# 标量数据

Perl也有数据类型——标量。这里的标量的概念与数学中的标量和向量没有多少关系。

大多数标量是数字或者字符串。Perl几乎以相同的观点看待数字和字符串。如果使用过别的编程语言，你可能把hello看作5个字符的组合，而不是一个单独的东西。但在Perl中，一个字符串是一个标量数据，当然这个字符串也有值，后面将详细介绍。

标量数据操作符（如相加和串联），通常会产生一个新的标量数据。标量数据的值可以存放在标量变量中。标量可以从文件或设备读取，也可以写进去。

## 数字

虽然标量在大多数情况下不是数字就是字符串，现在我们最好还是将它们分开来看待。我们首先讨论数字，再讨论字符串。

### 所有数字内部的格式一致

整数（如255,2001）和浮点数（如3.14159，1.35e25），在内部，Perl都把它们当作双精度浮点数来处理。也就是说Perl内部没有整数值。

## 字符串

字符串是一串字符（如hello）。

字符串可能是字符的任意组合，需要注意的是Perl中的NUL字符没有特殊的含义，并不代表字符串结束。另外Perl能计算字符的长度，不用靠null来判断字符串是否结束。

最短的字符串不包含任何字符。最长的字符串，可以填满整个内存。

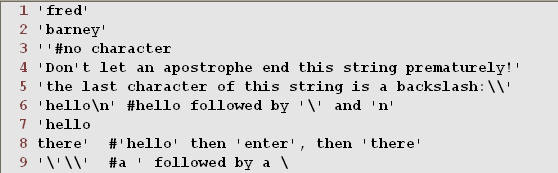
通常字符串是可打印字符、数字和标点符号的序列。但是，Perl中字符串可以包含任意字符，这意味着利用字符串（string），你可以创建、遍历、操作二进制数据，而利用别的方法可能遇到极大的困难。例如，你可以把要更新的图片或编译好的程序放入一个Perl的字符串变量中，做完相应的修改后，再写回去。

和数字一样，字符串也可以由文字符号（literal）来表示，它用来在Perl程序中代表某个字符串。**有两种类型的字符串：单引号字符串和双引号字符串**。

### 单引号字符串

单引号字符串是由单引号括起来的字符序列。单引号不是字符串的一部分，但Perl可以利用它来辨别字符串的开始和结束。

除了单引号或者反斜线（包括换行字符，如果字符在下一行继续）之外的任何字符都表示它自身。**要得到一个反斜线可以把两个反斜线放在一起；要得到单引号，需要在单引号前加上反斜线。**



单引号字符串中的\n不会被当作换行符来处理，而是两个字符\和n。只有当反斜线\后面接的是反斜线\或单引号`时，才会被当作特殊符号处理。

### 双引号字符串

双引号字符串是用双引号括起来的字符串。不同的是，反斜线\可以用来表示控制字符、八进制、十六进制数的表示。

# 列表和数组

如果把标量认为是Perl中的单数的话，那么可以将列表（list）和数组认为是Perl中的复数。

列表是标量的有序集。数组是包含列表的变量。列表是指数据，而数组是其变量名。可以有一些值（列表）但不属于数组；但每一个数组标量都有一个列表，虽然这个列表可以为空。

下面的表是一个列表的例子。

|  |  |
| --- | --- |
| 索引 | 值 |
| 0 | 35 |
| 1 | 12.4 |
| 2 | “hello” |
| 3 | 1.72e30 |
| 4 | “bye\n” |

列表：将单个的标量按照一定的顺序排列起来构成列表。“一定的顺序”反映在索引编号上，规定编号从整数0开始，依次加1，因此数组或者列表的第一个元素的索引（即编号）为0。

**注：**根据现在的理解，我们可以把Perl和C对比起来理解，Perl中的列表（list）类似于C中的数组常量，Perl中的数组的概念与C中数组变量的概念类似，但是Perl允许数组中元素的数据类型可以是不同的（正如上面的列表举例中看到的）。

## 访问数组元素

数组中的元素是由连续整数编了号的，编号从0开始，例如：

$fred[0] = “yabba”;

$fred[1] = “dabba”;

$fred[2] = “doo”;

数组名和标量变量名属于不同的命名空间。同一个程序中也可以包含叫$fred的变量，但是不推荐这样做，可读性较低，不好维护。

数组元素与标量变量等价，在任何能够使用标量变量（如$fred）的地方也能够使用数组元素（如$fred[2]）。

**注**：实际上，有一种特殊情况，在foreach循环中的控制变量必须是标量变量。

下标可以是任何能返回数值的表达式，如果表达式的值不为整数，则自动将其转换为小于它的最大整数值。

$number = 2.71828;

print $fred[$number-1]; # equivalent to $fred[1]

如果下标超出了数组的范围，则返回的值为undef。通常，如果没有值存放在变量中，则其值为undef。

$blank = $fred[142\_857]; # no value stored in $fred[142\_857], so undef will be #returned.

$blanc = $mel; #$mel未存放值（未初始化），得到undef。

## 特殊的数组索引

有时候需要知道数组**最后一个元素对应的编号（即其索引）**，可以使用：

$#rocks：返回数组名为rocks的数组的最后一个元素的索引。

**需要注意**的是$#rocks得到的是最后一个元素对应的编号，不是数组rocks中元素的个数，因为数组的编号是从0开始的，数组元素的个数应该是最后一个元素的索引值加1。

其实在实际的程序中，我们常常会用到数组的最后一个元素的索引，我们通常不会使用$#array\_name的方式来获取数组中最后一个元素的索引，而是引入了负数索引值。**负数索引值从-1开始，对应于数组中的最后一个元素**，-2对应于数组中的倒数第二个元素，以此类推。实际上，几乎没有人使用除了-1之外的其他的负数索引值。

## 列表

数组是由括号括起来并且其元素由逗号分隔开的列表。

(1, 2, 3) #含有1,2,3的列表

(1, 2, 3) #同上，最后一个逗号被忽略

() #空列表

(1 .. 100) #包含100个整数的列表

**“..”为范围操作符**，

(1 .. 5) #同(1, 2, 3, 4, 5)

(1.7 .. 5.7) #同上——最小值和最大值被转换成整数

(5 .. 1) #空列表—— .. 中的左值应小于右值，否则为空

(0, 2 .. 6, 10, 12) #同(0, 2, 3, 4, 5, 6, 10, 12)

($m .. $n) #由$m和$n的值决定

(0 .. $#rocks)

列表中的元素并非必须是常数，也可以是在执行此语句时再计算值的表达式。

### qw简写

字符串列表在Perl中经常使用。使用qw可以不用输入大量引号来表示一个字符串列表，例如：

qw(fred barney betty wilma dino)

(“fred”, “barney”, “betty”, “wilma”, “dino”)

上面两种方式效果相同，但是采用qw的方式更为简洁。

qw表示“quoted words”或者“quoted by white space”。Perl将它们当作单引号字符串处理，不能像双引号那样在qw中使用\n（换行符）和$fred（变量替换）。

white space（空格、tabs、newlines等字符串）将被忽略，剩下的字符串们组成了列表的元素。

由于qw是一种引用，因此不可以在qw内添加注释。

上面的关于qw的用法举例中，使用的分隔符是括号。分隔符是用来界定引用的内容的，在两个分隔符内的被white space隔开的字符串构成了列表。实际上，Perl允许使用其他符号作为分隔符，Perl允许使用任何标点符号作为分隔符。下面列举出了一些常用的分隔符：

qw! fred barney betty Wilma dino !

qw# fred barney betty Wilma dino#

qw( fred barney betty Wilma dino )

qw{ fred barney betty Wilma dino }

qw[ fred barney betty Wilma dino ]

qw< fred barney betty Wilma dino >

如果在字符串中含有与选用的分隔符相同的字符，那么需要使用反斜线“\”，例如：

qw! Yahoo\! Google excite lycos ! #其中一个元素为：字符串“Yahoo!”

同单引号字符串一样，两个反斜线“\\”可以得到一个反斜线。

但是在选用分隔符时，应尽量避免与列表中包含的字符相同，例如：

qw{

/usr/dict/words

/home/rootbeer/.ispell\_english

}

在上面这个例子中，如果使用“/”作为分隔符，那么将会变的极为繁琐。

## 列表赋值

列表值也可以赋给变量：

($fred, $barney, $dino) = (“flintstone”, “rubble”, undef);

列表在赋值之前已经建立。这句话的意思是说：在列表赋值语句中，先计算得到赋值运算符右边的列表，再进行赋值。因此在Perl中可以用以下简单的方法来交换两个变量的值：

($fred, $barney) = ($barney, $fred);

如果等号两边的列表长度不相等，在列表赋值中，额外的值会被自动忽略，额外的变量会被赋予undef。

($wilma, $dino) = qw[flintstone]; # $dino is undef.

下面的代码使用列表赋值来创建一个字符串数组：

($rocks[0], $rocks[1], $rocks[2], $rocks[3]) = qw/talc mica feldspar quartz/;

当想引用这个数组时，Perl有一种简单的写法。在数组名前加@来引用整个数组。

@rocks = qw / bedrock slate lava /;

@tiny = (); # empty list

@giant = 1 .. 1e5;

@stuff = (@giant, undef, @giant);

@dino = “granite”;

@quarry = (@rocks, “crushed rock”, @tiny, $dino);

最后一个赋值语句将五个元素(bedrock, slate, lava, crushed, rock, granite)赋给数组变量@quarry，@tiny没有元素（需要指出的是，@tiny没有undef这个值）。

没有赋值的数组变量的值为()，即空表。数组变量不能成为列表的一个元素。

将一个数组拷贝到另一个数组时，使用列表赋值的方式，即值拷贝。

@copy = @quarry; # copy the value of one array to another

### pop和push操作

如果使用pop，push和类似的操作符可以避免使用索引，并且你的程序通常会比大量使用索引的情况要快，此外，还可以避免在计算索引时因为边界值引起的错误。

通常将数组作为栈来使用，在其右边添加或删除数据。（这是数组中“最后”一个元素，其索引值最大。）

**pop操作将数组的最后一个元素取出并返回，即将数组的最后一个元素从数组中删除，并将该元素的值返回。**

@array = 5 .. 9;

$fred = pop(@array); # $fred is 9, and @array becomes (5, 6, 7, 8)

$barney = pop @array; # $barney is 8, and @array becomes (5, 6, 7)

pop @array; # @array becomes (5, 6)

**如果数组为空，那么pop什么也不做，并返回undef。**

pop后可以使用或者不使用括号。如果去掉括号含义不变，那么括号就是可选的。

和pop操作相反的操作是push操作，push操作可以将一个元素（或者一列元素）加在数组的末尾。

push(@array, 0); # @array becomes (5, 6, 0)

push @array, 8; # @array becomes (5, 6, 0, 8)

push @array, 1 .. 10; # 1 .. 10 are added to @array

@others = qw/ 9 0 2 1 0 /;

push @array, @others; # five more elements are added to @array

**注：push的第一个参数或者pop的唯一参数必须是数组变量。**

### shift和unshift操作

push和pop对数组的末尾（数组下标最大的元素那一侧为数组的末尾）进行操作。相应的，unshift和shift对一个数组的开头进行操作。

@array = qw#dino fred barney#;

$m = shift(@array); # $m = dino, @array = qw(fred barney)

$n = shift @array ; # $n = fred, @array = qw(barney)

shift @array; # @array becomes empty

$o = shift @array; # $o gets undef, because @array is empty.

## 将数组插入字符串

和标量类似，数组也可以插入双引号字符串中。插入的数组元素前后会自动添加空格隔开。插入的数组元素的第一个元素前面和最后一个元素后面不会插入空格，如果需要可以自己添加。

@rocks = qw{flintstone slate rubble};

print “quartz @rocks limestone\n”;

如果忘了数组插入的规则，当把email地址插入双引号字符串时可能会出现意想不到的结果。在Perl 5之前，Perl将不会替换没有定义过的数组标量。但是，一旦使用了与email地址中相同的数组标量时，将会出现错误。

@bedrock = qw{ flintstone slate rubble };

$email = [fred@bedrock.edu](mailto:fred@bedrock.edu); # Wrong! @bedrock will be replaced.

$email = “fred\@bedrock.edu”;

$email = ‘fred@bedrock.edu’; #使用单引号字符串回避字符替换。

**索引表达式**被当作普通表达式求值，但是索引表达式中的变量是不会被首先赋值的。也就是说，下面这个例子中，如果$y = 2 \* 4，那么索引表达式$y – 1的值为1，而不是7。因为“2\*4”首先会被当作一个数字（$y在数字表达式中）为2。如果想在一个标量变量后接一个左中括号符，那么应该在中间加入分隔符，否则会被当作数组变量看待。

@fred = qw(hello dolly);

$y = 2;

$x = “This is $fred[1]’s place”;

$x = “This is $fred[$y - 1]’s place”;

## foreach控制结构

如果能处理整个数组或列表，那将是非常方便的，Perl提供了foreach语句来处理整个数组。foreach从列表的第一个元素一直循环执行到最后一个元素，一次迭代一个元素。

foreach $rock (qw/ bedrock slate lava /){

print “One rock is $rock.\n”;

}

上面的foreach循环语句举例中，$rock为foreach循环语句的**控制变量**。控制变量每一次迭代从列表中取出一个新值。**需要注意的是**：控制变量从列表中取值时，取的不是这些列表元素的一个拷贝而是这些元素本身。也就是说，**如果在某一次循环中，对控制变量进行了赋值，那么这一次控制变量对应的列表元素也将会被更改。**

如下面这个例子所示：

@rocks = qw / bedrock slate lava /;

foreach $rocks(@rocks){

$rock = “\t$rock”; #@rocks的每一个元素前加入一个tab

$rock **.=** “\n”; #注意赋值运算是“.=”，在每个元素后加入一个换行符

}

那么，当循环结束以后$rock的值是多少呢？**循环结束以后$rock（控制变量）的值与循环开始前的值相同。**foreach循环中控制变量的值会被Perl自动保存和恢复。循环结束后，变量的值会回复到循环开始前，如果在循环开始前没有值，则为undef。

### Perl最常用的默认变量：$\_

如果在foreach循环中省略了控制变量，那么Perl会使用默认的变量：$\_。除了其不寻常的名字外，这和普通变量类似，如下面的例子所示：

foreach(1 .. 10){

print “I can count to $\_!\n”;

}

虽然它不是Perl中唯一的默认变量，但无疑是使用的最普遍的。在Perl中很多时候没有指定使用某个变量时，都会自动使用$\_，这将使得程序员不用花时间去思考使用哪个变量。

$\_ = “Yabba dabba doo\n”;

print; #打印出默认变量$\_。