第二章 Perl语法基础

主要内容

2.1 变量

- 标量
- 数组
- 关联数组

2.2 操作符

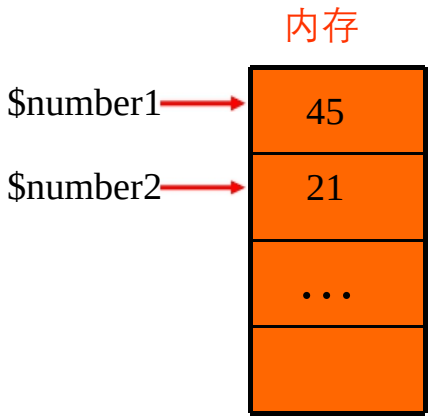
- 算术操作符
- 关系运算符
- 比较操作符
- 逻辑操作符
- 位操作符
- 赋值操作符
- 自增自减操作符
- 字符串连接和重复操作符
- 逗号操作符
- 条件操作符

2.3 控制结构

- 选择结构
- 循环结构

2.1 变量

变量概念:是指计算机内存的一个特定位置



变量名命名规则：由字母、数字和下划线组成，变量名必须以字母开头，大小写区别

例如：

```
#!/usr/bin/perl
#Give value to variable and print it.
$number1 = 45;
$number2 = 21;
print “$number1, $number2\n”;
```

2.1 变量

2.1.1 标量

标量变量的类型标识符为：**$**

- 数字变量
  - 整型变量 ⟹ 十进制（默认）
八进制（以0开头）
十六进制（以0x开头）
二进制(0b)
  - 浮点变量
    - 有小数点的数字或科学计算法的数字

例如：

```
$number1 = 45;
$number2 = 4.56;
$number3 = 9.0e+5;
```

2.1 变量

– 字符串变量

- 单引号字符串

  单引号字符串中插入单引号,需要用\',插入反斜杠,需要用\\

- 双引号字符串

单引号和双引号的区别:

(1)单引号没有变量替换功能

(2)单引号可以跨多行

(3)单引号的反斜杠不支持转义字符,而只在包含单引号和反斜杠时起作用

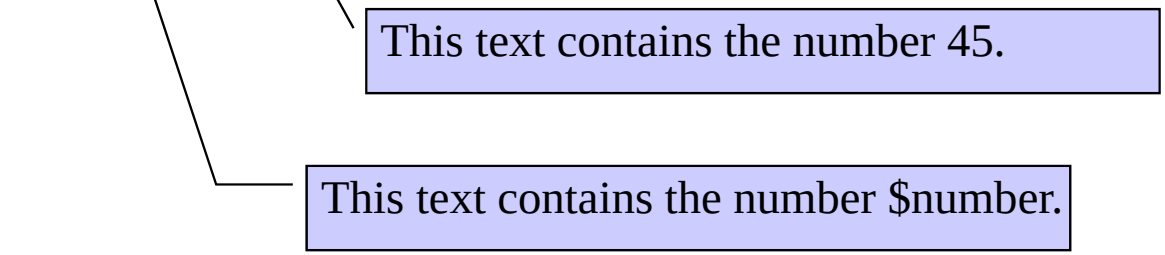
编程提示

字符串长度不受限制,最短的可以没有字符,最长的可以把内存添满

例如:

```
$number = 45;
$text1 = “This text contains the number $number.”;
$text2 = ‘This text contains the number $number.’;

print “$text1\n”;
print “$text2\n”;
```



例如:

```
$text = ‘This is two
lines of text
’;
```

与以下语句等同

```
$text = “This is two \nlines of text\n”;
```

举例3:

```
$number = 45;

$text2 = ‘This text contains the number $number.’;

print “\’$text2”;
print “$text2\\”;
```

‘This text contains the number $number.

This text contains the number $number.\

2.1 变量

双引号字符串中反斜杠转义表

双引字符串	结构含义	双引字符串	结构含义
\n	换行	\x20	任意十六进制ASCII值(20为空格的十六进制ASCII表示)
\r	回车	\cC	任意"控制"字符
\t	水平置表符	\\	反斜杠
\f	换页符	\"	双引号
\b	退格	\l	下一字母小写
\v	垂直置表符	\L	以后所有字母小写直到\E
\a	响铃	\u	下一字母大写
\e	Esc	\U	以后所有字母大写直到\E
\038	任意八进制ASCII值(38为空格的八进制ASCII表示)	\E	结束\L和\U

例如：

```
$a = “T\LHIS IS A \ESTRING”;
print “$a”;
```

结果为:

```
This is a STRING
```

例：

```
$a = 14;
print “The value of \$a is $a.\n”;
```

结果为:

```
The value of $a is 14
```

反斜杠可取消变量替换

2.1 变量

- 字符串和数字的相互转换
  - 若字符串变量和数字变量计算,则从字符串变量中取从左起的第一个数字至第一个非数字字符,再与数字变量进行运算

例:

```
$result = “12a34” + 1;
print “$result”;
```

结果为:

```
13
```

2. 1 变量

2.1.2 数组

数组的类型标识符为：@

1.赋值方法:

数组 = 列表

列表是指由一系列值或表达式构成的集合(, , ,)

```
@text1 = (“dog”, ”cat”, ”horse”);
@text2 = (‘hi’, 1, 23, ’c’);
@text3 = (1, 4+6, ’c’);
```

编程提示

数组长度不受限制,最短的可以不含元素,最大的数组可以占满全部可用内存

2.1 变量

几点说明:

如果把数组变量赋给标量变量,则赋给标量变量的就是数组长度

例:

```
@perlbaby = (1, 2, 3);
$a = @perlbaby;
```

$a结果为:

```
3
```

2.1 变量

2.元素引用:

如想引用数组**@a**的第**i**个元素，则表示为：

$a[i-1]

如数组变量@c，数组中的各个元素为：

$c[0], $c[1], $c[2],…

位置编号,或下标

下标可以为整数,也可以是一个整数表达式

```
$number1 = 5;
$number2 = 6;
$a[$number1 + $number2] = $a[11];
```

2.1 变量

编程提示:

当数组下标为非整数时,执行时不会当作语法错误,只是逻辑错误发生了

注意”第7个数组元素”和”数组元素7是不同的”,第7个数组元素的下标是6,而数组元素7的下标是7

Perl可根据不同的类型标识符来区分一个变量是标量还是数组,因此不同的类型可采用相同的名字.在此提示,为了提高程序的可读性,及易于调试和维护,尽量避免标量和数组同名.

2.1 变量

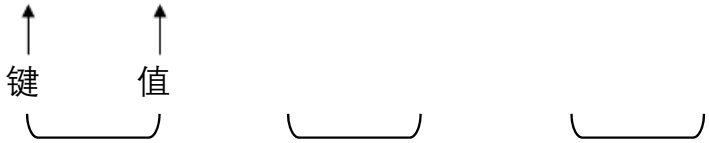
2.1.3 关联数组(Associative Array)

也叫散列或哈希（Hash）

关联数组的类型标识符为：**%**

1.关联数组格式：

%ARRAY = (key1, value1, key2, value2, key3, value3,…);



2.1 变量

2.关联数组值引用：

$关联数组变量名{键}

如想引用关联数组**%a**的键名为one和two的元素值，则表示为：

```
%fred = (“one”, “cat”, “two”, “horse”);
$a = $fred{“one”};
$b = $fred{“two”};
```

```
print “$a\n”;
print “$b\n”;
```

2.1 变量

3.关联数组操作：

keys(%array)

返回由关联数组%array中的所有当前关键字组成的列表

values(%array)

返回由关联数组%array中所有当前值组成的列表

each(%array)

返回由一个关键字和一个值构成的两个元素的表.对同一数组再造作时返回下一对值直至结 束.若没有更多的值时,each()返回空表.这通常在打印全部列表时很有用.

delete($array{key})

删除关联数组中的一个key以及这个key所对应的value

例如：

```
%fred = (‘one’, ‘cat’, ‘two’, ‘horse’);
$a = $fred{‘one’};
$b = $fred{‘two’};
@c = keys(%fred);
@d = values(%fred);
while (($key, $value) = each(%fred)) {
   print "$key = $value\n";
}
```

one=cat
Two=horse

2.2 操作符

- 算术操作符
- 关系运算符
- 比较操作符
- 逻辑操作符
- 位操作符
- 赋值操作符
- 自增自减操作符
- 字符串连接和重复操作符
- 逗号操作符
- 条件操作符

2.2 操作符

1、算术操作符

运算符号	实例	解释
+	$z = $x + $y;	将$x和$y相加之后，再将结果赋给$z
-	$z = $x - $y;	将$x减去$y之后，再将结果赋给$z
*	$z = $x * $y;	将$x和$y相乘之后，再将结果赋给$z
/	$z = $x / $y;	将$x除以$y之后，再将商赋给$z
%	$z = $x % $y;	将$x除以$y之后，再将余数赋给$z
**	$z = $x ** $y;	将$x乘以$y次后，再将结果赋给$z
++	$x++; ++$x;	将$x加1后，再将结果赋给$x
--	$x--; --$x;	将$x减1后，再将结果赋给$x
.	$z = $x . $y;	将$x字符串和$y字符串连接之后，再将结果赋给$z

例如:

```
$s1 = “Hello”;
$s2 = “Everyone”;
$s3 = $s1 . $s2;
```

则$s3结果为:

```
HelloEveryone
```

2.2 操作符

2、关系运算符

运算符号	实例	解释
<	$x < $y	如果$x小于$y，返回1，否则返回0
>	$x > $y	如果$x大于$y，返回1，否则返回0
=	$x = $y	如果$x等于$y，返回1，否则返回0
<=	$x <= $y	如果$x小于等于$y，返回1，否则返回0
>=	$x >= $y	如果$x大于等于$y，返回1，否则返回0
!=	$x != $y	如果$x不等于$y，返回1，否则返回0
<=>	$x <=> $y	如果$x大于$y，返回1，如果$x等于$y，返回0，如果$x小于$y，返回-1

2.2 操作符

3、比较操作符

运算符号	实例	解释
gt	$str1 gt $str2	如果$str1大于$str2，返回1，否则返回0
ge	$str1 ge $str2	如果$str1大于等于$str2，返回1，否则返回0
lt	$str1 lt $str2	如果$str1小于$str2，返回1，否则返回0
le	$str1 le $str2	如果$str1小于等于$str2，返回1，否则返回0
eq	$str1 eq $str2	如果$str1等于$str2，返回1，否则返回0
ne	$str1 ne $str2	如果$str1不等于$str2，返回1，否则返回0
cmp	$str1 cmp $str2	如果$str1大于$str2，返回1，如果$str1等于$str2，否则返回0，如果$str1小于$str2，返回-1

2.2 操作符

4、逻辑操作符

逻辑操作符	解释
||或or	逻辑或
&&或and	逻辑与
!或not	逻辑非
xor	逻辑异或

2.2 操作符

异或操作真值表

$x	$y	结果
true	true	false
true	false	ture
false	true	true
false	false	false

异或实际上就是判断两个输入**逻辑**值是否不同，如果不同则结果为1，相同则为0。

2.2 操作符

5、位操作符

操作符号	表示
&	位与
|	位或
~	位非
^	位异或
$x<<1	左移
$x>>2	右移

位运算是将运算符两边的数字换算成二进制（例：0000010001）后比较相同位置上的0、1进行运算的；
逻辑运算即比较运算符两边的逻辑值（true或false）；

2.2 操作符

6、赋值操作符

运算符号	实例	解释
=	$x = $y ;	将$x的值赋给$y
+=	$x += $y;	将$x和$y相加之后，再将结果赋给$x
-=	$x -= $y;	将$x减去$y之后，再将结果赋给$x
*=	$x *= $y;	将$x和$y相乘之后，再将结果赋给$x
/=	$x /= $y;	将$x除以$y之后，再将商赋给$x
**=	$x **= $y;	将$x乘以$y次后，再将结果赋给$x
%=	$x %= $y;	将$x除以$y之后，再将余数赋给$x
.=	$str1 .= $str2;	将字符串$str1加上$str2之后,再将结果赋给$str1这个字符串
x=	$str x= $y;	重复$str字符串$y次,并将结果赋给$str这个字符串

2.2 操作符

赋值操作符”=”在此需要注意几点:

“=”可在一个赋值语句中出现多次

```
$value1 = $value2 = “a string”;
```

```
($a = $b) += 3;
```

“=”作为子表达式

等价于:

```
$a = $b;
$a += 3;
```

2.2 操作符

7、自增自减操作符

- 自增运算符:++
  - 预自增
  - 后自增

```
++$a;
$a++;
```

- 自减运算符号:--
  - 预自减
  - 后自减

```
--$a;
$a--;
```

2.2 操作符

自增自减操作符注意以下几点:

1. 不要在变量两边都使用此种操作符

```
++$value1--;
```

2. 不要在变量自增/自减后在同一表达式中再次使用

```
$var2 = $var1 + ++$var1;
```

3. ++可用于字符串,但当结尾字符为”z””Z””9”时进位

例如:

```
$string1 = “abc”;
$string1++;
$string2 = “abz”;
$string2++;
$string3 = “bc999”;
$string3++;
```

$string1, $string2, $string3的结果分别为:abd, aca和bd000

2.2 操作符

4.如果字符串中含有非字母且非数字的字符,或数字位于字母中,则经过++运算前值转换为数字零,因此结果为1

```
$string1 = “ab*z”;
$string1++;
$string2 = “ab5z”;
$string2++;
```

$string1, $string2的结果均为:ab1a

2.2 操作符

8、字符串连接和重复操作符

– 连接符:.
– 重复符:x
– 连接且赋值:.=

```
$new1 = “potato” . “head”;
$new2 = “t” x 5;
$a = “be”;
$a .= “witched”;
```

则变量$new1,$new2和$a的值分别为:

```
potatohead
ttttt
bewitched
```

举例

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

#Store two DNA fragments into two variables called $DNA1 and $DNA2
$DNA1 = 'ACGGGAGGACGGATTAAGTTATGCCCCATGACCC';
$DNA2 = 'ATAGTCCTGATTCGGATTTA';

#Print the DNA onto the screen
print "Here are the original two DNA fragments:\n\n";
print $DNA1, "\n";
print $DNA2, "\n\n";

# Concatenate the DNA fragments into a third variable and print them
$DNA3 = $DNA1 . $DNA2;
print "Here is the concatenation of the first two fragments (version 2):\n\n";

print "$DNA3\n\n";
```

2.2 操作符

9、逗号操作符

作用是使前面的表达式先进行运算

```
$var1 += 1, $var2 = $var1;
```

等价于:

```
$var1 += 1;
$var2 = $var1;
```

使用此操作符的唯一理由是提高程序的可读性,将相关密切的两个表达式结合在一起

2.2 操作符

10、条件操作符

用法:

条件?值1:值2

当条件为真时取值1,为假时取值2

举例:

```
$result = $var == 0 ? 14 : 7;
```

```
$var == 43 ? $var1 : $var2 = 14;
```

2.2 操作符

11.区块操作符

..

这个操作符是Perl语言中特有的操作符,非常实用

举例:

```
@digit1 = (1..9);
@char = (‘A’..’E’);
@total = (1..3,’A’..’C’);
```

则数组变量分别为:

```
@digit1 = (1,2,3,4,5,6,7,8,9);
@char = (‘A’,’B’,’C’,’D’,’E’);
@total = (1,2,3,’A’,’B’,’C’);
```

操作符优先级

高

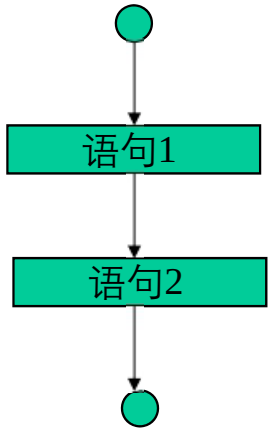
低

操作符	描述
++, --	自增、自减
-,~,!	单目
**	乘方
=~,!~	模式匹配
*,/,%,x	乘、除、取余、重复
+ , -, .	加、减、连接
<<,>>	移位
<,<=,>,>=,it,le,gt,ge	不等比较
==,!=,<=>,eq,ne,cmp	相等比较
&	位与
|,^	位或，位异或
&&	逻辑与
||	逻辑或
..	列表范围
?and:	条件操作符
=,+=,-=,*=	赋值
,	逗号操作符

单目运算符(unary operator)指运算所需变量为一个的运算符：
逻辑非运算符 !、按位取反运算符 ~

2.3 控制结构

1. 顺序结构



2.3 控制结构

2. 选择结构

- if判断
- if/else选择
- if/elsif/else选择
- unless判断
- unless/else选择

3. 循环控制

- while循环
- until循环
- do/while循环
- do/until循环
- for循环
- foreach循环

2.3 控制结构

选择结构

if选择

语法：

```
if (判别式) {
    程序叙述语句块；
}
```

如果只有一条语句,也可写成:

```
程序叙述语句 if 判别式;
```

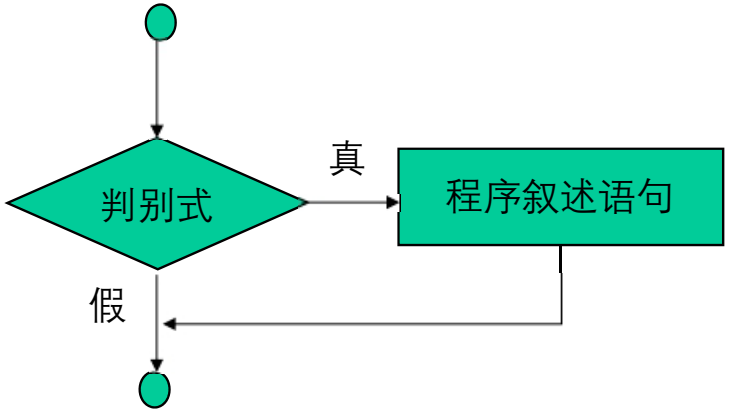
良好编程习惯

在程序中合理进行缩进,可显著增强程序可读性,将每个缩进单位设为3个空格字符或Tab位

为防丢失花括号,首先输入开始/结束两个花括号,再将语句输入它们之间

2.3 控制结构

if选择结构流程图



2.3 控制结构

举例：

```
#!/usr/bin/perl

print “请输入您的分数?\n”;
$score = <STDIN>;
chop($score);

if ($score >= 60) {
   print “您的分数及格了!\n”;
}
```

该语句也可写成:

Print “您的分数及格了!\n” if ($score >= 60);

2.3 控制结构

if/else选择

语法：

```
if (条件判别式) {
    程序叙述语句块1；
}
else {
    程序叙述语句块2;
}
```

条件运算符优先级极低，一定要为条件加括号

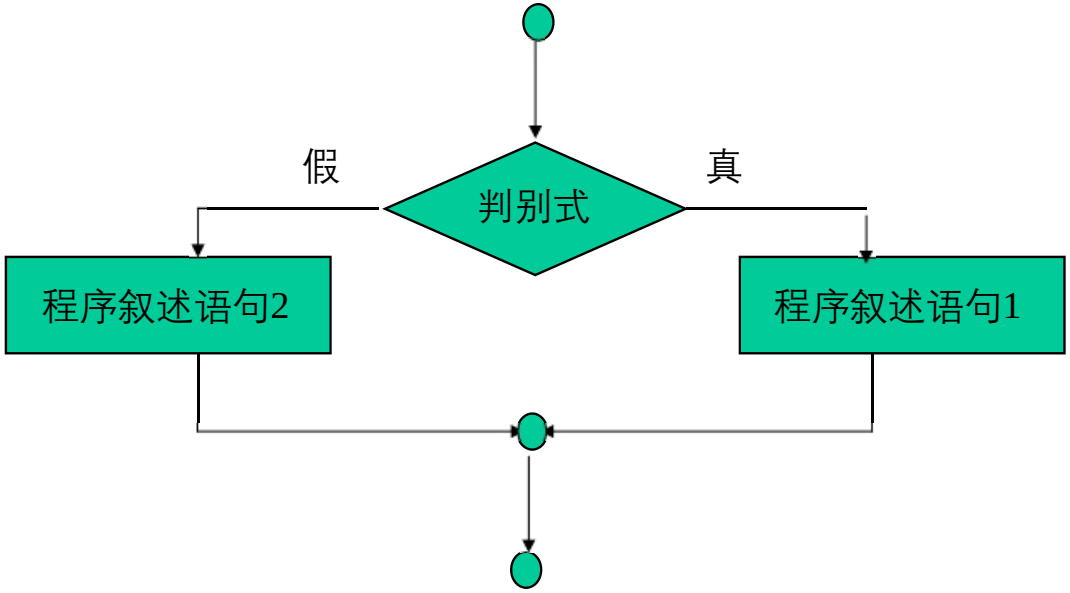
也可用条件运算符**?:**表达:

```
(条件判别式) ? 程序叙述语句1 : 程序叙述语句2;
```

条件为真时　条件为假时

2.3 控制结构

if/else选择结构流程图



2.3 控制结构

举例:

```
#!/usr/bin/perl

print “请输入您的分数?\n”;
$score = <STDIN>;
chop($score);

if ($score >= 60) {
   print “您的分数及格了!\n”;
}
else {
   print “您的分数不及格!\n”;
}
```

2.3 控制结构

if/elsif/else选择

语法：

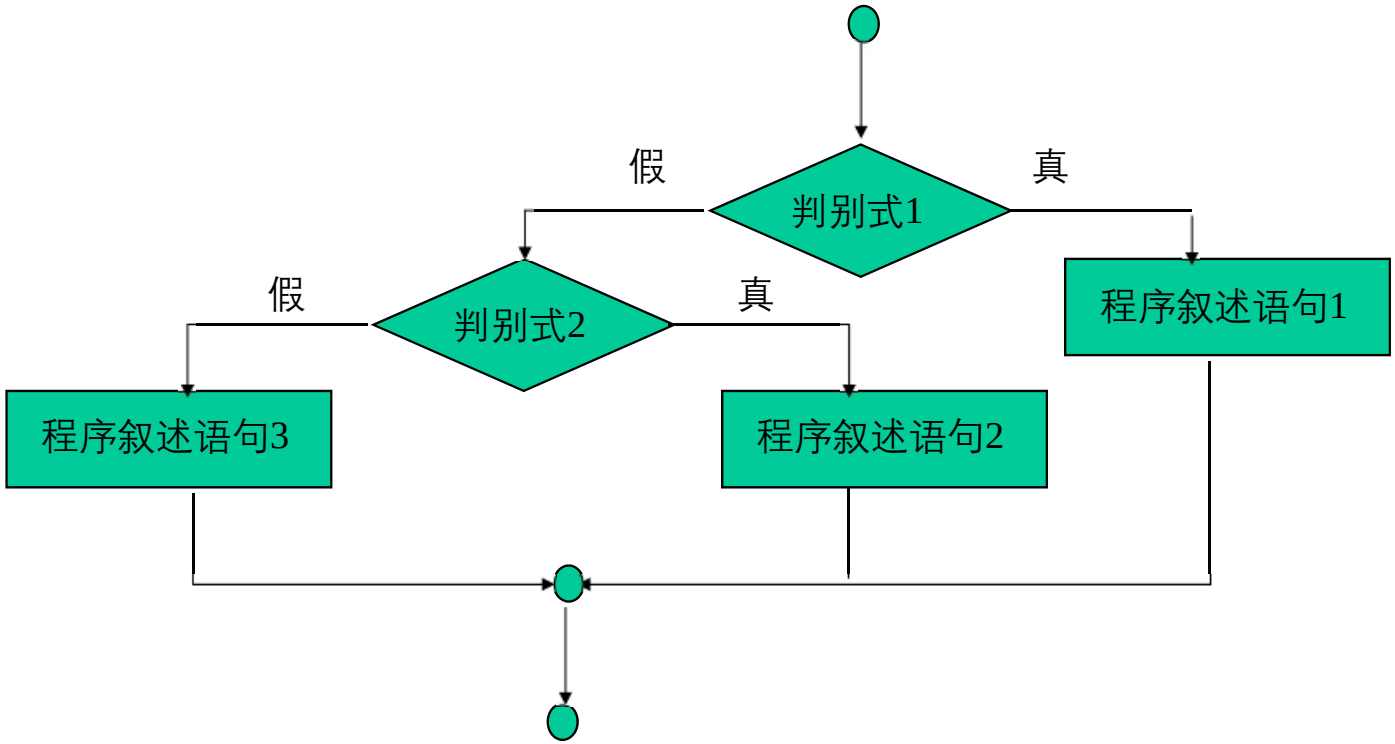
```
if (条件判别式1) {
    程序叙述语句1；
}
elsif (条件判别式2) {
    程序叙述语句2；
}
else {
    程序叙述语句3；
}
```

编程提示

尽量尽快检测条件为”真”的条件,这样可尽早退出控制结构,显著提高程序运行速度

2.3 控制结构

if/elsif/else选择结构流程图



2.3 控制结构

举例:

```
#!/usr/bin/perl

print “请输入您的分数?\n”;
$score = <STDIN>;
chop($score);
if ($score > 60) {
   print “您的分数大于!\n”;
} elsif ($score < 60) {
   print “您的分数小于60分!\n”;
} else {
  print “您的分数刚好是60分!\n”;
}
```

2.3 控制结构

unless判断

语法：

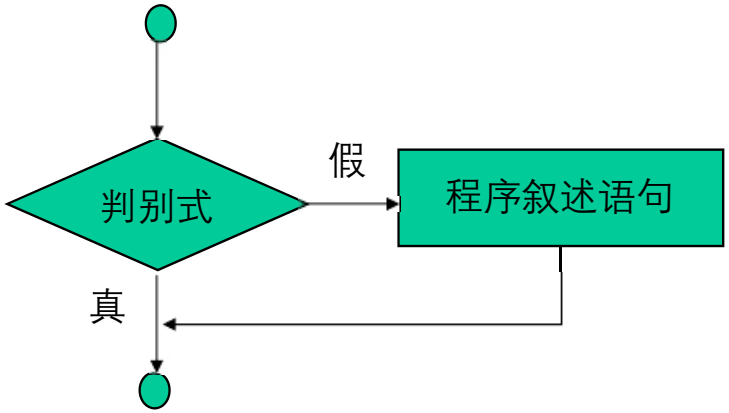
```
unless (判别表达式) {
    判别式为假时语句块；
}
```

也可写成：

```
判别式为假时语句块 unless (判别表达式)；
```

2.3 控制结构

unless判断结构流程图



2.3 控制结构

举例:

```
#!/usr/bin/perl

print “请输入您的分数?\n”;
$score = <STDIN>;
chop($score);

unless ($score < 60) {
   print “您的分数及格了!\n”;
}
```

2.3 控制结构

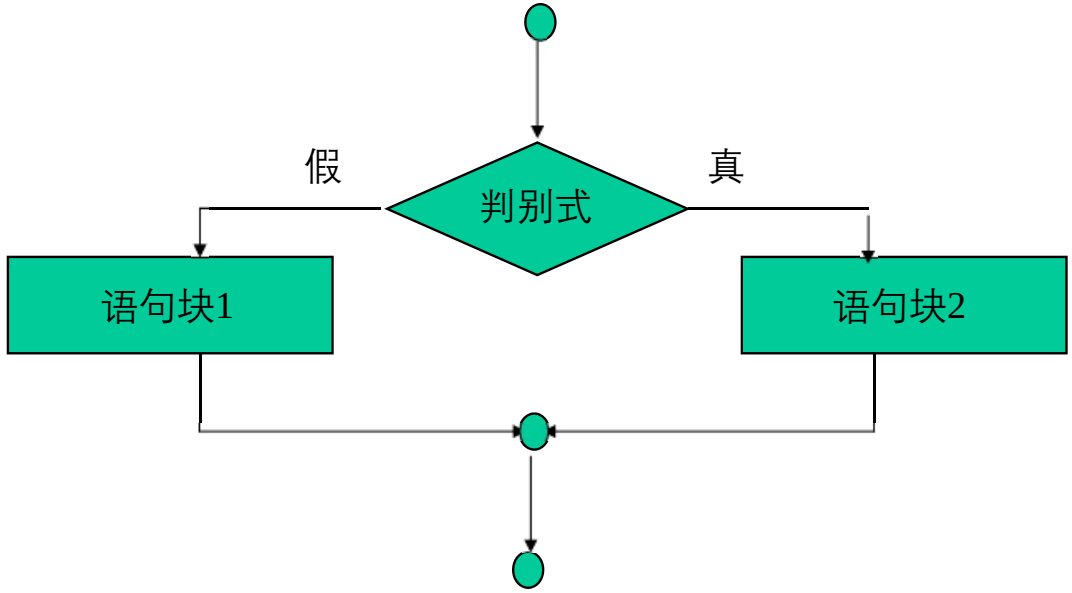
unless/else选择

语法:

```
unless (判别表达式) {
    判别式为假时语句块1；
} else {
判别式为真时语句块2；
}
```

2.3 控制结构

unless/else选择结构流程图



2.3 控制结构

举例:

```
#!/usr/bin/perl

print “请输入您的分数?\n”;
$score = <STDIN>;
chop($score);

unless ($score < 60) {
  print “您的分数及格了!\n”;
} else {
  print “您的分数不及格!\n”;
}
```

2.3 控制结构

循环控制

while循环

语法：

```
while (判别运算式) {
   程序叙述区块；
}
```

也可写成：

```
程序叙述语句 while (判别运算式)；
```

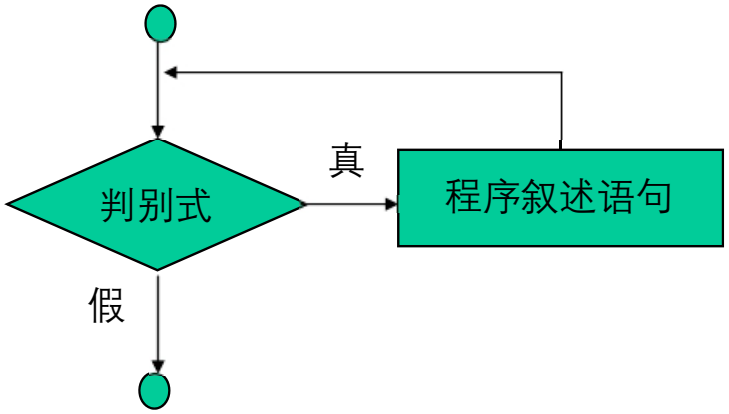
常见编程错误

将while写成了While,导致语法错误,因为Perl严格区分大小写

避免无限循环错误,即条件不能永远为真

2.3 控制结构

while循环结构流程图



2.3 控制结构

举例:

```
while ($i <= 10) {
   $sum += $i;
   $i++;
}

print “$sum\n”;
```

2.3 控制结构

until循环

语法：

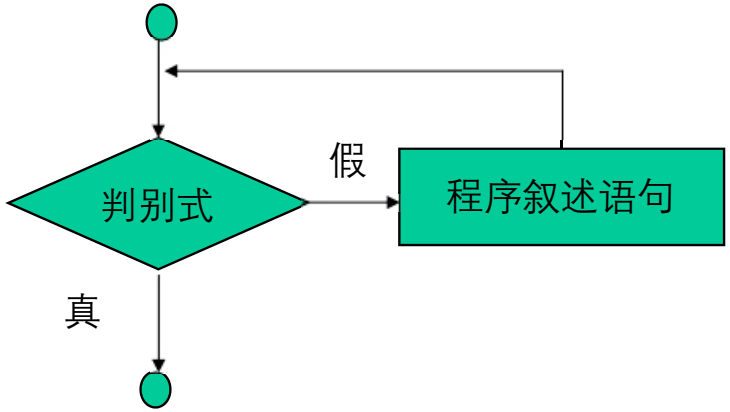
```
until (判别运算式) {
   程序叙述区块；
}
```

也可写成：

```
程序叙述区块 until (判别运算式)；
```

2.3 控制结构

until循环结构流程图



2.3 控制结构

举例:

```
until ($i > 10) {
    $sum += $i;
    $i++;
}

print “$sum\n”;
```

2.3 控制结构

do while循环

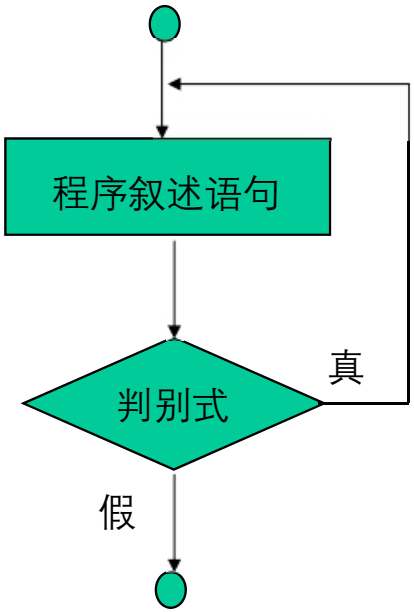
语法：

```
do {
   程序叙述区块；
} while (判别运算式)
```

注意：与while语句不同的是程序叙述区块至少被执行一次

2.3 控制结构

do while循环结构流程图



2.3 控制结构

举例:

```
do {
   $sum += $i;
   $i++;
} while ($i <= 10)

print “$sum\n”;
```

2.3 控制结构

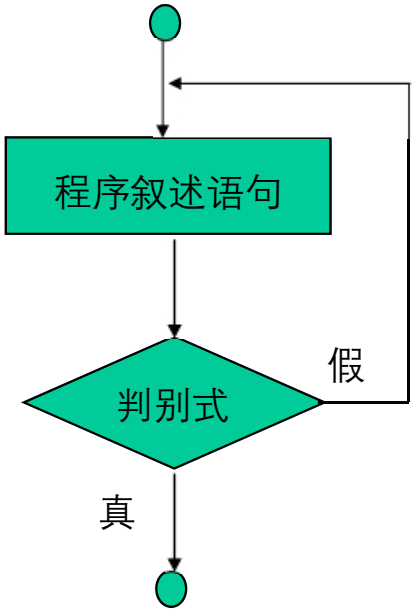
do until循环

语法：

```
do {
   程序叙述区块；
} until (判别运算式)
```

2.3 控制结构

do until循环结构流程图



2.3 控制结构

举例:

```
do {
    $sum += $i;
    $i++;
} until ($i > 10)

print “$sum\n”;
```

2.3 控制结构

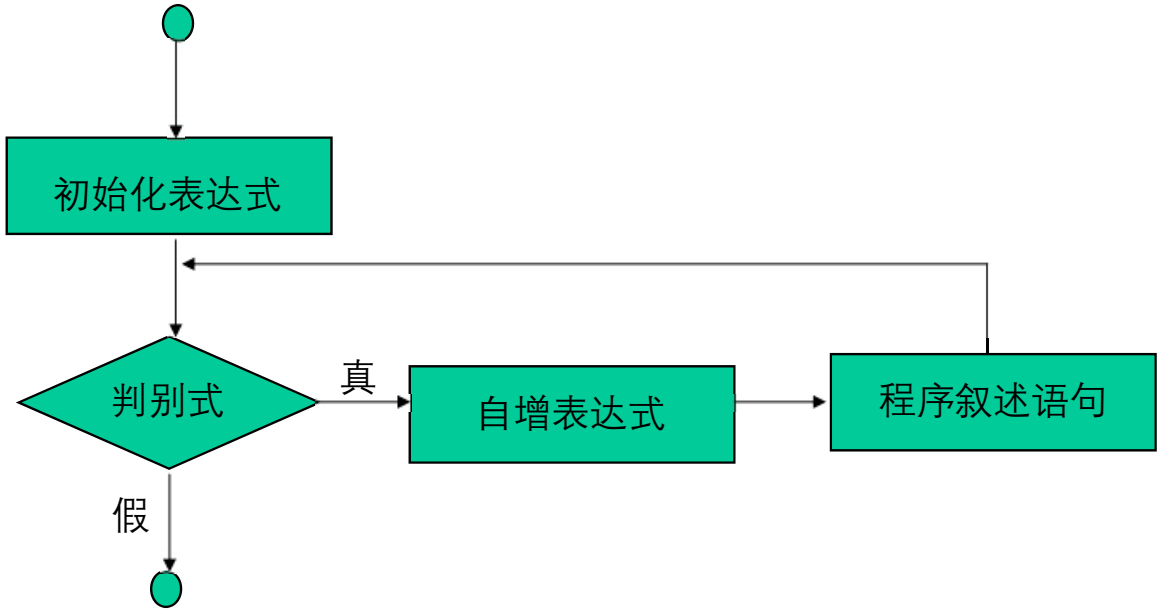
for循环

语法：

```
for (初始化表达式；循环继续条件；自增表达式) {
程序叙述区块；
}
```

2.3 控制结构

for循环结构流程图



2.3 控制结构

举例:

```
$sum = 0;
for ($i =1; $i <= 10; $i++) {
   $sum += $i;
}
print “$sum\n”;
```

```
@array = (3,6,9);
$number = @array;
for ($i =1; $i <= $number; $i++) {
   $sum += $array[$i];
}

print “$sum\n”;
```

2.3 控制结构

foreach循环

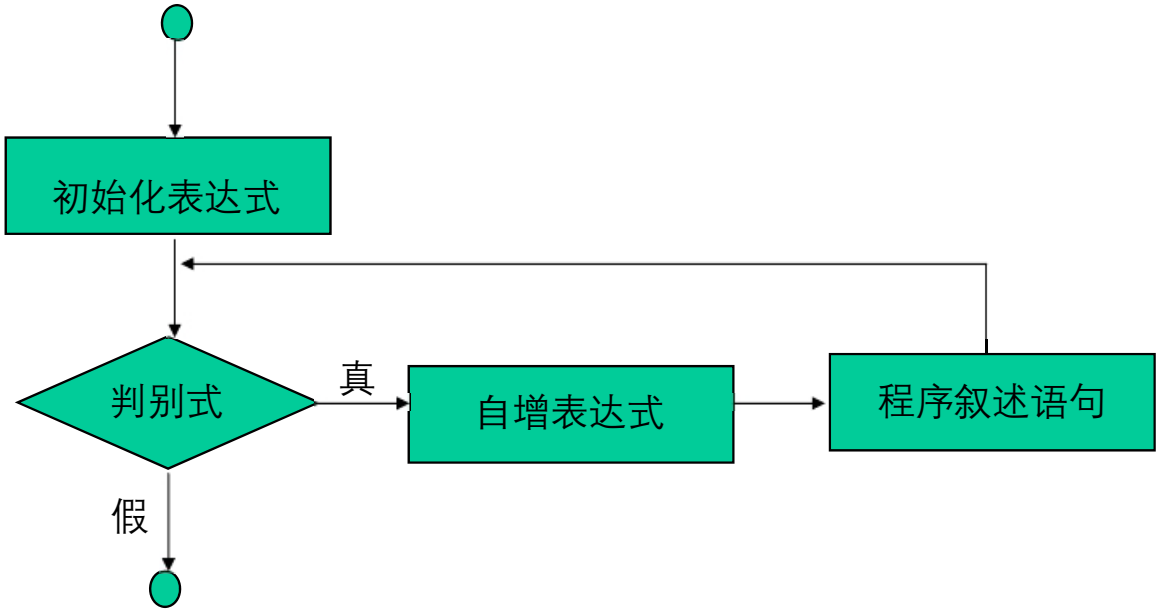
利用foreach循环,可在一系列值中进行循环,访问并处理其中的每个元素

语法：

```
foreach 控制变量 (列表)
{
程序叙述区块；
}
```

2.3 控制结构

foreach循环结构流程图



2.3 控制结构

举例:

```
@array = (3,6,9);
$number = @array;

foreach $int(@array) {
   $sum += $int;
}

print “$sum\n”;
```

2.3 控制结构

循环控制语句

next

在while和until结构中,执行next了语句后,会紧接着检测循环是否应继续下去的条件;而在for结构中,首先会执行自增表达式,再对循环继续条件进行检测;在foreach结构中,控制变量会被设成列表中的下一个元素.

2.3 控制结构

last

在while,until,for和foreach结构中执行last语句,便会造成立即退出当前结构

redo

在while,until,for和foreach结构中使用redo语句,会马上返回循环主体的第一条语句,而且不会对循环继续条件进行检测.

编程提示

redo命令在do循环中不起作用

2.3 控制结构

举例:

```
for ($i =1; $i <=10; $i++) {
   next if ($i % 2);
   print “$i是一个奇数\n”;
}
```

```
for ($i =1; $i <=10; $i++) {
   last if ($i == 5);
   print “$i\n”;
}
```