时，相对于文件末尾移动。 运行成功返回真（非零值），失败则返回零，常与tell函数合用。

函数名	tell
调用语法	tell (filevar);
解说	返回从文件头到当前位置的距离。 注意: 1、seek和tell不能用于指向管道的文件变量。 2、seek和tell中文件变量参数可使用表达式。

3、系统读写函数

函数名	read
调用语法	read (filevar, result, length, skipval);
	read函数设计得与UNIX的fread函数等效，可以读取任意长度的字符（字节）存入一个简单变量。其参数有四个:

解说	1、filevar：文件变量 2、result：存贮结果的简单变量（或数组元素） 3、length：读取的字节数 4、skipval：可选项，指定读文件之前跳过的字节数。 返回值为实际读取的字节数，如果已到了文件末尾，则返回零，如果出错，则返回空串。

函数名	sysread
调用语法	sysread (filevar, result, length, skipval);
解说	更快的读取数据，与UNIX函数read等效，参数与read相同。

函数名	syswrite
调用语法	syswrite (filevar, data, length, skipval);
解说	更快的写入数据，与UNIX函数write等效，参数： 1、filevar：将要写入的文件 2、data：存贮要写入数据的变量 3、length：要写入的字节数 4、skipval写操作之前跳过的字节数。

4、用getc读取字符

函数名	getc
调用语法	$char = getc (infile);
解说	从文件中读取单个字符。

5、用binmode读取二进制文件

函数名	binmode
调用语法	binmode (filevar);
解说	当你的系统（如类DOS系统）对文本文件和二进制文件有所区别时使用。必须在打开文件后、读取文件前使用。

二、目录处理函数

函数名	mkdir
调用语法	mkdir (dirname, permissions);
解说	创建新目录，参数为： 1、dirname：将要创建的目录名，可以为字符串或表达式 2、permissions：8进制数，指定目录的访问权限，其值和意义见下表，权限的组合方法为将相应的值相加。

值	权限
4000	运行时设置用户ID
2000	运行时设置组ID
1000	粘贴位
0400	拥有者读权限
0200	拥有者写权限
0100	拥有者执行权限
0040	组读权限
0020	组写权限
0010	组执行权限
0004	所有人读权限
0002	所有人写权限
0001	所有人执行权限

函数名	chdir
调用语法	chdir (dirname);
解说	改变当前工作目录。参数dirname可以为字符串，也可以为表达式。

函数名	opendir
调用语法	opendir (dirvar, dirname);
	打开目录，与下面几个函数合用，可查看某目录中文件列表。参数为： 1、dirvar：目录变量，与文件变量类似

解说	2、dirname：目录名，可为字符串或表达式 成功返回真值，失败返回假。 注：程序中可用同名的目录变量和文件变量，根据环境确定取成分。

函数名	closedir
调用语法	closedir (mydir);
解说	关闭打开的目录。

函数名	readdir
调用语法	readdir (mydir);
解说	赋给简单变量时，每次赋予一个文件或子目录名，对数组则赋予全部文件和子目录名。

函数名	telldir
调用语法	location = telldir (mydir);
解说	象在文件中前后移动一样，telldir和下面的seekdir用于在目录列表中前后移动。

函数名	seekdir
调用语法	seekdir(mydir, location);
解说	location必须为telldir返回的值。

函数名	rewinddir
调用语法	rewinddir (mydir);
解说	将读取目录的位置重置回开头，从而可以重读目录列表。

函数名	rmdir
调用语法	rmdir (dirname);
解说	删除空目录。成功则返回真（非零值），失败返回假（零值）。

三、文件属性函数

1、文件重定位函数

函数名	rename
调用语法	rename (oldname, newname);
解说	改变文件名或移动到另一个目录中，参数可为字符串或表达式。

函数名	unlink
调用语法	num = unlink (filelist);
解说	删除文件。参数为文件名列表，返回值为实际删除的文件数目。 此函数之所以叫unlink而不叫delete是因为它实际所做的是删除文件的链接。

2、链接和符号链接函数

函数名	link
调用语法	link (newlink, file);
解说	创建现有文件的链接--硬链接，file是被链接的文件，newlink是被创建的链接。 成功返回真，失败返回假。 当删除这两个链接中的一个时，还可以用另一个来访问该文件。

函数名	symlink
调用语法	symlink (newlink, file);
解说	创建现有文件的符号链接，即指向文件名，而不是指向文件本身。参数和返回值同上。 当原文件被删除（如：被unlinke函数删除），则被创建链接不可用，除非再创建一个与原被链接的文件同名的文件。

函数名	readlink
调用语法	filename = readlink (linkname);
解说	如果linkname为符号链接文件，返回其实际指向的文件。否则返回空串。

3、文件许可权函数

函数名	chmod
调用语法	chmod (permissions, filelist);

解说	改变文件的访问权限。参数为: 1、permissions为将要设置的权限，其含义见上述mkdir中权限表 2、filelist为欲改变权限的文件列表

函数名	chown
调用语法	chown (userid, groupid, filelist);
解说	改变文件的属主，有三个参数: 1、userid: 新属主的(数字)ID号 2、groupid: 新的组(数字)ID号，-1为保留原组 3、filelist: 欲改变属主的文件列表

函数名	umask
调用语法	oldmaskval = umask (maskval);
解说	设置文件访问权限掩码，返回值为当前掩码。

4、其它属性函数

函数名	truncate
调用语法	truncate (filename, length);
解说	将文件的长度减少到length字节。如果文件长度已经小于length，则不做任何事。其中filename可以为文件名，也可以为文件变量

函数名	stat
调用语法	stat (file);
解说	获取文件状态。参数file可为文件名也可为文件变量。返回列表元素依次为: ● 文件所在设备 ● 内部参考号(inode) ● 访问权限 ● 硬链接数 ● 属主的(数字)ID ● 所属组的(数字)ID ● 设备类型（如果file是设备的话） ● 文件大小（字节数） ● 最后访问时间 ● 最后修改时间 最后改变状态时间 ● I/O操作最佳块大小 ● 分配给该文件的块数

函数名	lstat
调用语法	lstat (file);
解说	与stat类似，区别是将file看作是符号链接。

函数名	time
调用语法	currtime = time();
解说	返回从1970年1月1日起累计秒数。

函数名	gmtime
调用语法	timelist = gmtime (timeval);
解说	将由time, stat 或 -A 和 -M 文件测试操作符返回的时间转换成格林威治时间。返回列表元素依次为: ● 秒 ● 分钟 ● 小时，0~23 ● 日期 ● 月份，0~11(一月~十二月) ● 年份 ● 星期，0~6(周日~周六) ● 一年中的日期，0~364 ● 是否夏令时的标志 详见UNIX的gmtime帮助。

函数名	localtime
调用语法	timelist = localtime (timeval);

解说	与gmtime类似，区别为将时间值转换为本地时间。

函数名	utime
调用语法	utime (acctime, modtime, filelist);
解说	改变文件的最后访问时间和最后更改时间。例如： $acctime = -A "file1"; $modtime = -M "file1"; @filelist = ("file2", "file3"); utime ($acctime, $modtime, @filelist);

函数名	fileno
调用语法	filedesc = fileno (filevar);
解说	返回文件的内部UNIX文件描述。参数filevar为文件变量。

函数名	fcntl flock
调用语法	fcntl (filevar, fcntlrtn, value); flock (filevar, flockop);
解说	详见同名UNIX函数帮助。

四、使用DBM文件

Perl中可用关联数组来访问DBM文件，所用函数为dbmopen和dbmclose，在Perl5中，已用tie和untie代替。

函数名	dbmopen
调用语法	dbmopen (array, dbmfilename, permissions);
解说	将关联数组与DBM文件相关联。参数为： 1、array：所用关联数组 2、dbmfilename：将打开的DBM文件名 3、访问权限，详见mkdir

函数名	dbmclose
调用语法	dbmclose (array);
解说	关闭DBM文件，拆除关联数组与之的关系。