

第三章 Perl高级语法

本章内容

- 3.1 数组高级技巧
- 3.2 关联数组高级技巧
- 3.3 基本函数
- 3.4 子程序
- 3.5 文件处理
- 3.6 文件和目录操作
- 3.7 模式匹配技巧
- 3.8 引用
- 3.9 包、模块、对象

3.7 模式匹配技巧

1、常规表达式

使用常规表达式(**regular expression**), 实现对字符串的强大处理能力。

常规表达式, 也翻译为正则表达式或文字处理模式, 即对文本进行筛选, 只匹配特定的字符串。一旦匹配到一个字符串, 就可以从大量的文本中将其抽取出来, 或者利用另一个字符串来替代这个字符串;

常规表达式的使用格式为 **/pattern/**。

2、操作符

匹配操作符

匹配操作符有” =~” 和” !~”

” =~” 检验匹配是否成功，如：

```
$result = $var =~ /abc/;
```

表示在\$var字符串中找abc模式，若找到则返回1
赋给\$result，不匹配则返回0赋给\$result

” !~” 的情况与” =~” 相反

★这两个操作符常用于条件控制中，如：

```
if ($question =~ /please/) {  
    print ("Thank you for being polite!\n");  
}  
else {  
    print ("That was not very polite!\n");  
}
```

替换操作符

替换操作符的语法为：

s/pattern/replacement/

表示将字符串中与pattern匹配的部分换成replacement，
例如：

```
$string = "abc123def";  
$string =~ s/123/456/;
```

则结果为：

```
$string = "abc456def";
```

翻译操作符

语法如下:

tr/string1/string2/

表示将string1中的第一个字符替换为string2中的第一个字符,把string1中的第二个字符替换为string2中的第二个字符,依此类推.

```
$string = "abcdefghicba ";  
$string =~ tr/abc/def/;
```

结果为:

```
$string = "defdefghifed ";
```

3、模式中的特殊字符技巧

字符 “+”

意味着一个或多个相同的字符，如：

```
$var =~ /de+f/;
```

模式表示def, deef, deef等

“+” 字符尽可能地多匹配相同字符

例如:模式/ab+/在字符串abbc中匹配的是abb，而不是ab

字符” []” 和” [^]”

“[]” 意味着匹配一组字符中的一个，如：

```
$var =~ /a[0123456789]c/;
```

“[^]” 表示除其之外的所有字符，如：

```
$var =~ /d[^eE]f/;
```

字符” *” 和” ?”

字符 “*” ,匹配0个、1个或多个相同字符， 如：

```
$var =~ /de*f/;
```

匹配:df,def,deef等

字符 “?” ,匹配0个或1个该字符， 如：

```
$var =~ /de?f/;
```

匹配:df或def

转义字符

如果想在模式中包含通常被看作特殊意义的字符,须在其前加斜线” \”,如在模式中包含+, *, ?, [] 和[^]等,则需要表示为:

`\+, *, \?, \[, \[^\].`

举例:

```
$var =~ /de\[tyr\]f/;
```

匹配:de[tyr]f

匹配任意字母或数字

任意小写字母: `[a-z]`

任意大写字母: `[A-Z]`

任意数字: `[0-9]`

任意大小写字母数字: `[0-9a-zA-Z]`

例如:

```
$var =~ /de[0-9a-zA-Z]f/;
```

锚模式

锚模式列表

锚	描述
<code>^</code> 或 <code>\A</code>	仅匹配串首
<code>\$</code> 或 <code>\Z</code>	仅匹配串尾
<code>\b</code>	匹配单词边界
<code>\B</code>	单词内部匹配

例如:

`/^def/`表示只匹配以def打头的字符串

`/def$/`只匹配以def结尾的字符串

`/^def$/`只匹配字符串def

字符范围转义

字符范围转义表

转义字符	描述	范围
<code>\d</code>	任意数字	<code>[0-9]</code>
<code>\D</code>	除数字外的任意字符	<code>[^0-9]</code>
<code>\w</code>	任意单词字符	<code>[0-9a-zA-Z]</code>
<code>\W</code>	任意非单词字符	<code>[^0-9a-zA-Z]</code>
<code>\s</code>	空白	<code>[\r\t\n\f]</code>
<code>\S</code>	非空白	<code>[^\r\t\n\f]</code>

```
$var =~ /\d[a-z]/;
```

其它技巧

字符”.”

匹配除换行外的任意一个字符

字符对{ }

匹配指定数目的字符,例如:

`/de{1, 3}f/` 匹配def, deef和deef

`/de{3}f/` 匹配deef

`/de{3, }f/` 表示匹配不少于3个e在d和f之间

`/de{0, 3}f/` 表示匹配不多于3个e在d和f之间

字符”|”

指定两个或多个选择来匹配模式,例如:

`/def|ghi/`

表示匹配def或ghi

部分重用


当模式中匹配相同的部分出现多次时,可用括号括起来,用”\次数”来表示多次引用,以简化表达式,例如:

`^d{2}-d{2}-d{2}/`

表示匹配12-05-92等

指定模式定界符

缺省的模式定界符为斜杠/,但其可以用字母m自行指定,例如:

 定界符
m!de[0-9]f **!**

注意:当用'单引号作为定界符时,不做变量替换;

当用特殊字符作为定界符时,其转义功能或特殊功能不能使用。

4、模式匹配选项

模式匹配选项的用法

选项	描述
g	匹配所有可能的模式
i	忽略大小写
m	将串视为多行
o	只赋值一次
s	将串视为单行
x	忽略模式中的空白

g选项

匹配所有可能的模式，例如：

```
@matches = "balata" =~ /.a/g;
```

结果为：

```
@matches = ("ba", "la", "ta");
```

split函数

语法:

```
@list = split (pattern, string, maxlength);
```

将字符串分割成一组元素的列表.每匹配一次pattern,就开始一个新元素,但pattern本身不包含在元素中.
maxlength是可选项,当指定它时,达到该长度就不再分割

例如：

```
$text = "Michael, Gevin, Mike";  
@name1 = split (/./, $text);  
@name2 = split (/./, $text, 2);
```

结果：

```
@name1 = ("Michael", "Gevin", "Mike");  
@name2 = ("Michael, Gevin");
```

程序举例

例1:输入一条DNA序列,求转录RNA

```
#!/usr/bin/perl -w
# Transcribing DNA into RNA

$DNA = 'ACGGGAGGACGGGAAAATTACTACGGCATTAGC';

print "Here is the starting DNA:\n\n";
print "$DNA\n\n";

# Transcribe the DNA to RNA by substituting all T's with U's.
$RNA = $DNA;
$RNA =~ s/T/U/g;

print "Here is the result of transcribing the DNA to RNA:\n\n";
print "$RNA\n";
```

程序举例

例2:输入一条DNA序列,求反转DNA补序列

```
#!/usr/bin/perl -w
# Calculating the reverse complement of a strand of DNA
$DNA = 'ACGGGAGGACGGGAAAATTACTACGGCATTAGC';
print "Here is the starting DNA:\n\n";
print "$DNA\n\n";

$revcom = reverse $DNA;
# A->T, T->A, G->C, C->G
$revcom =~ s/A/T/g;
$revcom =~ s/T/A/g;
$revcom =~ s/G/C/g;
$revcom =~ s/C/G/g;

print "Here is the reverse complement DNA:\n\n";
print "$revcom\n";
```


程序举例

```
print "\nThat was a bad algorithm, and the reverse complement was wrong!\n";
print "Try again ... \n\n";

# Make a new copy of the DNA
$revcom = reverse $DNA;

# See the text for a discussion of tr///
$revcom =~ tr/ACGTacgt/TGCAtgca/;

# Print the reverse complement DNA onto the screen
print "Here is the reverse complement DNA:\n\n";

print "$revcom\n";

print "\nThis time it worked!\n\n";
```

3.8 引用

引用(指针)

定义:

是一种标量,保存一个值,通过这个值可以判断在 内存的什么地方可以找到另一个值,这种类型的标量称为引用(reference).

类型:

硬引用(Hard reference):直接指向内存中的一个值(称为引用对象)

符号引用(Symbolic reference):只包含它们所指向变量的名字

注:

硬引用是最常使用的一种类型引用类型,比符号引用更有效率,一般的”引用”指的是硬引用.

引用的表示:

在名字前面加上\$ 符号

引用的创建:

在含有引用对象的变量名前面加上单反斜杠运算符 \

引用值的访问:

在引用前加上引用对象的类型标识符,如:

\$reference	→	引用所指向的标量
@reference	→	引用所指向的数组
%reference	→	引用所指向的关联数组

例子:创建对标量数据的引用

```
#!/usr/bin/perl
use strict;
use warnings;
my $variable = 10;
my $reference = \$variable;
print (“$variable\n”);
print (“$reference\n”);
print ($$reference\n);
$variable++;
print (“$variable\n”);
print (“$reference\n”);
print ($$reference\n);
$$reference++;
print (“$variable\n”);
print (“$reference\n”);
print ($$reference\n);
```

打印一个引用,会得到引用类型和引用对象在内存中的地址所组成

```
10
SCALAR(0x8a31018)
10
11
SCALAR(0x8a31018)
11
12
SCALAR(0x8a31018)
12
```

非标量的引用

数组的引用

指向整个数组: `@$reference`

访问特定的数组元素:

(法1): `$$reference[2]`

(法2): `$reference->[2]`

关联数组的引用

指向整个关联数组: `:%$reference`

访问特定的关联数组元素:

(法1): `$$reference{ '键名' }`

(法2): `$reference->{ '键名' }`

例子:对数组和关联数组的引用

```
#!/usr/bin/perl
use warnings;
use strict;
my @array = qw(duck pig horse rooster cow);
my %hash = (duck => "quack",
            pig => "oink",
            horse => "neigh",
            rooster => "cock-a-doodle-doo",
            cow => "moo");

my $arrayReference = \@array;
my $hashReference = \%hash;

sub returnReference
{
    return \@array;
}
```

```
print ("@$arrayReference\n");
```

```
print ("$$arrayReference[1]\n");
```

```
print ("$arrayReference->[1]\n");
```

```
print ("${returnReference()}[1]\n\n");
```

```
print ("$$hashReference{duck}\n");
```

```
print ("$hashReference->{duck}\n\n");
```

```
foreach (keys(%$hashReference)) {
```

```
    print ("The $_ goes $hashReference->{$_}.\n");
```

```
}
```

duck pig horse rooster cow

pig

pig

pig

quack

quack

The cow goes moo.

The rooster goes cock-a-doodle-doo.

The duck goes quack.

The pig goes oink.

The horse goes neigh.

3.9 包、模块、对象

1. 包的相关概念和基本使用方法
2. 模块相关概念和基本使用方法
3. 对象相关概念和基本使用方法

1. 包的相关概念和基本使用方法

包的定义

Perl程序把变量和子程序的名称存储在符号表中,符号表中名字的集合就称为包(package)

语法规则为:

`package packname;`

← 包名

例如:

```
$var = 1;  
package mypack;  
$var = 2;
```

包间切换

例如:

```
#!/usr/bin/perl  
package pack1;  
$var = 26;  
  
package pack2;  
$var = 34;  
  
package pack1;  
print ("$var\n");
```

结果为: 26

包的引用

方法:

在一个包中引用其他包中的变量或子程序，方法是在变量名前面加上包名和双冒号

例如:

```
#!/usr/bin/perl  
package mypack;  
$var = 26;  
package main;  
print ("mypack::var\n");
```

指定无当前包

package;

此时，所有的变量必须明确指出所属包名，否则就无效，直到用package语句指定当前包为止。

例如：

```
#!/usr/bin/perl  
package mypack;  
$var = 26;  
package;  
$mypack::var = 21;  
$var = 21;
```

——→ 出错

包和子程序

包的定义影响到程序中的所有语句,包括子程序.

例如:

```
package mypack;  
sub mysub {  
    local ($myvar);  
}
```

在包mypack外调用子程序mysub时要用
\$mypack::mysub

可以在子程序中切换包

例如:

```
package pack1;  
sub mysub {  
    $var1 = 1;  
    package pack2;  
    $var1 = b2;  
}
```

包的导入

`require 包名.pm`

首先程序会在当前所在的目录中寻找这个包文件,如没有找到,就会到数组@INC指定的目录中查找

2. 模块相关概念和基本使用方法

能完成某一个或多个任务的集合了变量和子程序的部件叫模块；

模块本质上就是一种“包”，它使程序员能更全面地控制模块用户对那个模块的包内的标识符进行引用；

模块文件要存为.pm文件，文件名和定义的包名相同；

use和require的差别：

use是在编译时导入模块和包的，require是在执行时导入的

use只允许导入.pm文件，require还允许指定其它扩展名，如.pl

例子:模块的定义和调用

模块文件firstmodule.pm

```
#!/usr/bin/perl

package FirstModule;

use Exporter;
our @ISA = qw(Exporter);

our @Exporter = qw(@array &greeting);

our @array = (1, 2, 3);

sub greeting {
    print "Modules are handy!";
}

return 1;
```

导入模块程序

```
#!/usr/bin/perl  
use FirstModule;  
print “using automatically imported  
names:\n”;  
print “@array\n”;  
greeting();
```

use语句的其他特点

use 版本号或模块号;

表示在指定的版本号(模块号)同系统中安装的版本号或模块号对比,假如指定的版本号大于当前的版本号,便会产生严重错误,而且程序会立即停止.例如:

use v5.6.0;

利用这一特性,可以编写出只适用于特定版本的代码

预编译指令

是一种特殊的语句，编译器用它来设置自己的编译选项。

`use strict`语句

作用：

可强迫程序员把所有变量都声明为包变量或字典作用域变量，强迫程序员用引号把所有字符串封闭起来，强迫程序员必须明确地调用每一个子程序。

两种特殊形式:

`use strict 'vars';`

检查所有包变量,以确保每个变量名都采用完整形式(包名::变量名)

`use strict 'subs';`

禁止程序员用一个裸字调用一个子程序

关闭: `no strict;`

use warnings语句

作用:

用于警告用户一些可能出现的打字错误,使用了未初始化的变量以及代码里其他潜在的问题

关闭:

no warnings;

use constant语句: 采用常量

例如: use constant PI => 3.14159;

use diagnostics语句

作用:

显示更详尽的错误提示信息

use integer语句

作用:

告诉编译器采用整数运算方式来执行所有数学运算

3. 对象相关概念和基本使用方法

面向对象编程(OOP)

面向对象编程指的是一种编程方法,它将每个事物都看作是对象.

面向对象编程将数据和函数(方法)封装在被称为类的包中.

根据一个类,程序员可以创建一个对象
对象的类型就是一个类

包、模块、类和对象

包（package）：

Perl程序把变量和子程序的名称存储到符号表中，符号表中名字的集合就称为包。

模块（module）

能完成某种一个或多任务的集合了变量和子程序的部件叫模块

类（class）

类实际就是一个包，类包括两个成员要素：

域(field)。定义了类所需要的数据,所以也叫数据。

方法(method)。定义了类的功能。

对象（object）

是对类中数据项的引用

对象的构造及方法的调用:

对象的构造:

对象名 构造函数 类名

\swarrow \swarrow \swarrow

`$ObjectName = new className;`

方法的调用:

`$ObjectName->method(arguments);`

\nearrow

方法

创建类

例子:处理日期的类Date

```
#!/usr/bin/perl
#A simple data class

package Date;

use strict;
use warnings;

sub new
{
    my $date = {the_year => 1000, the_month => 1, the_day =>1};
    bless($date);
    return $date;
}
```

构造函数

内置函数:功能是接收一个引用,
并将其转换为对象

```
sub year
{
  my $self = shift();
  $self->{the_year} = shift() if (@_);
}
return $self->{the_year};

sub month
{
  my $self = shift();
  $self->{the_month} = shift() if (@_);
}
return $self->{the_month};

sub day
{
  my $self = shift();
  $self->{the_day} = shift() if (@_);
}
return $self->{the_day};
```

```
sub setDate
{
    if (@_ == 4) {
        my $self = shift();
        $self->month($_[0]);
        $self->day($_[1]);
        $self->year($_[2]);
    } else {
        print ("Method setDate requires three arguments.\n");
    }
}

sub print
{
    my $self = shift();
    print ($self->month);
    print ("/");
    print ($self->day);
    print ("/");
    print ($self->year);
}

return 1;
```

注意:

以上类Date所在的程序文件存为Date.pm

使用类

例子:使用类Date

```
#!/usr/bin/perl
#using class Date
use Date;
use strict;
use warnings;

my $today = new Date;

$today->setDate(7,14,2000);
print ($today->month());
print ("\n");
$today->print();
print ("\n");
```

类的继承

从现存的类中创建一个新的类，这被称为继承

新类可以使用原来那些类所拥有的域和方法

继承是实现软件可重用性的一种重要手段。

例子:两个类Employee和Hourly, 类Hourly从类Employee继承而来

类Employee

```
#!/usr/bin/perl
#Implementation of class Employee.
package Employee;
use strict;
use warnings;
use Date;
sub new
{
    my $type = shift();
    my $class = ref($type) || $type;
    my $hireDat = new Date;
    my $self = {firstName=>undef, lastName=>undef, hireDay=>$hireDay};
    bless ($self, $class);
    return $self;
}
```

```
sub firstName
{
    my $self = shift();
    $self->{firstName} = shift() if (@_);
    return $self->{firstName};
}
```

```
sub lastName
{
    my $self = shift();
    $self->{lastName} = shift() if (@_);
    return $self->{lastName};
}
```

```
sub hireDay
{
    my $self = shift();
    if (@_) {
        $self->{hireDay}->setDate(@_);
    }else {
        $self->{hireDay}->print();
    }
}

return 1;
```

类Hourly

```
#!/usr/bin/perl
#implementation of class Hourly.
package hourly;
use strict;
use warnings;
use Employee;
use @ISA = ("Employee");

sub new
{
    my $object = shift();
    my $class = ref($object) || $object;
    my $self = $class->SUPER::new();
    $self->{rate} = undef;
    bless($self, $class);
    return $self;
}
```

用于从其他包导入变量和方法



```
sub rate
{ my $self = shift();
  $self->{rate} = shift() if (@_);
  return $self->{rate};
}

return 1;
```

类Employee和类Hourly的使用

创建Employee对象

```
#!/usr/bin/perl
#Using classes Hourly and Employee
use strict;
use warnings;
use Employee;
use Hourly;
my $worker = new Employee;
$worker->firstName("Jason");
$worker->lastName("Black");
$worker->hireDay(8,5,1995);
print ($worker->firstName(), " ", $worker->lastName(), "was hired on",
$worker->hireDay());
print (".\n\n");
```

创建Hourly对象

```
my $hour = new Hourly
$hour->firstName("John");
$hour->lastName("White");
$hour->hireDay("11,30,1999");
$hour->rate(9.50);
```

```
print ($hour->firstName(), "", $hour->lastName(), "was hired on", $hour->hireDay());
print (".\n");
printf ("He makes \$$% .2f per hour .\n", $hour->rate());
```