

## 第4学时 基本构件的堆栈：列表与数组

标量是Perl的单数名词。它们可以代表任何一种元素，如单词、记录、文档、一行文本或者一个字符。但是，有时需要一些元素的集合，比如许多个单词、几个记录、两个文档、50行文本或者十几个字符等。

当需要谈论Perl中的许多东西时，可以使用列表数据。可以用3种方法来表示列表数据，它们是列表、数组和哈希结构。

列表是列表数据最简单的表示方法，它们只是一个标量的组合。有时它们使用一组括号将标量括起来，各个标量之间用逗号隔开。例如，(2, 5, \$a, "Bob") 是两个数字，一个标量\$a和单词"Bob"的列表。列表中的每个项目称为列表元素。为了不违背自然随意的原则，Perl的列表可以根据你的需要包含任意多个元素。由于列表是标量的集合，并且标量也可以任意大，因此列表能够存放相当多的数据。

若要将一个列表存放在一个变量中，需要一个数组变量。在Perl中，数组变量用一个符号(@)后随一个有效的变量名（第1学时中的“数字与字符串”这一节做了介绍）来表示。例如，@FOO就是Perl中的一个有效的数组变量。数组变量可以与标量变量使用相同的名字，例如，\$names与@names可以指不同的东西，\$names指一个标量变量，而@names则指一个数组。这两个变量之间毫无关系。

数组中的各个项目称为数组元素。各个数组元素按它们在数组中的位置来引用，这个位置称为索引（比如说，数组@FOO的第三个元素，或者数组@names的第五个元素等等）。

另一种列表类型是哈希结构，它类似数组。哈希结构将在第7学时中详细介绍。

在本学时中，我们将要介绍：

- 如何填充和清空数组。
- 如果逐个元素查看数组。
- 如何对数组进行排序和输出。
- 如何将标量分割成数组，以及如何将数组重新合成为标量。

### 4.1 将数据放入列表和数组

将数据放入一个列表是很容易的。正如你刚刚看到的那样，列表的语法是用一组括号将一些标量值括起来。下面就是列表的一个例子：

```
(5, 'apple', $x. 3.14159)
```

这个例子用于创建一个由4个元素组成的列表，它包含数字5、单词apple、标量变量\$x和值。如果列表只包含简单的字符串，而用单引号将每个字符串括起来对你来说又太麻烦，那么Perl提供了一个快捷方式，即qw运算符。下面是使用qw的一个例子：

```
qw (apples oranges 45.6 $x)
```

这个例子创建了一个由4个元素组成的列表。列表的每个元素之间用一个白空间（空格、制表符或换行符）隔开。\$x是个直接量\$x，它没有内插到它的值中去。如果有一些嵌入了

白空间的列表元素，那么就不能使用 qw 运算符。在这种情况下，上面这个代码的作用就像编写的是下面这个代码一样：

```
('apples', 'oranges', '45.6' '$x')
```

请注意，\$x是用单引号括起来的。qw没有对看起来像变量的元素进行变量值内插，它们作为常规的形式来处理的，因此 '\$x'没有被转换成标量变量\$x的任何值，它只是留下了一个美元符号和字母x。

Perl有一个非常有用的能够对列表进行操作的运算符，称为范围运算符。范围运算符由一对圆点(..)来表示。下面是该运算符的用法的例子：

```
(1..10)
```

范围运算符用一个左边的操作数(1)和右边的操作数(10)构成了一个包含1到10(含1与10)之间的所有数的列表。如果需要在列表中使用若干个范围，那么只要使用多个范围运算符即可：

```
(1..10, 20..30);
```

上面这个例子创建了一个包含21个元素的列表，即包含1到10和20到30(含1、10、20和30)之间的数。如果范围运算符的右边的操作数小于左边的操作数，比如(10..1)，那么将产生一个空列表。范围运算符既可以用于字符串，也可以用于数字。范围(a..z)可以产生一个包含所有26个小写字母的列表。范围(aa..zz)可以生成一个大得多的列表，它由675个字母对组成，从aa、ab、ac、ad开始，到zx、zy、zz结束。

## 数组

直接量列表通常用于对某些其他结构进行初始化，比如数组或哈希结构。若要在 Perl 中创建一个数组，只需要将某些数据放入数组即可。Perl 与其他编程语言不同，不必预先告诉 Perl，你要创建一个数组，或者该数组将有多大。若要创建一个新数组，并用一个项目列表填入该数组，只要编写下面这个代码即可：

```
@boy=qw(Greg Peter Bobby);
```

这个例子称为数组赋值，它使用数组赋值运算符——等号，这与标量中的情况一样。当运行该代码后，数组 @boys 将包含3个元素，即Greg、Peter和Bobby。请注意，该代码也使用了qw运算符。使用该运算符后，你就不必键入6个引号和2个逗号。数组赋值也可以包含其他数组甚至空列表，如下面的例子所示：

```
@copy=@original;  
@clean=();
```

在这里，@original数组的所有元素都被拷贝到新数组 @copy中。如果 @copy 中原先已经拥有元素，那么这些元素就会丢失。这时 @clean 就变成空数组。将一个空列表(或者空数组)赋予一个数组变量，就会从该数组中删除所有的元素。

如果直接量列表中包含了其他列表、数组或哈希结构，那么这些列表将全部合并成一个大数据表。请看下面这个代码段：

```
@girls=qw( Marcia Jan Cindy );  
@kids=(@girls, @boys);  
@family=(@kids, ('Mike', 'Carol'), 'Alice');
```

在将有关的值赋予 @kids 数组之前，列表 ( @girls , @boys ) 被 Perl 合并成一个由所有小孩名字 ( Greg , Peter 等等 ) 组成的简单列表。在下一行上，数组 @kids 被合并，同时列表 ( 'Mike' , 'Carol' ) 被合并成一个长列表，然后该列表被赋予 @family。@boys、@girls、@kids 的原始结构和列表 ( 'Mike' , 'Carol' ) 本身则没有保留在 @family 中，保留的只是各个元素，即 Mike 和 Carol。

这意味着上面这个用于建立 @family 的代码段与下面这个赋值语句是等价的：

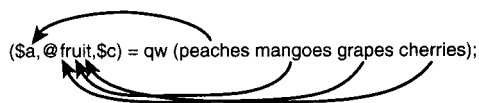
```
@family=qw(Greg peter Bobby Marcia Jan Cindy Mike Carol Alice);
```

如果赋值数组左边的列表只包含变量名，那么该列表可以用来对其元素进行初始化。请看下面这个例子：

```
($a, $b, $c)=qw (apples oranges bananas);
```

在这个例子中，\$a 被初始化为 'apples'，\$b 被初始化为 'oranges' 而 \$c 则被初始化为 'bananas'。如果左边的列表包含一个数组，那么该数组就接受来自右边列表的剩余值。现在请看下面的例子：

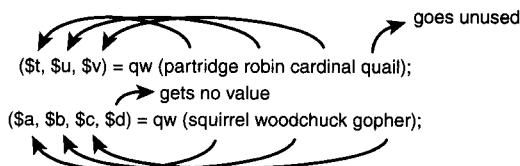
```
($a,@fruit,$c)=qw (peaches mangoes grapes cherries);
```



在这个例子中，\$a 被设置为 'peaches'。右边列表中的其余水果被赋予左边的 @fruit。\$c 没有留下任何元素来接受一个值（因为赋值语句左边的数组吸收了来自右边的所有剩余值），因此 \$c 被设置为 undef。

另一个值得注意的问题是：如果左边包含的变量比它拥有的元素多，那么多余的变量将接受 undef 这个值。如果右边的变量比左边的元素少，那么右边多余的元素将被忽略。下面让我们观察一个代码，以便理解这个概念：

```
($t, $u, $v) = qw (partridge robin cardinal quail);
($a, $b, $c, $d) = qw (squirrel woodchuck gopher);
```



在第一行中，\$t、\$u 和 \$v 均接受来自右边的值。右边多余的元素 ( 'quail' ) 将不用于该表达式。在第二行中，\$a、\$b 和 \$c 均接受来自右边的值。但是 \$d 没有从右边得到任何值 ( \$c 得到了最后一个值 'gopher' )，因此 \$d 被设置为 undef。

## 4.2 从数组中取出元素

到现在为止，我们在本学时中一直是在介绍整个数组和列表方面的内容。但是在许多情况下，需要获得数组中的各个元素，需要搜索数组，改变元素的值，或者将元素添加给数组或从数组中删除各个元素。

若要获得整个数组的内容，最简单的方法是使用双引号中的数组：

```
print "@array";
```

这个例子用于输出 @array 的元素，每个元素之间用空格隔开。数组中的各个元素可以按索引来访问，如下面这个代码所示。数组元素的索引从数字 0 开始，每增加一个元素，索引便递增 1。数组的每个元素都有一个索引值，如下所示：

@trees	0	1	2	3
	oak	cedar	maple	apple

数组中元素的数量只受系统内存的限制。若要访问一个元素，可以使用句法 `$array[index]`，其中 `array` 是数组的名字，`index` 是你想要的元素的索引。在引用各个元素之前，数组不一定存在。数组会魔术般地自动弹出来。下面是访问数组元素的一些例子：

```
@trees=qw(oak cedar maple apple);
print $trees[0];      # Prints "oak"
print $trees[3];      # Prints "apple".
$trees[4]='pine';
```

请注意，如果要指 `@trees` 中的某个元素，该代码可以使用一个 `$`。你可能会问：“`$` 标记通常是保留供标量使用的，这里是怎么回事呢？”在 `$trees[3]` 中的 `$` 是指 `@trees` 中的一个标量值。标量也可以用美元符号来表示，因为它们也是单数。你应该注意这里的一个模式。

在本学时的开头，我们讲过标量和数组可以拥有相同的变量名，但是它们互不相关。Perl 能够说明 `$trees` 与 `@trees[0]` 之间的差别，因为 `$trees[0]` 中有一个方括号。Perl 知道你指的是 `@trees` 的第一个元素，而根本不是指 `$trees`。

还可以将数组划分成分组，称为片。若要使用数组的一个片，可以使用 `@` 标号，以指明你说的是一组元素，也可以使用方括号，以指明你说的是数组的各个元素，如下所示：

```
@trees=qw(oak cedar maple apple cherry pine peach fir);
@trees[3,4,6];      # Just the fruit trees
@conifers=@trees[5,7]; # Just the conifers
```

#### 4.2.1 寻找结尾

有时需要寻找数组的结尾，例如，要查看 `@trees` 数组中究竟有多少树状结构，或者从 `@trees` 数组中砍下一些树枝来。Perl 提供了两个机制，可以用来查找数组的结尾。第一个方法是个特殊变量，其形式是 `$#arrayname`。它能够返回数组的最后一个有效索引的号码。请看下面这个例子：

```
@trees=qw(oak cedar maple apple cherry pine peach fir);
print $#trees;
```

这个例子包含 8 个元素，但是你必须记住，数组是从 0 开始编号的。因此，上面这个例子输出的号码是 7。如果修改 `$#trees` 的值，就会改变数组的长度。如果要缩小数组，请在你设定的某个索引处截断数组；如果要扩大数组，那么就给它增加更多的元素。新增加的元素的值将全部设置为 `undef`。

寻找数组大小的另一种方法是在期望存在标量的位置上使用数组变量：

```
$size=@array;
```

这将把 `@array` 中的元素数量放入 `$size` 中。它利用了 Perl 的一个概念，称为上下文(环境)，这将在下一节中介绍。



也可以为数组设定负索引。负索引号从数组的结尾开始计数，然后反向递增。例如，`$array[-1]` 是 `@array` 的最后一个元素，`$array[-2]` 是倒数第二个元素，依次类推。

#### 4.2.2 关于上下文的详细说明

什么是上下文呢？上下文是指有关项目周围的一些事物，这些事物可以用来帮助定义这个项目的含义。例如，你看到一个人穿着外科消毒服，根据他所在的地方，这可能表示不同的含义。如果是在医院，那么穿着这样的衣服的人是一名医生。如果是在万圣节上，那么他只是一位化装了宾客。

人类的语言使用上下文来帮助确定词汇的含义。例如，单词 level 可以拥有若干不同的含义，这要根据如何使用这个单词以及在什么上下文中使用它：

- 木工使用水平仪 (level) 使安装的门能够达到水平。
- 调解人说话的语调平和 (level)。
- 池塘里的水只有齐腰深 (level)。

虽然每次使用的单词都是 level，但是它的含义在不同的场合却发生了变化。根据它在句子中的不同用法，它可以是个名词或形容词，也可以是另一种类型的名词。

Perl 能够对上下文作出敏感的反应。在 Perl 中，函数和运算符能够根据使用时所在的上下文而具有不同的运行特性。Perl 中的两个最重要的上下文是列表上下文和标量上下文。

如你所见，可以将赋值运算符（等号）用于数组和标量。赋值运算符左边的表达式类型（列表或标量）用于决定右边的表达式计算时所在的上下文。现在请看下面这个代码段：

```
$a=$b;      # Scalar on the left, this is scalar context.
@foo=@bar;  # Array on the left, this is list context
($a)=@foo;  # List on the left, this is also list context.
$b=@bar;    # Scalar on the left, this is scalar context.
```

该代码段的最后一行很有意思，因为在标量上下文中计算时，数组将返回该数组中元素的数量。

观察下面这几行代码中的 \$a 和 \$b，它们执行的操作几乎相同：

```
@foo=qw( water peps coke lemonade );
$a=@foo;
$b=$#foo;
print "$a\n";
print "$b\n";
```

在该代码的结尾，\$a 包含数字 4，\$b 包含数字 3。为什么会出现这个差别呢？因为 @foo 是在标量上下文中运行的，它返回 \$a 的元素的数量。\$b 设置为最后一个元素的索引号，而索引是从 0 开始计数的。由于标量上下文中的数组返回数组中的元素的数量，因此测试数组中是否包含元素就变得如此简单：

```
@mydata=qw( oats peas beans barley );
if (@mydata) {
    print "The array has elements!\n";
}
```

在这里，数组 @mydata 被计算为一个标量，它返回元素的数量，在这个例子中，这个数量是 4。在 if 语句中，数量 4 计算的结果是真，if 语句块的本身就开始运行。



实际上，@mydata 在这里是用在一个特殊的标量上下文中，称为布尔上下文，不过它的运行特性是相同的。当 Perl 希望得到一个真或假的值时，就会出现布尔上下文，比如在一个 if 语句的测试表达式中。另一个上下文称为无效(void)上下文，我们将在第 9 学时中对该上下文进行介绍。



### 4.2.3 回顾以前的几个功能

Perl的许多运算符和函数能够强制它们的参数成为标量上下文或列表上下文。有时这些运算符和函数的运行特性将根据它们所在的上下文而各不相同。你已经遇到的有些函数具备这些属性，但是以前这个情况却并不重要。

Print函数希望有一个列表作为其参数，不过该列表在什么上下文中计算却并不特别重要。因此，用类似这个print的函数来输出数组将会导致该数组在列表上下文中被计算，从而产生@foo的各个元素：

```
print @foo;
```

可以使用一个称为scalar的特殊伪函数来强制将某个东西放入标量上下文：

```
print scalar (@foo);
```

这个例子用于输出 @foo中的元素的数量。Scalar函数强制 @foo在一个标量上下文中进行计算，因此 @foo返回 @foo中的元素的数量。然后print函数就输出返回的数量。

你在第2学时中了解到的chomp函数既能够将数组作为参数，也能够将标量作为其参数。如果chomp函数获得一个标量，那么它就从标量的结尾处删除记录分隔符。如果它获得一个数组，它将从数组中的每个标量的结尾处删除记录分隔符。

我们在第2学时中介绍chomp时，还讲述了如何使用<STDIN>读取键盘输入的一行数据。尖括号实际上是Perl中的一个运算符，它们的运行特性随着上下文的不同而各有差异。在标量上下文中，该运算符能够读取来自终端的一行输入。但是在列表上下文中，它能够读取来自终端的所有输入，直到读到文件的结尾，并将数据放入列表。请看下面这个代码：

```
$a=<STDIN>; # Scalar context, reads one line into $a.  
@whole=<STDIN>; # List context, reads all input into the array @whole.  
($a)=<STDIN>; # List context, reads all input into the assignable list.
```

在第三个例子中，\$a将收到什么呢？本学时的开头我们讲过，在列表赋值语句中，如果左边没有足够的变量来存放右边的全部元素，那么右边的多余元素将被放弃。这里，来自终端的所有输入均被读取，但是\$a只接收第一行。



什么是文件结尾呢？当Perl读取来自终端的全部输入且你完成Perl数据的输入时，你必须发出通知。为此通常键入一个End of File（文件结束）字符（EOF）。该字符随着你使用的操作系统的不同而各有差别。在UNIX下，该字符通常是在一行的开头使用Ctrl+D。在MS\_DOS或者Windows系统上，该字符是在输入的任何位置两次使用Ctrl+Z。

我们在第1学时中介绍的重复运算符在列表上下文有一个特殊的运行特性。如果左边的操作数放在括号中，那么运算符本身将用在列表上下文中，它返回一个重复的左边操作数的列表。下面的例子用于建立一个100个星号的数组：

```
@stars= ("*") x 100;
```

x左边的操作数“\*”放在括号中，如果将它赋予一个数组，那么就使它进入一个列表上下文中。该句法可以用于将数组的元素初始化为一个特定的值。

另一个运算符你一直在使用，但是可能不知道它是个运算符，这就是逗号（,）。到现在为止，你一直使用逗号来分隔列表中的各个元素，比如下面这个例子：

```
@pets=('cat', 'dog', 'fish', 'canary', 'iguana');
```

上面这个代码段是在通常的列表上下文中对列表进行计算操作的。但是，在标量上下文中使用的逗号是个运算符，它用于从左至右对每个元素进行计算，再返回最右边的元素的值：

```
$last_pet=('cat', 'dog', 'fish', 'canary', 'iguana'); # Not what you think!
```

在这个代码段中，赋值运算符右边的那些宠物名字实际上并不是一个列表。它们只是一组字符串直接量，从左到右进行计算，因为表达式的右边是在标量上下文中计算的（因为等号的右边有一个`$last_pet`）。结果是该`$last_pet`被设置为等于`'iguana'`。

另一个例子是`localtime`函数，根据它所在的上下文，可以用两种完全不同的方法来运行。在标量上下文中，`localtime`函数返回一个格式化很好的当前时间字符串。例如，`print scalar(localtime)`这个代码，它输出的结果将类似于`Thu Sep 16 23:00:06 1999`。在列表上下文中，`localtime`将返回能够描述当前时间的一个元素列表：

```
($sec, $min, $hour, $mday, $mon, $year_off, $wday, $yday, $isdst)=localtime;
```

表4-1 显示了这些值代表的含义。

表4-1 列表上下文中`localtime`返回的值

字 段	值
<code>\$sec</code>	秒，0 ~ 59
<code>\$min</code>	分，0 ~ 59
<code>\$hour</code>	时，0 ~ 23
<code>\$mday</code>	月份中的日期，1 ~ 28、29、30或31
<code>\$mon</code>	年份中的月份，0 ~ 11（这里请特别要小心）
<code>\$year_off</code>	1900年以来的年份。将1900加上这个数字，得出正确的4位数年份
<code>\$wday</code>	星期几，0 ~ 6
<code>\$yday</code>	一年中的第几天，0 ~ 364或365
<code>\$isdst</code>	如果夏令时有效，则为真



不要将`'19'`附加给`localtime`返回的年份。它返回的年份是1900的偏移量。比如，在1999年，年份是`'99'`；在2000年中，它是`'100'`。将1999与该值相加，可以在2000年以后正确地产生年份。Perl不存在2000年问题，但是，如果简单地将`'19'`（或`'20'`）附加给该年份，就会导致程序中产生2000年问题。

怎么能够知道函数或运算符使它的参数在什么上下文中进行计算呢？又怎么知道它在标量上下文或列表上下文中运行的情况呢？很简单，你无法知道，你也没有什么好办法来猜测这个问题的答案。如果你不清楚的话，在线文档列出了每个函数和运算符，并且说明了它们的各个因子。在本书的其他章节中，如果一个函数或运算符强制让它的参数在某个上下文中运行，或者它们在自己运行的上下文中出现了完全不同的运行特性，那么我们将在首次介绍该函数时指明这些情况。这种情况不会经常发生，不过当本书中发生这种情况时，一定有特别说明。

### 4.3 对数组进行操作

既然你已经了解到创建数组的基本规则，那么现在我们可以介绍一些工具来帮助你

这些数组进行操作，以便执行一些有用的任务。

### 4.3.1 遍历数组

在第3学时中，我们介绍了如何使用 while、for和其他结构进行循环的方法。你想使用数组执行的许多操作都涉及到查看数组的每个元素，这个进程称为数组的迭代。迭代的方法之一是使用for循环，如下所示：

```
@flavors=qw( chocolate vanilla strawberry mint sherbert );
for($index=0; $index<@flavors; $index++) {
    print "My favorite flavor is $flavors[$index] and..."
}
print "many others.\n";
```

第一行代码用于以冰淇淋的风味对该数组进行初始化，为了清楚起见，它使用了 qw运算符。如果使用两个单词组成的冰淇淋风味，比如 Rocky Road，那么将需要一个普通的单引号列表。第二行进行了代码的大部分工作。\$index被初始化为0，并且按1进行递增，直到到达@flavors。请记住，@flavors是在标量上下文中计算的，计算的结果是5，即@flavors中的元素的数量。

上面这个例子似乎要对数组的迭代进行大量的工作。在Perl中，如果需要进行大量的工作，那么通常可以找到一种比较简单的方法来进行这项操作，这没有例外。Perl还有另一个循环语句，称为foreach语句，我们在第3学时没有介绍。foreach语句设置一个索引变量，称为迭代器，它相当于列表的每个元素。请看下面这个例子：

```
foreach $cone (@flavors) {
    print "I'd like a cone of $cone\n";
}
```

在这个代码中，变量\$cone设置为@flavors中的各个值。当\$cone被设置为@flavors中的各个值时，循环体就开始执行，为@flavors中的每个值输出消息。请注意，在一个foreach循环中，迭代器并不只是设置为列表中的每个元素的值，它实际上是对列表的元素的引用。因此，在上面这个foreach循环中，如果修改该循环中的\$cone，就能修改@flavors中的对应元素。下面让我们来观察一下这个例子：

```
foreach $flavor (@flavors) {
    $flavor="$flavor ice cream"; # This modifies @flavors!
    print "I'd like a bowl of $flavor, please.";
}
```

第2行修改了\$flavor，将ice cream附加给了结尾处，因此第3行负责输出I'd like a bowl of chocolate ice cream，然后继续输出vanilla，strawberry等等。当该循环最后运行结束时，@flavors改为在每个元素的结尾处包含ice cream。



在Perl中，foreach和for循环语句实际上是同义语句，它们可以互换使用。为了清楚起见，在本书的整个篇幅中，你将发现使用foreach()循环语句在数组上进行迭代运行，并将for()循环语句用于像第3学时中那样的普通for循环。请记住它们是可以互换的。

### 4.3.2 在数组与标量之间进行转换

一般来说，Perl并没有关于在标量与数组之间进行转换的规则。我们提供了许多函数和运



算符，以便进行它们之间的转换。将标量转换成数组的方法之一是使用 `split` 函数。`Split` 函数拥有一个模式和一个标量，并且使用该模式来分割该标量。第一个参数是该模式（这里用斜杠括起来），第二个参数是要分割的标量：

```
@words=split(/ /, "The quick brown fox");
```

当运行该代码时，`@words` 包含了各个单词，即 `The`、`quick`、`brown` 和 `fox`，单词之间没有空格。如果要设定一个字符串，请使用变量 `$_`。如果没有设定一个模式或字符串，那么使用空白来分割变量 `$_`。一个特殊模式 `' '`（即空模式）用于将标量分割成各个字符，如下面所示：

```
while(<STDIN>) {
    ($firstchar)=split(/ /, $_);
    print "The first character was $firstchar\n";
}
```

第一行用于读取来自终端的数据，每次读一行，并将 `$_` 设置为等于该行。第二行使用空模式来分割 `$_`。`Split` 函数返回来自 `$_` 中的这个行的每个字符的列表。该列表被赋予左边的列表，而该列表的第一个元素则被赋予 `$firstchar`，其余均放弃。



`split` 使用的模式实际上是一些正则表达式。正则表达式是一种复杂的模式匹配语言，我们将在第 6 学时介绍这方面的内容。而现在，我们的例子将使用一些简单的模式，如空格、冒号、逗号等等。当你学习了正则表达式的内容之后，我们将举例说明如何使用比较复杂的模式来用 `split` 分割标量。

在 Perl 中使用这种方法来分割标量变量的列表是非常常见的。当分割一个标量，而标量中的每个元素都是与众不同的元素（比如记录中的域）时，就能够比较容易地确定哪个元素是什么。请看下面这个例子：

```
@Music=('White Album,Beatles',
        'Graceland,Paul Simon',
        'A Boy Named Sue, Goo Goo Dolls');
foreach $record (@Music) {
    ($record_name, $artist)=split(/ /, $record);
}
```

当你直接分割带有命名标量的列表时，能够清楚地看到哪个域代表什么。第一个域是记录名，第二个域是艺术家。如果将代码分割成一个数组，各个域之间的区别也许不会那样清楚。

若要用数组来创建一个标量，也就是进行 `split` 的反向操作，可以使用 Perl 的 `join` 函数。`join` 函数取出一个字符串和一个列表，使用该字符串将列表的各个元素组合在一起，然后返回产生的字符串。请看下面这个例子：

```
$numbers=join(',', (1..10));
```

这个例子将字符串 `1, 2, 3, 4, 5, 6, 7, 8, 9, 10` 赋予 `$numbers`。然后可以使用 `split` 和 `join` 将字符串分割后又将它们重新组合在一起。一个函数的输出（返回值）可以用作另一个函数的输入值，请看下面的代码：

```
$message="Elvis was here";
print "The string \"$message\" consists of:",
      join('-', split(/ /, $message));
```

在这个例子中，\$message被split分割成一个列表。该列表被join函数使用，并用逗号重新组合在一起。产生的结果是下面这个消息：

```
The string "Elvis was here" consists of: E-l-v-i-s- -w-a-s- -h-e-r-e
```

#### 4.3.3 给数组重新排序

当你创建一个数组时，常常想让它们用不同于创建时的顺序来显示。例如，如果你的 Perl 程序从文件中读取一个客户列表，那么用字母顺序来输出该客户列表是可行的。若要给数据排序，Perl提供了sort函数。Sort函数将一个列表作为它的参数，并且大体上按照字母顺序对列表进行排序，然后该函数返回一个排定顺序的新列表。原始数组保持不变，如下面这个例子所示：

```
@Chiefs=qw(Clinton Bush Reagan Carter Ford Nixon);  
print join(' ', sort @Chiefs);
```

这个例子用于输出Bush Carter Clinton Ford Nixon Reagan。应该预先注意的是，它的默认排序次序是ASCII顺序。这意味着以大写字母开头的所有单词均排在以小写字母开头的单词的前面。用ASCII顺序对数字进行排序的方式与你期望的不一样，它们不按值来排序。例如，11排在100的前面。在这种情况下，必须按非默认排序来进行排序。

使用sort函数，你可以使用代码块（或者子例程名）作为第二个参数，按照你想要的任何顺序进行排序。在代码块（或者子例程）中，两个变量 \$a和\$b被设置为列表的两个元素。代码块的任务是：如果 \$b小于、等于或大于 \$a，则分别返回 -1、0或1。下面是进行数字排序的硬办法，假设 @numbers是包含了许多数字值的话：

```
@sorted=sort { return(1) if ($a>$b);  
               return(0) if ($a==$b);  
               return(-1) if ($a<$b); } @numbers;
```

上面这个例子当然按照数字顺序给 @numbers进行排序。但是，该代码从事这样一个普通的排序任务看起来太复杂了。由于你可能认为这个方法太麻烦，所以 Perl有一个捷径可供使用，你可以使用飞船运算符 <=>。飞船运算符因为从侧面看它像一个飞行的碟子而得名。如果它左边的操作数小于右边的操作数，那么它返回 -1，如果左边的操作数大于右边的操作数，则返回0：

```
@sorted=sort { $a<=>$b; } @numbers;
```

这个代码看上去比较清楚，并且更加直观。飞船运算符只应该用来比较数字值。

若要比较字母字符串，请使用cmp运算符，它的运行方式与飞船运算符完全相同。只需要编写一个更加复杂的排序例程，就可以将更加复杂的排序参数放在一起。如果你需要了解更多的情况，本学时第7节中的Perl常见问题给出了一些比较复杂的例子。

本学时要介绍的最后一个函数是个非常容易使用的函数，即 reverse。当在标量上下文中被赋予一个标量值时，reverse函数能够对字符串的字符进行倒序操作，返回倒序后的字符串。例如，在标量上下文中调用 reverse("Perl")将返回lrep。当在列表上下文中被赋予一个列表时，reverse函数能够返回倒序后的列表元素，如下面的例子所示：

```
@lines=qw(I do not like green eggs and ham);  
print join(' ', reverse @lines);
```

这个代码段用于输出ham and eggs green like not do I。若要继续进行这个试验，充分展示该函数堆栈的功能，可以给混合列表添加更多的奇怪内容：

```
print join(' ', reverse s@tlines);
```

该代码首先运行 sort 函数，产生一个莫名其妙的列表（I, and, do, eggs, green, ham, like, not）。该列表被倒序后再被传递给 join 函数，以便将这些元素连接起来，并且加上一个空格。结果是 not like ham green egg do and I，真不敢恭维这样的句子。

#### 4.4 练习：做一个小游戏

本学时充满了许多新鲜的内容，比如，你熟悉的运算符在不同的上下文中产生不同的运行特性，一些新的函数和运算符，还有一些需要记住的新的句法规则。为了使你不至于编写出难以运行的代码，所以增加了这个练习，让你做一个游戏，使你具备的关于数组和列表的知识能够得到很好的应用。

使用文本编辑器，将程序清单 4-1 中的程序键入编辑器，并将它保存为 Hangman。务必按照第1学时中的说明使该程序成为可执行程序。

当完成上面的操作后，键入下面这个命令，使该程序运行：

```
perl Hangman
```

程序清单 4-1 完整的 Hangman 程序清单

```
1:  #!/usr/bin/perl -w
2:
3:  @words=qw( internet answers printer program );
4:  @guesses=();
5:  $wrong=0;
6:
7:  $choice=$words[rand @words];
8:  $hangman="0-|--<";
9:
10: @letters=split(//, $choice);
11: @hangman=split(//, $hangman);
12: @blankword=(0) x scalar(@hangman);
13: OUTER:
14:     while ($wrong<@hangman) {
15:         foreach $i (0..$#letters) {
16:             if ($blankword[$i]) {
17:                 print $blankword[$i];
18:             } else {
19:                 print "-";
20:             }
21:         }
22:         print "\n";
23:         if ($wrong) {
24:             print @hangman[0..$wrong-1]
25:         }
26:         print "\n Your Guess: ";
27:         $guess=<STDIN>; chomp $guess;
28:         foreach(@guesses) {
29:             next OUTER if ($_ eq $guess);
30:         }
31:
32:         $right=0;
33:         for ($i=0; $i<@letters; $i++) {
34:             if ($letters[$i] eq $guess) {
35:                 $blankword[$i]=$guess;
36:                 $right=1;
```

```

37:         }
38:     }
39:     $wrong++ unless(not $right);
40:     if (join('', @blankword) eq $choice) {
41:         print "You got it right!\n";
42:         exit;
43:     }
44: }
45: print "$hangman\nSorry, the word was $choice.\n";

```

第1行：包含到达解释程序的路径（可以修改这个路径，使它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：数组 @words 使用游戏能够使用的单词列表进行初始化。

第4~5行：有些变量被初始化。@guesses 用于存放游戏的玩主过去猜测的所有单词的列表。@wrong 用于存放迄今为止猜错的单词的数量。

第7行：从数组 @words 中随机选择的一个单词，并将它赋予 \$choice。rand ( ) 函数期望得到一个标量参数，由于 @words 被视为一个标量，因此它返回元素的数量（在这里返回 4）。rand 函数返回一个 0 至 3 之间的一个数字，但是不包括 0 或 4。结果，当将一个十进制数字用作数组索引时，小数部分将被放弃。

第8行：hangman 被定义。他很难看，但是通过了。

第10行：\$choice 中的 mystery 单词在 @letters 中被分割成各个字母。

第11行：hangman 标量在 @hangman 中被分割成小块。头是 \$hangman[0]，颈部是 \$hangman[1]，等等。

第12行：数组 @blankword 用于标明游戏的玩主猜对了哪些字母。(0) x scalar ( @hangman ) 创建了一个列表，其长度与 @hangman 中的元素的数量相同，它被存放在 @blankword 中。当猜字母时，这些 0 改变为第35行中的字母，这将标出猜对的字母的位置。

第13~14行：建立一个包含大部分程序的循环。它有一个标识符 OUTER，这样，在循环的内部，就能够对它进行某些具体的控制。它不断进行循环，直到猜错的字母数量与 hangman 的长度相同为止。

第15~21行：这个 foreach 循环为猜测的每个字母在数组 @blankword 上迭代运行。如果 @blankword 不包含该特定元素中的一个字母，那么就输出一个破折号，否则，输出该字母。

第23~25行：\$wrong 包含猜错的字母的数量。如果这个数量至少是 1，那么第24行就使用一个程序片来输出 hangman 数组，从位置 0 开始，直到猜错的字母的数量（减 1）。

第26~27行：这两行用于获得游戏玩主的猜测。chomp ( ) 删除结尾处的换行符。

第28~30行：这几行用于搜索 @guesses，以了解玩主是否已经猜到该字母。如果他已经猜到，就可以重新启动第13行的循环。如果玩主多次猜错，他不会受到处罚。

第32~38行：这是该程序的核心！搜索包含猜字母游戏的 @letters 数组。如果在游戏中找到了猜测的字母，那么对应的 @blankword 元素就被设置为该字母。数组 @blankword 既包含猜对的字母，也包含任何特定元素中的 undef。称为 \$right 的标志被设置为 1，表示至少有一个字母已经被找到。

第39行：除非游戏的玩主猜到了一个字母，否则 \$wrong 被递增。

第40~43行：数组 @blankword 的各个元素被连接在一起，构成一个字符串，并与原始的要猜测的字符串相比较。如果它们完全相同，表示游戏的玩主猜到了所有的字母。

第45行：玩主没有能够猜对字母，解释程序放弃了从第 13行开始的循环。这一行输出一条安慰玩主的消息，然后退出游戏。

程序清单 4-2显示了 Hangman程序的输出的一个示例。

程序清单 4-2 Hangman程序的输出示例

```
Your Guess: t
----t-

Your Guess: s
----t-
O
Your Guess: e
----te-
O
Your Guess:
```

这个游戏对本学时中介绍的大部分概念进行了操作练习，这些概念包括直接量列表、数组、split、join、上下文和 foreach循环等。你可以使用各种各样的方法来实现这个小型 Hangman游戏程序，不过希望你能够掌握数组具备的一些基本功能。

## 4.5 课时小结

数组和列表是 Perl的集合变量。你可以使用它们来存放数量不受限制的标量，并且既可以对它们进行整体操作，也可以将它们作为单个元素来进行操作。Perl提供了许多使用非常方便的机制，可以用来对数组进行拷贝、排序、组合，以及在标量与数组的数据之间来回进行转换。另外，Perl的许多运算符和函数对它们所在的上下文是非常敏感的，它们的运行特性将根据它们是在标量上下文中还是列表上下文中而各有不同。

## 4.6 课外作业

### 4.6.1 专家答疑

问题：你能告诉我一种在数组元素中快速找到特定字符串的方法吗？

解答：迭代运行该数组，并查看每个元素，这是进行这项操作的常用方法。如果你经常发现你正在搜索一个数组，以了解某个元素是否在该数组中，那么不要首先将数据存储在一个数组中。随机访问元素的一种更加有效的结构是哈希结构，这将在第 7学时中介绍。

问题：如何消除数组中的重复元素？如何计算数组中的各个元素的数目？如何来了解两个数组是包含相同的还是不同的元素？

解答：这几个问题的答案都是一样的，那就是使用哈希结构。使用哈希结构，你就能够非常迅速而有效地对数组进行某些有意思的操作。所有这些问题将在第 7学时中回答。

### 4.6.2 思考题

1) 如果要将 \$a和\$b这两个标量变量中包含的值进行交换，哪个方法最有效？

- \$a=\$b ;
- ( \$a , \$b ) = ( \$b , \$a ) ;
- \$c=\$a ; \$a=\$b ; \$b=\$c ;



2) 语句 `$a=scalar ( @array )` ; 将什么赋值给变量 `$a` ?

- a. `@array` 中的元素的数量;
- b. `@array` 的最后一个元素的索引;
- c. 该语句无效。

#### 4.6.3 解答

1) 答案是 b。第一个选择显然不能成立, 因为 `$a` 中包含的值将被撤消。c 这个选择虽然回答了这个问题, 但是它要求数据交换时用第三个变量来存储数据。b 这个选择能够正确地进行数据的交换, 它不使用额外的变量, 并且是个非常清楚的代码。

2) 答案是 a。使用标量上下文中的数组能够返回数组中的元素数量。 `$#array` 将返回数组的最后一个索引。在这个例子中使用 `scalar ( )` 是不必要的, 在赋值运算符的左边有一个标量, 就足以将 `@array` 放入一个标量上下文中了。

#### 4.6.4 实习

- 修改 Hangman 游戏程序, 以使用竖向位置来输出 hangman。