第5学时　进行文件操作

到现在为止，我们介绍的Perl程序都是独立的程序。除了向用户提供消息和接收来自键盘的输入信息外，它们无法与外界进行通信。这种状况将要改变。

Perl是一种能够进行文件输入和输出（文件I/O）的非常出色的语言。Perl的标量能够延伸，以便将尽可能长的记录存放在文件中，另外，Perl的数组能够扩展，以便存放文件的全部内容，当然这必须是在内存允许的情况下才能做到。当数据包含在 Perl的标量和数组中时，可以对该数据进行不受限制的操作，并且可以编写新的文件。

当你读取数据或者将数据写入文件时， Perl总是尽量设法不妨碍你的操作。在某些地方，Perl的内置语句甚至进行了优化，以便执行常用类型的 I/O操作。

在本学时中，我们将要介绍Perl怎样使你能够访问文件中你可以使用的所有数据。

在本学时中，你将要学习:

- 如何打开和关闭文件。
- 如果将数据写入文件。
- 如何从文件中读取数据。
- 如何使你编写的Perl程序具备保护功能，从而使其更加强大。

5.1 打开文件

若要在Perl中读取文件或写文件，必须打开一个文件句柄。 Perl中的文件句柄实际上是另一种类型的变量，它们可以作为在你的程序与操作系统之间对某个特定文件使用的非常方便的一个引用（即句柄，如果你愿意这样说的话）。句柄包含了关于如何打开文件和你在文件中读(或写)到了什么位置等信息。它们还包含了用户定义的关于如何读写文件的属性。

在前面的课程中，你已经熟悉了一个句柄，即 STDIN。该句柄是在启动程序时 Perl自动赋予你的，它通常与键盘设备相连接（后面还要更加详细地介绍 STDIN）。文件句柄名字的格式与第2学时介绍的变量名基本相同，不同之处是句柄的名字前面没有类型标识符（ $、@）。由于这个原因，句柄名字最好使用大写字母，这样就不会与 Perl的当前和将来的保留字 foreach、else和if等发生冲突。

也可以将字符串标量或能够返回字符串的任何东西（如函数）用作文件句柄名。这种类型的句柄名称为间接句柄。描述它们的用法会给 Perl的初学者造成一些混乱。关于间接句柄的详细说明，请参见 perlfunc手册页中关于open的在线文档。

每当需要访问磁盘上的文件时，必须创建一个新的文件句柄，并且打开该文件句柄，进行相应的准备。当然必须使用 open函数来打开文件句柄。 Open的句法如下：

```
open (filehandle, pathname)
```

open函数将文件句柄作为它的第一个参数，将路径名作为第二个参数。路径名用于指明

要打开哪个文件，因此，如果没有设定完整的路径名，比如 c:/windows/system/，那么open函数将设法打开当前目录中的文件。如果 open函数运行成功，它将返回一个非 0值。如果open函数运行失败，它返回undef（假）：

```
if (open(MYFILE, "mydatafile")) {
    # Run this if the open succeeds
} else {
    print "Cannot open mydatafile!\n";
    exit 1;
}
```

在上面这个代码段中，如果open运行成功，它计算得出的值是真，而 if代码块则用打开的文件句柄MYFILE来运行。否则，文件不能打开，代码的 else部分开始运行，这表示出现了错误。在许多Perl程序中，这个“打开或失败”语句是使用 die函数来编写的。die函数用于停止Perl程序的执行，并且输出下面这个出错消息：

```
Died at scriptname line xxx
```

在这个消息中，scriptname是Perl程序的名字，xxx是遇到die的行号。die和open这两个函数常常以下面的形式同时出现：

```
open(MYTEXT, "novel.txt") || die;
```

这一行代码可以读作“打开或撤消”，它有时表示你想要让程序如何处理没有打开的文件。如果open运行没有成功，也就是说它返回 FALSE，那么逻辑 OR（||）必须计算右边的参数（die）；如果open运行成功了，也就是说它返回 TRUE，那么就不要计算die的值。这个习惯用语也可以用逻辑OR的另一个符号or来书写。

当你完成文件句柄的操作后，将文件句柄关闭，这是个很好的编程习惯。关闭文件句柄的操作，将通知操作系统说，该文件句柄可以重复使用，同时，尚未为文件句柄写入的数据现在可以写入磁盘。另外，你的操作系统只允许打开规定数量的文件句柄，如果超过这个数量，你就不能打开更多的文件句柄，除非关闭某些句柄。若要关闭文件句柄，可以使用下面这个close函数：

```
close(MYTEXT);
```

如果文件句柄名字重复使用，即另一个文件用相同的文件句柄名字打开，那么原始文件句柄将先被关闭，然后重新打开。

5.1.1 路径名

到现在为止，我们只是用类似 novel.txt的名字来打开文件。当试图打开没有设定目录名的文件名时，Perl假定该文件是在当前目录中。若要打开位于另一个目录中的文件，必须使用路径名。路径名用于描述Perl为了打开系统中的文件而必须使用的路径。

若要设定路径名，可以使用你的操作系统期望的方式来设定，如下面这个例子所示：

```
open(MYFILE, "DISK5:[USER.PIERCE.NOVEL]") || die;     # VMS
open(MYFILE, "Drive:folder:file") || die;             # Macintosh
open(MYFILE, "/usr/pierce/novel") || die;             # Unix.
```

在Windows和MS-DOS系统下，设定Perl中的路径名时可以使用反斜杠作为路径名分隔符，比如\Windows\user\pierce\novel.txt。需要注意的惟一的问题是：当在带有双引号字符串中使用反斜杠分隔符路径名时，反斜杠字符序列将被转换成一个特殊的字符。请看下面的例子：

```
open(MYFILE, "\Windows\users\pierce\novel.txt") || die;    # WRONG
```

这个例子可能运行失败，因为双引号字符串中的 \n是个换行符，而不是字母 n，而且其他的所有反斜杠将被Perl悄悄地删除。下面是打开文件的正确方法：

```
open(MYFILE, "C:\\Windows\\users\\pierce\\novel.txt") || die; # Right, but messy.
```

为了使这行代码看是去更好一些，也可以使用 Windows和MS-DOS中的正斜杠（/）来分隔路径中的各个元素。Windows和DOS能够正确地对它们进行转换，请看下面的代码：

```
open(MYFILE, "C:/Windows/users/pierce/novel.txt") || die;    # Much nicer
```

你设定的路径名可以是绝对路径名，例如 UNIX中的/home/foo或Windows中的c:/windows/win.ini，也可以是相对路径名，如 UNIX中的../junkfile或Windows中的../bobdir/bobsfile.txt。open函数也能够接受Microsoft Windows下的通用命名约定（UNC）路径名。UNC路径名的格式如下：

```
\\machinename\sharename
```

Perl能够接受使用反斜杠或正斜杠的 UNC路径名，如果你的操作系统的网络和文件共享特性的设置正确的话，它能打开远程系统上的文件，请看下面的例子：

```
open(REMOTE, "//fileserver/common/foofile") || die;
```

在Macintosh计算机上，路径名是按卷、文件夹，然后文件这样的顺序来设定的，各个元素之间用冒号分开，如表5-1所示。

表5-1 MacPerl路径名的说明符

Macintosh路径	含 义
System:Utils:config	系统驱动器，文件夹实用程序，文件名 config
MyStuff:friends	从该文件夹向下到文件夹 MyStuff，文件名friends
ShoppingList	该驱动器，该文件夹，文件名 ShoppingList

5.1.2 出色的防错措施

在计算机上进行编程肯定会产生一种自我陶醉的感觉。程序员会说：“这次程序一定能够运行，”或者说：“我已经找到了所有的错误。”你在工作中产生这种自豪感在某种意义上讲当然是很好的，对程序的改进往往是出于这样一种信念，即你能够做到似乎不可能的事情。不过，你在编程过程中要切忌盲目自信。

自从计算机最初使用程序以来，人们就产生了这种看法。 Fredrick P.Brooks曾经在他的经典著作“ Mythical Man-Month ”中说：“所有程序员都是乐观主义者。但是，由于我们的思路常常出错，因此在编程的时候总是会犯错误，所以我们的乐观主义是没有理由的。”

到现在为止，你看到的所有代码段和程序练习都是用来处理内部数据（对数字进行因子分解，对数据进行排序等等）的，用户输入的信息很简单。当你进行文件的处理时，你的程序就要与外部信息源进行交流，而这些程序是无法控制外部信息源的。当你与不是在你的计算机上的数据源（比如网络上的数据）进行通信时，情况也是如此。请记住，如果某个地方可能出现错误，那么它就可能出错，因此你应该根据情况来编写程序。用这种方法来编写程序称为防错性编程，如果你进行防错编程，那么从长远来说你会感到更加乐观。

每当程序要与外界进行交互操作时，比如打开一个文件句柄，始终必须确保操作成功之后才能进行下一步操作。有人调试了上百个程序，在这些程序中，程序员要求操作系统执行某项操作，但是却不检查结果，因此就导致程序的错误。甚至当你的程序只是个“程序示例”，或者是个“简易程序”时，也应该进行程序检查，以确保程序能够产生你期望的结果。

5.1.3 以适当的方式运行die函数

在Perl中，die函数可以用来在出现错误的时候停止解释程序的运行，并输出一条有意义的出错消息。正如你在前面已经看到的那样，只要调用 die函数，就能够输出类似下面的消息：

```
Died at scriptname line xxx
```

die函数也可以带有一系列的参数，这些参数将取代默认消息而被输出。如果消息的后面没有换行符，那么消息的结尾就附有 at scriptname line xxx字样：

```
die "Cannot open";      # prints "Cannot open at scriptname line xxx"
die "Cannot open\n";    # prints "Cannot open"
```

Perl中有一个特殊的变量$!，它总是设置为系统需要的最后一个操作（比如磁盘输入或输出）的出错消息。当 $!用于数字上下文时，它返回一个错误号，这个号可能对任何人都没有什么用处。在字符串上下文中， $!返回来自你的操作系统的相应的出错消息：

```
open(MYFILE, "myfile") || die "Cannot open myfile: $!\n";
```

如果由于文件不存在，上面这个代码段中的代码运行失败，那么输出的消息将类似 cannot open myfile : a file or directory in the path does not exist，这个出错消息很好。在你的程序中，好的出错消息应该能够指明什么错了，为什么出错，你想怎么办。如果程序的某个方面出错了，那么好的诊断程序能够帮助你找出问题的所在。

不要使用$!的值来检查系统函数的运行是失败还是成功。只有当系统执行一项操作（比如文件输入或输出）之后， $!才有意义，并且只有在该操作运行失败后，$!才被设置。在其他时间中， $!的值几乎可以是任何东西，并且是毫无意义的。

不过有时并不想使程序停止运行，只是想要发出一个警告。若要创建这样的警告， Perl有一个warn函数可供使用。 warn的运行方式与die完全一样，你可以从下面这个代码中看出来，不过差别是它的程序将保持运行状态：

```
if (! open(MYFILE, "output")) {
    warn "cannot read output: $!";
} else {
    :   # Reading output...
}
```

5.2 读取文件

可以使用两种不同的方法读取 Perl的文件句柄。最常用的方法是使用文件输入运算符，也叫做尖括号运算符（<>）。若要读取文件句柄，只需要将文件句柄放入尖括号运算符中，并将该值赋予一个变量：

```
open(MYFILE, "myfile") || die "Can't open myfile: $!";
$line=<MYFILE>;         # Reading the filehandle
```

标量变量中的尖括号运算符能够读取来自文件的一行输入。当该文件被读完时，尖括号返回值undef。

“一行输入”通常是指发现第一个行尾序列之前的文本流。在 UNIX中，行尾是一个换行符（ASCII 10）；在DOS和Windows中，它是回车符与换行符的序列（ASCII 13、10）。这个默认行尾值可以被Perl进行操作，产生某些有趣的结果。这个问题将在第12学时中介绍。

若要读取和输出整个文件，那么如果MYFILE是个开放的文件句柄，你可以使用下面的代码：

```
while(defined($a=<MYFILE>)) {
    print $a;
}
```

== "while ($a=<MYFILE>) {

结果表明，读取文件句柄的快捷方式是使用 while循环。如果尖括号运算符是while循环的条件表达式中的唯一元素，那么 Perl将自动把输入行赋予该特殊变量 $_，并重复运行该循环，直到输入耗尽为止：

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

while循环将负责把输入行赋予 $_，并确保文件句柄尚未耗尽（称为文件结束）。这个奇特的运行特性只有在while循环中才能发生，并且只有在尖括号运算符是条件表达式中的唯一表达式时才能发生。

请记住，在Perl中用文件句柄读取的所有数据中，除了包含文件行中的文本外，还包含行尾字符。如果只需要文本，请在输入行上使用 chomp，以便舍弃行尾字符。

在列表上下文中，尖括号运算符能够读取整个文件，并将它赋予该列表。文件的每一行被赋予列表或数组的每个元素，如下所示：

```
open(MYFILE, "novel.txt") || die "$!";
@contents=<MYFILE>;
close(MYFILE);
```

在上面这个代码段中，文件句柄 MYFILE中的剩余数据也被读取并赋予 @contents。文件novel.txt的第一行被赋予@contents:$contents [0] 的第一个元素。第二行被赋予 $content [1]，如此等等。

在大多数情况下，将整个文件读入一个数组（如果文件不是太大），是处理文件数据的一种非常容易的方法。你可以来回通过该数组，对数组元素进行操作，并可使用数组和标量的所有运算符来处理该数组的内容，而不必担心会出现什么问题，因为你实际上只是对数组中的一个文件拷贝进行操作。程序清单 5-1显示了对内存中的文件可能进行的某些操作。

程序清单5-1 对文件进行倒序

```
1:  #!/usr/bin/perl -w
2:
3:  open(MYFILE, "testfile") || die "opening testfile: $!";
4:  @stuff=<MYFILE>;
```

```
5:   close(MYFILE);
6:   # Actually, any manipulation can be done now.
7:   foreach(reverse(@stuff)) {
8:           print scalar(reverse($_));
9:   }
```

如果该文件的测试文件包含文本 I am the very model of a modern major-general, 那么程序清单5-1中的程序将产生下面的输出：

```
.lareneg-rojam nredom a
fo ledom yrev eht ma I
```

第1行：这一行包含到达解释程序的路径（可以修改它，使之适合你的系统的需要）和开关-w。请始终使警告特性处于激活状态！

第3行：使用文件句柄FH打开文件的测试文件。如果该文件没有正确地打开， die函数便开始运行，并产生一个出错消息。

第4行：测试文件的整个内容被读入数组 @stuff。

第7行：数组@stuff被倒序，第1行变成最后一行，依次类推。 foreach语句遍历产生的列表。倒序后的列表的每一行被赋予 $_，同时foreach循环体被执行。

第8行：列表的每一行（现在位于 $_中）本身也进行倒序，由从左到右，变为从右到左，并且输出倒序后的每一行。它需要使用 scalar函数，因为print期望一个列表。另外，在列表上下文中， reverse用于对列表进行倒序，这样， $_就不会发生任何问题。 scalar函数将强制reverse进入标量上下文中，同时它按逐个字符对 $_进行倒序。

只有对小型文件，才能将整个文件读入数组变量，以便进行操作处理。如果要将非常大的文件读入内存，虽然是允许的，但是可能导致 Perl占用系统中的全部可用内存。

如果将太大的文件读入内存，或者进行其他的一些操作，从而使你占用的内存超出了 Perl能够使用的内存， Perl就会显示下面这条出错消息：

```
Out of memory!
```

这时你的程序就会终止运行。如果当一次性地将整个文件读入内存时出现了这种情况，你应该考虑每次读入文件的一行。

5.3 写入文件

若要将数据写入文件，首先必须有一个打开的文件句柄，以便进行写入操作。打开一个文件以便进行写入操作的句法如下：

```
open(filehandle, ">pathname")
open(filehandle, ">>pathname")
```

第一个语句行你应该是熟悉的，不过在路径名的前面有一个 >。>符号用于告诉 Perl，pathname设定的文件应该用新数据改写，而现有的任何数据都应该删除，同时 filehandle是打开的，可以用于写入。在第二个例子中， >>告诉Perl打开该文件，以便进行写入操作，但是，如果文件存在，那么将数据附加到该文件的结尾处。现在请看下面这些例子：

```
# Overwrite existing data, if any
open(NEWFH, ">output.txt") || die "Opening output.txt: $!";
# Simply append to whatever data may be there.
open(APPFH,   ">>logfile.txt") || die "Opening logfile.txt: $!";
```

到现在为止，你的 Perl程序几乎还不可能对任何东西造成损害。现在，你已经知道如何将数据写入文件，那么必须非常注意只能对你想要修改的文件进行写入操作。在操作系统文件非常容易受到损坏的系统（Windows 95/98、Mac）上，如果不小心将数据写入文件，那么就会损坏你的操作系统。你应该非常清楚要将数据写入什么文件。要想对不小心用 >打开的文件中的数据进行还原，那几乎是不可能的。想要清除你不小心用 >>打开的文件中的数据，那也是困难的，因此你要格外小心。

当完成了对打开的文件句柄的写入操作后，关闭该文件句柄是特别重要的。当你对文件进行写入操作时，操作系统并不将数据存放到磁盘，它只是将数据放入缓存，然后随时进行写入操作。close函数通知操作系统说，你已经完成写入操作，数据应该移到磁盘上的永久性存储器中：

```
close(NEWFH);
close(APPFH);
```

当你打开了用于写入操作的文件句柄时，将数据写入文件实际上是非常容易的，因为你已经熟悉print函数。到现在为止，你一直使用 print函数将数据显示在屏幕上。 print函数实际上可以用于将数据写入任何文件句柄。将数据写入文件句柄的句法如下：

```
print filehandle LIST
```

filehandle是要将数据写入到的文件句柄， LIST是要写入的数据的列表。

在print语句中，请注意文件句柄名与列表之间没有逗号，这一点很重要。在列表中，逗号用于分隔各个项目，迄今为止一直是这样做的。如果在文件句柄与列表之间没有逗号，那么Perl就会知道print后面的标记是个文件句柄，而不是列表中的文件元素。如果忘记使用这个逗号，但是打开了 Perl的警告特性，那么 Perl就会向你发出下面这条警告消息： No comma allowed after filehandle（文件句柄后面不允许有逗号）。

现在请看下面这个代码：

```
open(LOGF, ">>logfile") || die "$!";
if (! print LOGF "This entry was written at", scalar(localtime), "\n" ) {
    warn "Unable to write to the log file: $!";
}
close(LOGF);
```

在上面这个代码段中，名字为logfile的文件被打开，以便进行附加操作。 print语句将一条消息写入LOGF文件句柄。print的返回值被检查核实，如果记录项无法输出，便发出警告消息。然后文件句柄被关闭。

可以同时打开多个文件句柄，以便进行读取和写入操作，正如下面这个代码段显示的那样：

```
open(SOURCE, "sourcefile") || die "$!";
open(DEST, ">destination") || die "$!";
@contents=<SOURCE>;                 # Slurp in the source file.
print DEST @contents;             # Write it out to the destination
close(DEST);
close(SOURCE);
```

上面这个代码段实现了一个简单的文件拷贝。实际上同时进行读取和写入操作可以将例程缩短一些：

```
print DEST <SOURCE>;
```

由于print函数希望有一个列表作为其参数，因此 <SOURCE>是在列表上下文中计算的。当尖括号运算符在列表上下文中进行计算时，整个文件将被读取，然后输出到文件句柄DEST。

5.4 自由文件、测试文件和二进制数据

文件和文件系统不一定只包含你写入文件的数据。有时文件句柄代表的不仅仅是个简单文件。例如，文件句柄可以被附加给键盘设备、网络套接字和大容量存储设备，如磁带驱动器。

另外，文件系统也可以包含所谓的关于你的文件的元数据。 Perl可以用来查询文件系统，以便确定你的文件究竟有多大，上次修改文件的时间，谁修改了这个文件，以及文件中究竟有些什么信息等。在有些操作系统中，文件的元数据甚至能够确定文件是作为文本文件还是二进制文件来处理。

5.4.1 自由文件句柄

最初，Perl是个UNIX实用程序，有时，甚至在非UNIX平台上也会出现它的许多蛛丝马迹。当你的Perl程序启动运行时，它会得到3个自动打开的文件句柄。它们是STDOUT（标准输出）、STDIN（标准输入）和STDERR（标准错误）。按照默认设置，它们均与你的终端相连接。

当你键入数据时， Perl能够读取来自STDIN文件句柄的输入：

```
$guess=<STDIN>;
```

当想要显示输出信息时，可以使用 print函数。按照默认设置， print使用STDOUT文件句柄：

```
print "Hello, World!\n";                    # is the same as...
print STDOUT "Hello, World!\n";
```

在第12学时中，我们将要介绍如何改变print的默认文件句柄。

STDERR通常也可以设置为你的终端，但是它用于显示出错消息。在 UNIX中，出错消息和正常的输出可以发送到不同的显示设备，并且，将出错消息写入 STDERR文件句柄是一种传统的做法。die和warn函数都能够将它们的消息写入 STDERR。如果你的操作系统没有单独的报告错误的文件句柄，比如像 Windows或DOS那样，那么STDERR输出将被送往STDOUT设备。

在UNIX中将出错消息和输出信息送往其他设备的问题不在本书讲解的范围之内，而且这方面的技术将根据你使用的 shell程序而各有差别。凡是介绍UNIX的好的著作都会全面介绍这方面的内容。

5.4.2 二进制文件

有些操作系统、比如VMS、Atari ST，尤其是Windows和DOS，都对二进制文件（原始文件）与文本文件进行了区分。这种区分产生了许多的问题，因为Perl无法知道它们之间的差别，并且你也不希望它进行这样的区分。文本文件只不过是以行尾字符(称为记录分隔符)结束的记

录。二进制文件是指需要进行内部转换的许多信息位的集合，如映像、程序和数据文件等。

当你将数据写入一个文本文件时， Perl将\n字符序列转换成你的操作系统使用的记录分隔符。在UNIX中，\n变成一个ASCII 10（LF）；在Macintosh上，\n转换成ASCII 13（CR）；在DOS和Windows系统上，它变成序列ASCII 13和ASCII 10（CRLF）。当你写入文本时，这个运行特性是合适的。

当写入二进制数据即GIF文件、EXE文件或MS word文档时，这种转换是你所不想要的。每当你真的想要写入二进制数据并且不希望 Perl或操作系统对它进行转换时，你必须使用binmode函数，将文件句柄标记为二进制文件。应该在文件句柄打开之后和对它进行输入输出之前使用binmode：

```
open(FH, "camel.gif") || die "$!";
binmode(FH);            # The filehandle is now binary.
# Start of a valid GIF file…
print FH "GIF87a\056\001\045\015\000";
close(FH);
```

只能对文件句柄使用一次binmode函数，除非将它关闭后再重新打开。如果在不能区分二进制文件和文本文件的系统（UNIX或Macintosh）上使用binmode函数，不会造成任何危害。

5.4.3 文件测试运算符

在打开文件之前，有时应该了解一下该文件是否存在，或者该文件是否是个目录，或者打开文件是否会产生一个permission denied（不允许访问）的错误。对于这些情况， Perl提供了文件测试运算符。这个文件测试运算符的句法如下：

```
-X filehandle
-X pathname
```

这里的x是指你想进行的特定测试，而filehandle是你想要测试的文件句柄。也可以在不打开文件句柄的情况下测试一个pathname。表5-2列出了一些运算符。

表5-2 部分文件测试运算符一览表

运 算 符	举 例	结 果
-r	-r ‘ file ’	如果可以读取‘ file ’，则返回真
-w	-w $a	如果$a中包含的文件名是可以写入的文件名，则返回真
-e	-e ‘ myfile ’	如果‘ myfile ’存在，则返回真
-z	-z ‘ data ’	如果‘ data ’存在，但是它是空的，则返回真
-s	-s ‘ data ’	如果‘ data ’存在，则返回‘ data ’的大小，以字节为计量单位
-f	-f ‘ novel.txt ’	如果‘ novel.txt ’是个普通文件，则返回真
-d	-d ‘ /tmp ’	如果‘ /tmp ’是个目录，则返回真
-T	-T ‘ unknown ’	如果‘ unknown ’显示为一个文本文件，则返回真
-B	-B ‘ unknown ’	如果‘ unknown ’显示为一个二进制文件，则返回真
-M	-M ‘ foo ’	返回程序启动运行以来‘ foo ’文件被修改后经过的时间（以天数计算）

可以通过在线文档来查看文件测试运算符的完整列表。在命令提示符处键入 perldoc perlfunc，查看“ Alphabetical List of Perl Functions ”（按字母顺序排列的Perl函数列表）这一节的内容。

文件测试运算符可以像下面这样在改写文件之前用于检查该文件是否存在，或者用于核

实用户的输入信息，也可以用于确定需要的目录是否存在以及是否可以进行写入操作：

```
print "Save data to what file?";
$filename=<STDIN>;
chomp $filename;
if (-s $filename ) {
    warn "$file contents will be overwritten!\n";
    warn "$file was last updated ",
-M $filename, "days ago.\n";
}
```

5.5 课时小结

本学时我们介绍了如何在perl中打开和关闭文件句柄。可以使用open函数来打开文件，使用close函数来关闭文件。当文件句柄打开时，它们可以使用 <>或read函数进行读取，使用print进行写入操作。另外，我们还介绍了你的操作系统处理文件时使用的一些特殊方法，以及如何使用binmode来处理文件。

此外，希望你也了解到了一些关于防错编程的方法。

5.6 课外作业

5.6.1 专家答疑

问题：我的open语句经常出错，我不知道究竟问题何在。请问问题究竟在哪里？

解答：首先，请检查open语句的句法是否正确。应该确保打开的是正确的文件名。如果希望更加有把握的话，请在 open的前面放上文件名。如果打算写入文件，请务必在文件名的前面加上一个>，你必须这样做。最重要的是，你有没有使用 open（）|| die “ $! ”;语句来检查open的退出状态呢？die消息在帮助你查找错误方面是非常重要的。

问题：我对文件进行写入操作，可是什么也写不进去。我的输出数据到哪里去了呢？

解答：你的文件句柄正确打开了吗？如果使用了错误的文件名，你的数据就会送往别的文件。常见的错误是在路径名中使用反斜杠并且用双引号将路径名括起来，以便打开文件进行写入操作，如下所示：

```
open(FH, ">c:\temp\notes.txt") || die "$!";    #WRONG!
```

这行代码创建了一个文件，称为c:（tab）emp(newline)otes.txt，也许这不是你想要创建的文件。另外，你应该确保open函数运行成功。除非你激活了Perl的警告特性，否则，如果你将数据写入一个尚未正确打开的文件， Perl就会悄悄地删除输出数据。

问题：当我试图打开一个文件时， open运行失败了， perl报告说permission denied（不允许访问）。为什么？

解答：Perl要遵守你的操作系统的文件安全性规则。如果你不拥有对文件、目录或者文件所在驱动器的访问权，那么Perl也无权访问它们。

问题：我如何才能一次读取一个字符？

解答：Perl的函数getc能够从文件读取单字符输入。从键盘读取单字符就比较困难，因为一次输入一个字符要取决于你的操作系统。当你学习了第 15学时中关于模块的内容后，并且读到第16学时中的专家答疑时，请查看里面的有关说明。在那里的专家答疑中，详细介绍了如何从各种不同的平台读取单个字符的方法，并且配有许多代码举例。大多数代码不属于本

书讲解的范围。

问题：如何才能使其他程序能够同时将数据写入同一个文件？

解答：你所说的问题称为文件锁定。文件锁定将在第 15学时中介绍。应该说明的是，这项操作并不太容易，也不是太难。

5.6.2 思考题

1) 为了打开一个名叫data的文件，以便写入数据，你应该使用下面的哪个代码：

a. open（FH,“ data ”, write）；

b. open（FH,“ data ”）；并且只是输出到FH

c. open（FH,“ >data ”）|| die “ Cannot open data: $! ”;

2)（-M $file > 1 and -s $file）是真，如果

a. $file已经在几天以前被修改，并且里面有数据。

b. 该表达式不可能是真。

c. $file可以写入，并且里面没有数据。

5.6.3 解答

1) 答案是c。选择a是不能成立的，因为这不是 open如何打开一个文件以便进行写入操作的问题；选择b也是不行的，因为它打开的文件句柄只能读取。 c 是正确的，因为它才符合你的要求，并且使用正确的形式来进行错误检查。

2) 答案是a。-M返回文件已经存在的天数（>1是指一天以上），如果文件里面有数据，则-s返回真。

5.6.4 实习

- 修改第4学时中的Hangman程序，以便从数据文件中取出可能的单词的列表。