第三章 Perl高级语法

内 容

3.1 数组高级技巧

1、列表

- 列表的灵活表示
  - 用于整数

```
(1..10)
(2,5..7,10)
(3..30)
```

  - 用于字符串

```
(“aaa”..”aad”)
```

– qw运算符

qw运算符可将一系列用空格分隔的字串转换成一个字串列表,如:

qw(horse donkey mule)

完全等价于:

(‘horse’, ‘donkey’,’mule’)

```
@array = qw(horse donkey mule);
```

2、数组的存取

– 赋值

- 数组对简单变量的赋值

```
@array = (5, 7, 11);
($var1, $var2) = @array;
```

结果为:

```
$var1 = 5;
$var2 = 7;
```

– 数组间拷贝

```
@result = @original;
```

```
@list1 = (2, 3, 4);
@list2 = (1, @list1, 5);
```

结果为:

```
@list2 = (1, 2, 3, 4, 5);
```

```
@smallArrayOne = (5..10);
@smallArrayTwo = (1..5);
@largeArray = (@smallArrayOne, @smallArrayTwo);
```

结果为:

```
@ largeArray = (5, 6, 7, 8, 9, 10, 1, 2, 3, 4, 5 );
```

– 子数组

```
@array = (1, 2, 3, 4, 5);
@subarray1 = @array[0, 1];
@subarray2 = @array[0..3];
@array[1, 2] = @array[2, 1];
```

结果为:

```
@subarray1 =(1, 2);
@subarray2 =(1, 2, 3, 4);
@array = (1, 3, 2, 4, 5);
```

```
@array = (1, 2, 3, 4, 5);
@array[0, 1] = (“string”, 46);
```

结果为:

```
@array = (“string”, 46, 3, 4, 5);
```

– 列表/数组的长度

```
@array = (1, 2, 3);
$scarlar = @array;
($var) = @array;
```

编程技巧:以数组的长度为循环次数可如下编程:

```
#!/usr/bin/perl

@array = (1, 2, 3, 4, 5);

$count = 1;
while ($count <= @array) {
   print ("element $count: $array[$count - 1]\n");
   $count++;
}
```

注意:

访问不存在的数组元素,则结果为NULL

```
@array = (1, 2, 3);
$var = $array[3];
```

如果给超出数组大小的元素赋值,则数组自动增长,原来没有的元素值为NULL

```
@array = (1, 2, 3);
$array[4] = 8;
```

结果:

```
@array = (1, 2, 3, “ ”, 8);
```

3.2 关联数组高级技巧

1、关联数组定义与访问

– 创建关联数组

```
%fruit = (“apples”, 17, “bananas”, 9, “oranges”, 3);
```

注意：用列表给关联数组赋值时，允许使用“=>”来分割下标与值

```
%fruit = (“apples”  => 17, “bananas” => 9, “oranges” => 3);
```

– 访问关联数组元素

```
$var = $fruit {“apples” };
```

注意：标量变量也可作为下标

```
$myfruit = “apples”;
$var = $fruit {$myfruit};
```

– 数组变量复制到关联数组

```
@fruit = (“apples”, 17, “bananas”, 9, “oranges”, 3);
%fruit = @fruit;
```

- 反之，可以把关联数组赋给数组变量

```
%fruit = (“apples”, 17, “bananas”, 9, “oranges”, 3);
@fruit = %fruit;
```

- 关联数组变量之间可以直接赋值

```
%fruit1 = (“apples”, 17, “bananas”, 9, “oranges”, 3);
%fruit2 = %fruit1;
```

- 还可以把数组变量同时赋给一些简单变量和一个关联数组

```
@fruit = (“apples”, 17, “bananas”, 9, “oranges”, 3);
($var1, $var2, %myarray)  = @fruit;
```

– 用关联数组循环

```
%fruit = (“apples”, 17, “bananas”, 9, “oranges”, 3);
foreach $holder (keys(%fruit)) {
  $var = $fruit{$holder};
}
```

一种更有效的循环方式：使用内嵌函数each()，返回一个双元素的列表，其第一个元素为下标，第二个元素为相应的值，最后返回一个空列表

```
%fruit = (“apples”, 17, “bananas”, 9, “oranges”, 3);
while (($holder, $var) = each(%fruit)) {
 …
}
```

注意:

关联数组是随机存储的,因此当用keys()函数访问其所有元素时,不保证元素以何种顺序出现,特别是它们不会以创建时的顺序出现.要想控制关联元素出现的次序,可以用sort()函数对返回值由小到大顺序排列.

```
%fruit = (“apples”, 17, “bananas”, 9, “oranges”, 3);
foreach $holder (sort(keys(%fruit))) {
   $var = $fruit{$holder};
 }
```

2、关联数组元素操作

– 增加元素

```
$fruit{“apples”} = 1;
```

– 删除元素

- 方法:使用delete函数

```
delete ($fruit{“apples”});
```

注意:

要使用delete函数来删除关联数组的元素,这是唯一的方法

不要对关联数组使用内嵌函数push,pop,shift及splice,因为其元素位置是随机的

特殊变量

- 是Perl自带的,可简化编程工作,往往作为默认变量使用
  - $_变量
    - 是许多函数的默认参数
    - 是许多控制结构的默认控制变量

    例如:

```
foreach (列表) {
   语句
}
```

$_每一次循环时负责保存列表中当前元素的值

- 例如:

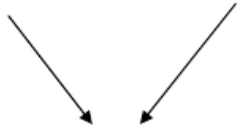
```
Amanda Jeff Search David

Amanda Jeff Search David

Amanda Jeff Search David
```

```
#!/usr/bin/perl

foreach $name(‘Amanda’, ‘Jeff’, ‘Search’, ‘David’) {
  print “$name”;
}
print “\n”;

foreach (‘Amanda’, ‘Jeff’, ‘Search’, ‘David’) {
  print “$_”;
}
print “\n”;

foreach (‘Amanda’, ‘Jeff’, ‘Search’, ‘David’) {
  print;
}
print “\n”;
```

3.3 基本函数

函数格式:

函数名(参数1,参数2…)



可以是标量,数组,关联数组或表达式

分类介绍:

1. 数学函数
2. 字符串处理函数
3. 标量转换函数
4. 数组和列表函数
5. 关联数组函数
6. 错误控制函数

1. 数学函数

函数	说明
sin($x)	求x的正弦(参数x为弧度值)
cos($x)	求x的余弦(参数x为弧度值)
exp($x)	求e的x次方
sqrt($x)	求x的平方根
abs($x)	求x的绝对值
log($x)	求x的自然对数(以e为底)

函数	说明
rand ($x)	随机数函数,返回0到整数x之间的一个浮点数
srand ($x)	初始化随机数生成器,保证每次调用的rand真正随机
hex($x)	返回十六进制数$x的十进制值
oct($x)	返回八进制数$x的十进制值
atan2($y, $x)	返回-□和□ 之间的y/x的反正切值

2. 字符串处理函数

– index函数

返回给定字串在另一给定字串中的位置,如果子串没有找到,函数返回-1

语法规则为:

index(string, substring, startposition)

可以是文本也可以为标量

可选项,表示从位置startposition开始查找,缺省从0开始查找

例如:

```
$position = index(‘perl how to program’, ‘how’);
```

5


```
$position = index(‘perl how to program’, ‘how’, 6);
```

-1

– **rindex函数**

与index功能相同,只是以反方向进行查找,从字符串的最后一个字符开始反向查找.

例如:

```
$position = rindex(‘perl how to program’, ‘how’);
```

– **length函数**

返回字符串长度,即字符数目.

– **substr**函数

语法规则为:

substr(string, offset, length, replacement)

可选项,表示用replacement替代函数返回的子串,缺省函数返回子串

该函数返回从字符位置offset开始,长度为length的子串,可以用replacement来替代函数要返回的子串

– **uc , lc** 函数

用于切换表达式中每个字符的大小写状态并将结果字串返回

- uc函数从参数中返回大写字串
- lc函数从参数中返回小写字串

例:

```
$string2 = uc(‘perl how to program’);
```

结果:

```
PERL HOW TO PROGRAM
```

– ucfirst, lcfirst函数

用于切换给定表达式中第一个字符的大小写状态

- ucfirst函数返回第一个字符为大写的表达式
- lcfirst函数返回第一个字符为小写的表达式

例如:

```
$string2 = ucfirst('perl how to program');
```

```
Perl how to program
```

– join函数

语法:

join(separator, list)

该函数用于将列表中的个体合并成一个字串.返回的字串包括列表中的每个字串,并用特定的分隔符分开.分隔符可以是任何字串,但一般用逗号、冒号或其他适用的字段分隔符。

```
$string2 = join('and', (1..5));
```

结果：

```
1and2and3and4and5
```

– print函数

作用是将引号之间的字串打印到标准的输出设备上---屏幕

同时也是传递给print函数的一个参数或自变量

通常打印出来出来的字符和它们在双引号间的形式是一模一样的,除了转义字符

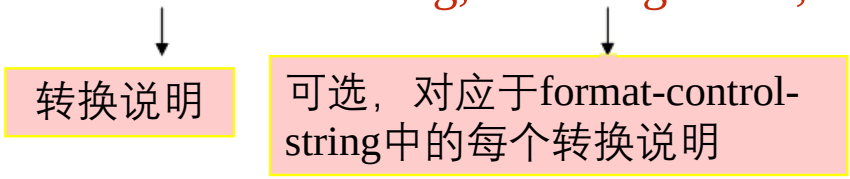
– printf函数

格式化输出函数，具有如下格式化能力：

- 圆整浮点数至十进制的某位
- 以小数点来对齐一列数
- 右对齐和左对齐输出
- 在输出行的准确位置插入文本字符
- 以指数形式来显示浮点数
- 以八进制或十六进制的形式显示无符号整数

- 用固定字段宽度和精度来显示各种标量数据
  语法格式如下：

printf format-control-string, other-argument;



format-control-string与其它输出字串具有相同的外观和规则，只是多加了转换定义符

转换定义符

	转换定义符	说明
整数	%d	显示一个有正负号的二进制整数
	%o	显示一个无正负号的八进制整数
	%u	显示一个无正负号的十进制
	%x或%X	显示一个无正负号的十六进制整数,大写表明显示0-9的数位或A-F的字母,而小写表明显示0-9数位或a-f的字母
浮点数	%e或%E	用指数形式显示一个浮点数
	%f	显示浮点数
	%g或%G	显示一个浮点数可采用浮点形式或指数形式
字符和字串	%c	将指定的字符转换成ASCII
	%s	显示一个字串
	%%	显示一个百分号

- 例如:

```
#!/usr/bin/perl

printf “%d\n”, 455.954;
printf “%d\n”, +455.34;
printf “%d\n”, -455;
printf “%o\n”, 455;
printf “%u\n”, 455;
printf “%u\n”, -455;
printf “%x\n”, 455;
printf “%x\n”, -455;
```

```
455
455
-455
707
455
4294966841
1c7
fffffe39
```

– sprintf函数

sprintf函数与printf函数功能相同，只是sprintf返回的是已格式化的字串，而不是输出。

```
#!/usr/bin/perl
$product = “sweater”;
$price = 39;

$line = sprintf “The %s costs \$%d\n.”, $product, $price;
print $line;
```

结果:

```
The sweater costs $39.
```

3. 标量转换函数

– **chop**函数

用于移除字串最后一个字符，并返回该字串。

- 一般使用这个函数从用户输入行结束处移除换行符，例如标准输入<STDIN>在结束输入时插入一换行符，而chop函数可用来移除并返回字串的最后一个字符

- chop函数的参数可为列表变量．如果参数缺省，函数默认地应用于特殊变量$_

语法：

```
$lastchar = chop ($var);
```

例如：

```
$url = “perl.linux.org”;
chop ($url);
```

也可写成：

```
chop ($url = “perl.linux.org”);
```

– **chomp**函数

- 检查字符串或字符串列表中元素的最后一个字符是否为由系统变量$/定义的换行符，如果是就将其删除，返回值为实际删除的字符个数．

– **int**函数

将浮点数舍去小数部分转化为整型数

– defined函数

判断变量、数组或数组的一个元素等是否已经被赋值，如果已定义返回真，否则返回假。例如：

```
#!/usr/bin/perl

$array[0] = “happy”;
print “@array\n”;
$array[3] = “birthday”;
print “@array\n\n”;

for ($i = 0; $i <4; ++$i) {
  if (!defined($array[$i])){
    $array[$i] = “happy”;
  }
}
print “@array\n”;
```

结果为：

```
happy
happy    birthday

happy happy happy birthday
```

4. 数组和列表函数

- **push函数**

在数组末尾增加一个或多个元素,返回值为结果(列表)的长度

语法：

```
push(@array, elements);
```


可以是一个元素也可以为一个列表

与以下语句结果相同:
$array[2] = “three”;

例如：

```
@array = (“one”, “two”);
push(@array, “three”);
```

结果为:

```
@array = (“one”, “two”, “three”);
```

– pop函数

与push作用相反,删去列表最后一个元素,并将其作为返回值,当列表已空,则返回”未定义值”,即空串。

例如：

```
@array = (“one”, “two”, “three”);
$rm = pop(@array);
```

结果为:

```
$rm = “three”;
@array = (“one”, “two”);
```

– shift函数

删去数组第一个元素,剩下元素前移,返回被删去的元素

例如:

```
@array = (“one”, “two”, “three”);
$rm = shift(@array);
```

结果为:

```
$rm = “one”;
@array = (“two”, “three”);
```

– unshift函数

作用与shift相反,在数组之前插入一个或多个新元素,
返回值：结果(列表)的长度

语法:

```
unshift(@array, elements);
```


可以是一个元素也可以为一个列表

编程提示

如数组较大,由于shift和unshift需要对数据进行批量移动,所以尽可能避免使用shift和unshift

例如:

```
@array = (“one”, “two”);
$rm = unshift(@array, “three”);
```

结果为:

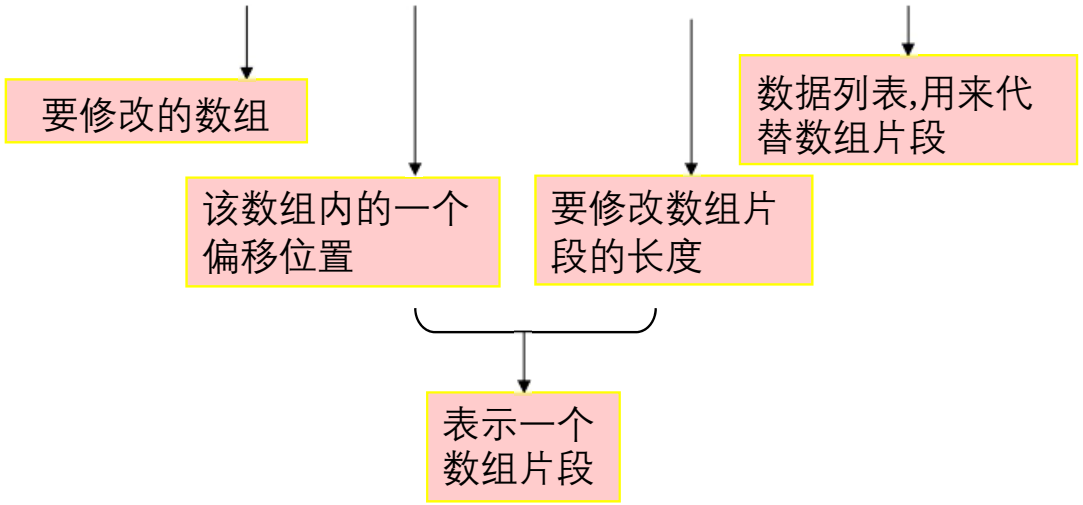
```
$rm = 3;
@array = (“one”, “two”, “three”);
```

– splice函数

插入元素、删除或替换一个数组的各个片段

语法:

splice(@array, skipelements, length, @newlist)



以上后三个参数可省略

- splice(@array, skipelements, length)
  表示指定的片段直接从数组中删除
- splice(@array, skipelements)
  表示从指定偏移位置开始一直删到数组末尾
- splice(@array)
  表示会删掉所有数组元素

注:

如果length为0,则相当于向数组列表中插入元素

如果skipelements为-1,lenght为0,则在数组列表末尾插入元素

– sort函数

按照字典顺序(即ASCII顺序)对列表的一个拷贝排序

语法:

@sorted = sort(@list);

例:

```
@array = (“d”, “b”, “c”, “a”);
@sortedarray = sort(@array);
```

结果:

```
@sortedarray = (“a”, “b”, “c”, “d”);
```

例:

```
@array =(2, 6, 9, 1);
@sortedarray = sort(@array);
```

结果:

```
@sortedarray = (1, 2, 6, 9);
```

例:

```
@array =(2, 6, 9, 15);
@sortedarray = sort(@array);
```

结果:

```
@sortedarray = (15, 2, 6, 9);
```

- 以上ASCII顺序排列适用于字符串,不适用于数字排序
- 对数字进行排序时,需要按照数字顺序排列,方法:采用”<=>”符号

  语法:

  @sorted = sort{$a<=>$b}(@list); 由小到大排列

  @sorted = sort{$b<=>$a}(@list); 由大到小排列

- 对字符串按照ASCII顺序排列也可写成如下:

  @sorted = sort{$a cmp $b}(@list); 由小到大排列

  完全等价于:

  @sorted = sort(@list);

  @sorted = sort{$b cmp $a}(@list); 由大到小排列

编程提示

用于数字顺序排列时的两个变量名必须为$a和$b,否则sort函数无法对列表中的值进行排序,而且不会因此报告任何错误

在程序中,尽量避免使用$a和$b这两个变量,属于函数的保留变量

– reverse函数

将列表的拷贝反转顺序

语法:

@reversed = reverse (@list);

例如:

```
@array = (“d”, “b”, “c”, “a”);
@reversedarray = reverse (@array);
```

结果:

```
@reversedarray = (“a”, “c”, “b”, “d”);
```

5. 关联数组函数

exists($array{key})

判断关联数组中是否存在元素,若存在返回为真(1),否则返回假

举例:

```
%fred = (“one”, “cat”, “two”, “horse”);
$var1 = $fred{“one”};
$var2 = $fred{“two”};
@array1 = keys(%fred);
@array2 = values(%fred);
$var3 = exists($fred{“one”});
while (($key, $value) = each(%fred)) {
  print "$key = $value\n";
}
```

6. 错误控制函数

- die函数

终止程序执行,并输出一条出错信息

语法:

die “message”;

用法:

```
unless (关键条件) {
   die “错误:关键条件没有满足\n”;
}
```

或者

```
关键条件 or die “错误:关键条件没有满足\n”;
```

– **warn**函数

产生相同的输出,只是程序不会终止执行

例：如果输入的分母为０，则程序退出

```
#!/usr/bin/perl
#using function ‘die’ to terminate a program

print “please enter a numerator:”;
chomp ($numerator = <STDIN>);

print “please enter a denominator:”;
chomp ($denominator = <STDIN>);

$denominator != 0 or die “Cannot divide by zero”;
print “\nThe result is “, $numerator / $denominator, “\n”;
```

结果：

```
Please enter a numerator: 7
Please enter a denominator: 3

The result is 2.3333333333333333
```

程序举例

- 产生相同随机数序列

```
#! /usr/bin/perl
#Seeding the random number generator.
#during each iteration, set seed to 1, then produce three random integers
for (1..3) {
  print “\n\nSetting seed to 1\n”;
  srand(1);

  #produces same three values each time
   for (1..3) {
      print “ ”, 1 + int(rand(6));
    }
}
```

```
Setting seed to 1
 1 3 6
Setting seed to 1
 1 3 6
Setting seed to 1
 1 3 6
```

程序举例

- 输出时规定字段宽度和精度

```
#! /usr/bin/perl
#Printing integers right – justfied
printf “%4d\n”, 1;
printf “%4d\n”, 12;
printf “%4d\n”, 123;
printf “%4d\n”, 1234;
printf “%4d\n”, 12345;
printf “%4d\n”, 123456789;
printf “%4d\n”, -1;
printf “%4d\n”, -12;
printf “%4d\n”, -123;
printf “%4d\n”, -1234;
printf “%4d\n”, -12345;
printf “%4d\n”, -123456789;
```

```
1
12
123
1234
12345
123456789
-1
-12
-123
-1234
-12345
-123456789
```

%d :显示一个有正负号的二进制整数，
4 : 最小宽度为4字节，左对齐，右补空格