# 第12学时　使用Perl的命令行工具

到现在为止，Perl一直只是一个非常简单的解释程序。你将一个程序键入一个文件，然后调用Perl解释程序，以便运行你的程序。不过Perl解释程序比这个程序灵活得多。

Perl解释程序中内置了一个调试程序。使用该调试程序，可以像播放录像带那样运行你的Perl程序。可以将程序倒到开头，使它慢速运行，也可以快速运行，还可以将它定格，以仔细观察程序的内部结构。在查找Perl程序中存在的问题时，调试程序常常是个使用得很不充分的工具。

Perl也能运行不是键入文件的程序。例如，可以直接从系统的命令提示符处运行一些小程序。

在本学时中，你将要学习：

- 如何使用Perl的调试程序。
- 如何使用命令行开关来编写Perl程序。

## 12.1　什么是调试程序

Perl调试程序是个Perl解释程序的内置特性。它使你能够取出任何一个 Perl程序，然后逐个语句运行该程序。在运行过程中，你可以查看各个变量，修改这些变量，让程序运行较长的时间，中断程序的运行，或者从头开始运行该程序。

从你的程序角度来看，它与普通程序并无区别。输入仍然来自键盘，输出仍然送往屏幕。程序并不知道何时停止运行，何时它正在运行。实际上，你可以观察程序的运行情况，根本不必中断程序的运行。

### 12.1.1　启动调试程序

若要启动Perl调试程序，必须打开操作系统的命令提示符。如果你是DOS和Windows用户，那么要打开MS-DOS的标准提示符C:\。如果是UNIX用户，这个提示符应该是你登录时显示的提示符（通常是%或$）。

对于运行Perl的Macintosh用户来说，只需从Script菜单中选定Debugger。这时就会为你打开带有提示符的Debugger窗口。

本节中的所有代码例子均使用第 9学时的程序清单9-2中的Employee程序。你会发现可以很容易将一个书签放在这一页上，然后前后翻阅，查找你想要的信息。若要在提示符处启动调试程序（本例中使用DOS提示符），请键入下面这行命令：

```
C:\> perl -d Employee
```

Perl的-d开关可使Perl以调试方式启动运行。命令行上也指明了被调试的程序。然后显示关于版本信息的某些消息，如下所示：

```
Loading DB routines from perl5db.pl version 1.0401

Emacs support available.
```

```
Enter h or `h h' for help.

main::(Employee:5):     my @employees=(
main::(Employee:6):             'Smith,Bob,123101,9.35,40',
main::(Employee:7):             'Franklin,Alice,132912,10.15,35',
main::(Employee:8):             'Wojohowicz,Ted,198131,6.50,39',
main::(Employee:9):             'Ng,Wendy,141512,9.50,40',
main::(Employee:10):            'Cliburn,Stan,131211,11.25,40',
main::(Employee:11):    );
 DB<1>__
```

该调试程序首先显示版本号（1.0401，你的版本号可能不一样）和 help (帮助)提示。接着显示该程序的第一行可执行代码。由于第一个语句实际上包含 7行，从“ my @employees=（”开始，以“ ）；”为结尾，因此所有 7行语句均显示一个描述，以说明它们来自什么文件（Employee），以及它们是在文件的哪一行或哪几行上找到的（第 5至第11行）。

最后，你看到调试文件的提示符 DB<1>。1表示调试文件正在等待它的第一个命令。调试程序提示符后面的光标正等待你输入命令。

这时，你的Perl程序实际上暂停在第一个指令 -my @employees=(的前面。每当调试程序向你显示程序中的一个语句时，它就是准备要执行的语句，而不是上一个运行的语句。

现在调试程序已经作好准备，等待你输入命令。

### 12.1.2 调试程序的基本命令

输入调试程序的第一个和最重要的命令是 help（帮助）命令。如果在调试程序的提示符处键入h，那么调试程序的所有可用命令均被输出。也可以使用该命令的某种变形，如 h h，它能输出命令和语句的汇总， h cmd用于输出某个命令的帮助信息。

帮助命令的列表也许比较长，一个屏幕显示不下，开头的几个命令显示后，就需要向下滚动。若要每次显示一屏调试命令，可以在命令的前面加上一个 |字符。因此，如果想每次查看一屏帮助命令，请使用命令 |h。

调试程序的最常用特性是每次运行一个 Perl代码的指令。因此，如果继续使用上面的例子，若要转至你的Perl程序的下一个语句，可以使用调试程序的命令 n：

```
   DB<8> L
Employee:
 33:            print_emp($_);
```

当你键入命令 n后， Perl就执行 Employee程序的第 5至11行语句。然后调试程序输出要执行的下一个语句（但尚未运行） my($L1，$F1)=split（’，’，$a）；并显示另一个提示符。

当程序运行到这个时候， @employees被初始化为5个名字和工资等。若要查看这些信息，可以将它们输出：

```
  DB<1> print @employees
Smith,Bob,123101,9.35,40Franklin,Alice,132912,10.15,35Wojohowicz,Ted,198131,6.50
,39Ng,Wendy,141512,9.50,40Cliburn,Stan,131211,11.25,40
  DB<2> __
```

实际上， Perl的任何语句都可以在调试程序提示符后面运行。请注意，来自 @employees的数组元素都是一道运行的。可以输入下面的命令，以便很好地将它们输出：

```
  DB<2> print join("\n", @employees)
Smith,Bob,123101,9.35,40
Franklin,Alice,132912,10.15,35
Wojohowicz,Ted,198131,6.50,39
Ng,Wendy,141512,9.50,40
Cliburn,Stan,131211,11.25,40
  DB<3> __
```

若要继续运行该程序，只需不断键入 n，如下所示：

```
  DB<3> n
main::(Employee:23):    @employees=sort {
  DB<3> n
main::(Employee:25):                    my ($L2, $F2)=split(',', $b);
  DB<3> n
main::(Employee:26):                    return( $L1 cmp $L2
main::(Employee:27):                                ||
main::(Employee:28):                            $F1 cmp $F2
main::(Employee:29):                    );
  DB<3> n
main::(Employee:23):    @employees=sort {
  DB<3>
```

显然，调试程序将该程序倒退到这个位置，第 23行准备再次运行。 Perl的sort语句实际上是个循环，调试程序逐步通过 sort代码块中的每个语句。如果你不断键入 n，那么调试程序将不断循环运行，直到sort运行结束，这需要花费一定的时间。

若要重复运行上面的命令，也可以在调试程序的提示符处按 Enter键。

### 12.1.3 断点

如果不是每次执行一个指令，逐步执行该程序，你也可以让调试程序连续运行你的 Perl程序，直到到达某个语句，然后停止运行。这些停止运行的位置称为断点。

若要设置断点，必须在程序中选定一个要停止运行的位置。命令 l用于列出程序的下面 10行。再次键 l，可以列出下面的 10行，如此类推。若要列出从某一行开始的程序，请键入 l lineno，其中lineno是程序的行号。也可以设定要列出的行的范围，方法是键入命令 l start-end。

在程序清单中，标号= = = >用于指明调试程序准备执行的当前行，请看下面的代码：

```
  DB<3> l  11
23:==>   @employees=sort {
24:                     my ($L1, $F1)=split(',', $a);
25:                     my ($L2, $F2)=split(',', $b);
26:                     return( $L1 cmp $L2
27:                                 ||
28:                             $F1 cmp $F2
29:                     );
30:                      } @employees;
31:
32:     foreach(@employees) {
33:             print_emp($_);
  DB<3> __
```

在这个例子中，第33行是设置断点的好位置。它位于 sort语句的后面，并且它是程序的主循环中的第一个语句。你可以在 Perl程序中的任何位置上设置断点，只要这个断点是个有效的 Perl语句。但是断点不能设置在花括号（第 30行）、标点符号（第 29行）、空行（第31行）或只包含注释的代码行上。

若要设置断点，请使用 b breakpoint命令，其中breakpoint可以是行号或子例程名。例如，

若要在第33行上设置断点，可以输入下面这个命令：

```
DB<3> b 33
DB<4>
```

必须知道的另一个关于断点的命令是继续命令 c。命令c向调试程序发出指令，使Perl程序运行到下一个断点或程序的结尾：

```
  DB<5> c
main::(Employee:33):            print_emp($_);
  DB<6>
```

在这个代码中，调试程序按照要求使Perl程序停止在第33行上，print-emp函数被调用之前。断点仍然可以设置，因此，键入另一个 c子句，可使程序继续运行 print_emp()函数，并且再次停止在第33行上：

```
main::(Employee:5):     my @employees=(
main::(Employee:6):             'Smith,Bob,123101,9,35,40',
main::(Employee:7):             'Franklin,Alice,132912,10,15,35',
main::(Employee:8):             'Wojohowicz,Ted,198131,6,50,39',
main::(Employee:9):             'Ng,Wendy,141512,9,50,40',
main::(Employee:10):            'Cliburn,Stan,131211,11,25,40',
main::(Employee:11):    );
 DB<1> n            ←“下一个”指令的命令
main::(Employee:24):                    my ($L1, $F1)=split(',', $a);
 DB<1>
```

若要查看你在程序中已经设置的断点，可以像下面这样使用命令 L：

```
DB<7> C
131211 Stan Cliburn   11.25  40  450.00        ← print_emp ( ) 的输出
main:: (Employee : 33):    print _ emp ($_) ;  ← 待运行的下一行代码
DB<8>
```

这个例子显示了调试程序有一个断点，在文件 Employee中的第33行上。

若要撤消程序中的断点，可以采用设置断点时的相同方法使用命令 d，比如d line或d subname：

```
DB<9> d 33
DB<10>
```

### 12.1.4 其他调试程序命令

如果想查看print-emp()函数的运行情况，可以用若干不同的方法来执行这项操作。首先使用命令R，重新启动你的程序：

```
DB<11> R

Warning: some settings and command-line options may be lost!

Loading DB routines from perl5db.pl version 1.0401

Emacs support available.

Enter h or `h h' for help.

main::(Employee:5):     my @employees=(
main::(Employee:6):             'Smith,Bob,123101,9.35,40',
main::(Employee:7):             'Franklin,Alice,132912,10.15,35',
main::(Employee:8):             'Wojohowicz,Ted,198131,6.50,39',
main::(Employee:9):             'Ng,Wendy,141512,9.50,40',
main::(Employee:10):            'Cliburn,Stan,131211,11.25,40',
```

```
main::(Employee:11):     );
  DB<12> b 33
```

命令R用于使Perl程序回到它的开始处，准备再次执行该程序。你已经设置的断点保持已设置状态，Perl程序中的所有变量均复位。在上面这个代码中，断点设置在第 33行上。这时可用下面的命令继续运行该程序：

```
  DB<13> c
main::(Employee:33):            print_emp($_);
  DB<14> __
```

如果执行命令n，就可以执行下面的指令：

```
  DB<14> n
131211 Stan Cliburn          11.25  40  450.00
main::(Employee:32):    foreach(@employees) {
  DB<15> n
main::(Employee:33):            print_emp($_);
  DB<16> __
```

但是，如果用这种方法逐步执行该程序，你就无法查看 print-emp()中究竟有什么。若要单步进入print-emp()，不要使用命令n，而应该使用命令s，即单步执行命令。s命令与n命令的作用很相似，不过s命令并不仅仅执行函数，然后转入下一个指令，而是执行函数，然后在函数中的第1个指令处停止运行，如你在下面看到的情况一样：

```
main::(Employee:33):            print_emp($_);
  DB<16> s
main::print_emp(Employee:14):          my($last,$first,$emp,$hourly,$time)=
main::print_emp(Employee:15):                  split(',', $_[0]);
  DB<17>__
```

在这里，显示了 print_emp()的第1个语句。也可以用 b print_emp 设置一个断点，在print_emp()中停止运行。现在可以继续运行该程序，使用命令 n逐步通过该函数，如下所示：

```
  DB<18> n
main::print_emp(Employee:16):          my $fullname;
  DB<19> n
main::print_emp(Employee:17):          $fullname=sprintf("%s %s", $first,
$last);
  DB<20> n
main::print_emp(Employee:18):          printf("%6d %-20s %6.2f %3d %7.2f\n",
main::print_emp(Employee:19):                 $emp, $fullname, $hourly, $time,
main::print_emp(Employee:20):                 $hourly * $time);
  DB<21> __
```

还可以在Perl程序运行时修改程序里的变量。例如，若要给员工每小时临时增加 2.50美元的工资，可以输入下面的代码：

```
  DB<21> print $hourly
10.15
  DB<22> $hourly=$hourly+2.50

  DB<23> n
132912 Alice Franklin       12.65  35  442.75
main::(Employee:32):    foreach(@employees) {
  DB<24>
```

在上面的代码段中，变量 $hourly被输出（10.15），并增加2.50，然后程序继续运行。Printf语句输出$hourly的新值。

最后，若要退出调试程序，只需在调试程序提示符处键入 q。

## 12.2 练习：查找错误

这个练习向你展示如何使用调试程序来查找程序中的错误。程序清单 12-1中的程序存在

一个问题（实际上是两个问题）。它应该输出下面这些消息：

```
20 glasses of Lemonade on the wall
19 glasses of Lemonade on the wall
:
1 glass of Lemonade on the wall
0 glasses of Lemonade on the wall
```

但是它并没有输出这些消息。你的任务是键入程序清单 12-1中的程序，并没法找出里面的错误。这些错误都不属于语句问题， Perl的警告特性没有被激活，同时， use strict没有输出任何消息，不过调试程序应该使得错误可以非常容易地被找到。

当你键入程序后，用调试程序运行 Perl，以便设法找到错误。请记住，要经常输出有关的变量和表达式，并单步通过各个函数调用，每次调用一个函数。

**程序清单12-1 Buggy程序**

```
1:   # !/usr/bin/perl -w
2:   # This program contains TWO errors
3:   use strict;
4:
5:   sub message {
6:      my($quant)=@_;
7:      my $mess;
8:      $mess="$quant glasses of Lemonade on the wall\n";
9:      if ($quant eq 1) {
10:             $mess=s/glasses/glass/;
11:      }
12:      print $mess;
13:  }
14:
15:  foreach(20..0) {
16:     &message($_);
17:  }
```

这个问题的解决办法请参见本学时结尾处的“ 思考题 ”。

## 12.3 其他命令行特性

调试程序并不是Perl解释程序中可以用命令行开关激活的惟一特性。实际上，许多非常有用的Perl程序都可以在命令行提示符处编写。

Macintosh用户应该通过 Script菜单来运行这些命令行代码，方法是选定 l-liners，然后将命令键入对话框。

### 12.3.1 单命令行程序（One-Liners）

这种程序的关键是在命令行上赋予 Perl的-e开关。-e的后面可以是任何 Perl语句，如下例所示：

```
C:\> perl -e "print 'Hello, world';"
Hello, world
```

你可以使用多个-e开关来插入多个语句，或者用分号将这些语句隔开，如下所示：

```
C:\> perl -e "print 'Hello, world';" -e "print 'Howzit goin?'"
Hello, worldHowzit goin?
```

应该注意的是，大多数命令解释程序都规定了引号的使用规则。 Windows/DOS命令解释程序（command.com或NT的命令Shell）允许你使用双引号将单词括起来，比如上例中的 print和Hello World，但是不能随意将双引号放入别的双引号中，也不能随意将 >、<、|或^等符号放入双引号中。关于DOS/Windows中引号的详细使用规则，请查看你的操作系统手册。

在UNIX下，一般来说，只要引号配对，即每个开引号有一个闭引号，并且嵌入的引号前面有一个反斜杠\，那么你的代码就没有问题：

```
$ perl -e 'print "Hello, World\n";' -e 'print "Howzit goin?\n"'
```

上面这个代码在大多数 UNIX shell(csh、ksh、bash等)下均能运行，并能输出带有正确换行符的消息。若要了解你的shell的引号使用规则的完整列表，请查看 shell的在线手册页。

-e开关的一个经常的非常有用的用法是与 -d组合起来使用，并将它直接放入 Perl的调试程序中，但是没有需要调试的程序：

```
C:\> perl -d -e 1

Loading DB routines from perl5db.pl version 1
Emacs support available.

Enter h or `h h' for help.

main::(-e:1):   1
  DB<1> __
```

这时，调试程序就等待你输入命令了。这种特殊的开关并置方法可以用来测试 Perl语句，而你不必编写完整的程序。然后再进行测试、调试、编辑、再测试和再调试。你只需要在调试程序中运行你的语句，直到它能够运行为止。命令行上的 1是指最起码的Perl程序，它是计算为1并且返回1的一个表达式。

### 12.3.2 其他开关

Perl解释程序中的-c开关可供Perl用来查看你的代码，以便找出语句上的问题，但是它实际上并不运行程序：

```
C:\> perl -c Example.pl
Example.pl syntax OK
```

如果出现语句错误， Perl就会输出下面这样一条消息：

```
C:\> perl -c Example.pl
syntax error at Example.pl line 5, near "print"
Example.pl had compilation errors
```

与-w组合起来后， -c开关就能对你的程序进行编译，然后显示 Perl认为适当的警告消息。

当你向知识更丰富的Perl用户或系统管理员了解调试代码的情况时，常常必须提供正在使用的Perl解释程序的版本。目前正在使用的 Perl语言的主要版本是 Perl 5。该解释程序本身有一个可以查看的版本，你可以在命令行上使用开关 -v，便可了解该版本号，如下如示：

```
C:\> perl -v

This is perl, version 5.004_02

Copyright 1987-1997, Larry Wall

Perl may be copied only under the terms of either the Artistic License or the
GNU General Public License, which may be found in the Perl 5.0 source kit.
```

在上面这个代码中，Perl解释程序的版本是5.004_02。若要了解更加详细的信息，比如该解释程序是如何创建的，何时创建的，等等，可以运行带有 -v开关的解释程序，如下所示：

```
C:\> perl -V
Summary of my perl5 (5.0 patchlevel 4 subversion 02) configuration:
  Platform:
    osname=MSWin32, osvers=4.0, archname=MSWin32
    :
Compiler:
    cc='bcc32', optimize='-O', gccversion=
    :
Characteristics of this binary (from libperl):
  Compile-time options: DEBUGGING
  Built under MSWin32
  Compiled at Aug  9 1997 21:42:37
  @INC:    C:\PERL\lib\site    C:\PERL\lib
    c:\perl\lib    c:\perl\lib\site    c:\perl\lib\site  .
```

如果你试图查找Perl解释程序本身的某个问题，比如安装时出现的一个问题，那么上面这个输出可能对你有用。在这个输出的结尾处，请注意 @INC的各个值。这个特定安装的 Perl希望在这些目录中找到它的各个模块。当 Perl安装后，它不能简单地从一个目录移动到另一个目录。该解释程序本身对于何处能找到它的各个模块有一个内置的思路，如果改变这个思路，就会导致Perl到错误的地方去查找它的模块。关于模块的问题，将在第 14学时中详细介绍。

### 12.3.3 空的尖括号与更多的单命令行程序

迄今为止介绍的尖括号运算符（<>）具有两个功能：

1) 如果尖括号中间是文件句柄，尖括号运算符允许你读取文件句柄，比如 <STDIN>。

2) 如果尖括号中间是搜索模式，尖括号运算符能返回与该模式匹配的文件列表，这称为一个glob，比如<*.bat>。

尖括号运算符还有另一个功能。一组尖括号运算符如果中间没有任何东西，那么它可以读取命令行上所有文件的内容；如果没有给出文件名，则可以读取标准输出。有时空尖括号运算符称为菱形运算符（因其形状而得名）。例如，请看下面这个小型Perl程序：

```
#!/usr/bin/perl -w

while(<>) {
    print $_;
}
```

如果将上面的程序保存为Example.pl，那么用下面这个命令行运行该程序：

```
C:\> perl -w Example.pl file1 file2 file3
```

就可使运算符<>读取file1的内容，每次读1行，然后读取file2，接着读取file3。如果没有设定文件，则尖括号运算符从文件句柄 STDIN中读取数据。这个运行特性类似 UNIX实用程序 Sed、awk等的特性，如果命令行上设定了文件，则从文件中读取输入，否则读取标准输入。

Perl程序的参数，即去掉 Perl的参数-w、-c、-d和-e之后，将被存放在称为@ARGV的数组中。例如，对于上面这个代码段的参数， $ARGV[0]将包含file1，$ARGV[1]包含fole2，如此等等。

Perl程序的-n开关可用于将任何-e语句封装在该小程序中：

```
LINE:
while(<>) {
... # Your -e statements here.
}
```

因此，如果要创建一个简短的单命令行程序，从输入数据中删除前导空格，你可以编写下面的命令：

```
C:\> perl -n -e 's/^\s+//g; print $_;' file1
```

上面这个命令实际上运行类似下面这个Perl程序：

```
LINE:
while(<>) {
    s/^\s+//g;
    print $_;
}
```

在上面这个代码段中，名字为file1的文件被打开，并被赋予while循环中的$_，每次1行。该行用S/^\S+//g进行编辑，然后进行输出。-p与-n开关的作用相同，差别在于语句执行后各个文件行便自动输出。因此，重新编写上面这个命令行便产生下面这个命令行：

```
C:\> perl -p -e 's/^\s+//g' file1
```

当你用Perl的单命令行程序来编辑一个文件时，必须注意不要在打开文件进行读取操作的同时，又试图对它进行写入操作，像下面这个例子那样：

```
C:\>perl -p -e 's/\r//g' dosfile > dosfile
```

上面这个代码段试图从称为dosfile的文件中删除回车符。问题是在Perl命令被处理之前，dosfile文件已经被>dosfile改写。编辑文件的正确方法应该是将输入重定向到另一个文件中，并将文件改为它的原始名字，如下所示：

```
C:\>perl -p -e 's/\r//g' dosfile > tempfile
C:\>rename tempfile dosfile
```

有些Perl爱好者认为，编写简短的“单命令行程序（one-liners）”只不过是一种娱乐。他们认为，程序越复杂，功能越多，就越好。Perl Journal是介绍Perl的一份季刊，它在每一期上刊登了许多单命令行程序。

## 12.4 课时小结

在本学时中，我们介绍了如何有效地使用调试程序来查找Perl程序中存在的问题；介绍了尖括号运算符（< >）的另一个功能，它使Perl能够处理命令行上的所有文件；另外，还介绍了如何使用Perl解释程序的-n和-p开关，来编写小型单行Perl程序。

## 12.5 课外作业

### 12.5.1 专家答疑

**问题：**我真的希望Perl有一个图形调试程序。是否存在这样的东西？

**解答：**是的，有几个这样的调试程序。如果你在Windows下使用Perl，Activestate拥有很好的图形调试程序。

**问题：**调试程序不断输出的main::究竟是个什么东西？

**解答：**它与Perl的程序包命名约定有关。它的一些情况将在下一个学时中介绍，因此现在你不必对它考虑太多。

**问题：**Perl是否还有别的命令行开关？

**解答：**是的，还有一些。你可通过在线手册查看这些开关的完整列表。若要访问这些信息，请在命令提示符处键入 perldoc perlrun。

### 12.5.2 思考题

1) 程序清单12-1中存在哪些错误？

2) 如果命令行上没有给定文件，那么读取 <>时将返回

a. undef。

b. 来自标准输入的数据行。

c. True。

3) Prel调试程序执行时能够输出 Prel语句，这称为跟踪方式。你如何使调试程序进入跟踪方式呢？（提示：必须查看调试程序的帮助消息，才能回答这个问题。）

a. 使用T命令，使之进入跟踪方式。

b. 使用t命令，使之进入跟踪方式。

### 12.5.3 解答

1) 首先，在第15行中，范围（20..0）无效。范围运算符“ .. ”不应该是降序，而应该是升序。这一行应该改为 for（$_=20；$_>-1；$_-）循环，将范围倒过来（ 0..20），或者使用类似的范围。其次，在第 10行上， $mess=s/glasses/glass/看上去像是对 $mess的替换表达式，但实际上并非如此。替换实际上是对 $_进行的，因为赋值运算符（=）应该是个连接运算符（=~）。

2) 答案是b。如果没有给定文件名，那么 <>便开始读取STDIN。

3) 答案是b。t命令能够在程序执行时输出程序的所有语句。 T命令用于输出堆栈跟踪记录，这是当前正在执行的函数、调用该函数的函数等的列表。