# 第7学时　哈 希 结 构

哈希是Perl中的第三种基本数据类型。你学习的第一种数据类型是标量，它是一种简单的数据类型，用于存放一个数据（任何一个大小任意的数据，但是只能存放一个数据）。第二种数据类型是数组，它是标量的集合。数组可以根据你的需要存放任意多个标量，但是，如果要在数组中搜索你需要的标量，通常必须顺序访问该数组，直到找到你需要的标量。

哈希是另一种集合型数据类型。与数组一样，哈希包含了许多个标量。数组与哈希的差别是：哈希是按照名字来访问它们的标量的，而不是像数组那样使用数字标号进行访问。哈希元素包含两个部分，即一个关键字和一个值。关键字用于标识哈希的每个元素，而值则是与该关键字相关的数据。这种关系称为关键字值对。

许多应用程序非常适合于这种类型的数据结构。例如，如果想存放某个州的持照驾驶员的信息，那么就可以使用这些驾驶员的执照号码作为关键字来存放执照信息。这些号码是独一无二的（每个驾驶员只有一个号码）。与每个号码相关的数据就是驾驶员的信息（执照类型、地址和年龄等）。每个驾驶员的执照代表哈希结构中的一个元素，其号码和信息就构成了关键字值对的关系。具备哈希性质的其他数据结构有库存零件号、医院病历、电话付款记录、磁盘文件系统、音乐光盘收藏、 Rolodwx信息、国会图书馆、ISBN号码和其他许多数据结构。

Perl中的哈希结构可以根据你的需要包含任意多个元素，至少可以包含系统内存允许存放的最大数量的元素。当将元素添加到该哈希结构或者从哈希结构中删除元素时，哈希结构就会改变其大小。访问哈希结构中的各个元素是非常快的，并且不会因为哈希结构变大而大幅度降低访问速度。这意味着不管哈希结构拥有 10个元素还是10万个元素，Perl都能够得心应手并迅速地处理哈希结构。哈希结构的关键字的长度可以根据需要而定（它们只是标量而已），而哈希结构的数据部分的长度也可以根据需要来确定。

传统上，在Perl和其他语言中，哈希结构称为关联性数组。这是个冗长的术语，用于说明关键字是与值相关的。由于Perl程序员不喜欢冗长的词语，因此关联性数组现在简称为哈希结构。

在Perl中，哈希变量是以百分比符号（%）来标识的，它们与数组和标量不使用相同的名字。例如，你可以拥有一个名字叫 %a的哈希变量，也可以有一个名字叫 @a的数组，还可以有一个名字叫$a的标量。这些名字指的是3个互不相关的变量。

在本学时中，你将要学习如何进行下面的操作：

- 创建哈希结构。
- 将元素插入哈希结构和从哈希结构中删除元素。
- 使用哈希结构对数组进行操作。

## 7.1　将数据填入哈希结构

若要创建哈希元素，只需要将值赋予这些元素即可，这与创建数组的元素很相似。例如，可以使用类似下面的代码来创建各个哈希元素：

```
$Authors{'Dune'}='Frank Herbert';
```

在这个例子中，将 %Authors赋予哈希结构。该元素的关键字是单词 Dune，数据是名字 Frank Herbert。这个赋值操作在哈希中创建了 Dune与Frank Herbert之间的一种关系。$Authors{‘ Dune ’}可以像任何其他标量那样来进行处理，它可以被传递给函数，被操作员修改，可以输出，也可以重新赋值。当修改一个哈希元素时，请始终记住，你是修改存放在哈希元素中的值，不是修改哈希本身的值。

为什么这个例子使用的是 $Authors{}，而不是%Authors{}呢？与数组一样，当哈希结构作为一个整体来展示时，它们的变量名的前面有它自己的标记（ %)。当你访问哈希结构的单个元素，即一个标量值时，要在变量名的前面加上一个美元符号（ $)，表示它引用的是单个值，同时使用花括号来指明该值。对于 Perl来说，$Authors{‘ Dune ’}代表单个标量值，在这个例子中，代表Frank Herbert。

只有一个关键字的哈希结构并不特别有用。若要将若干个值放入一个哈希结构，可以使用一系列的赋值语句，如下面的代码所示：

```
$food{'apple'}='fruit';
$food{'pear'}='fruit';
$food{'carrot'}='vegetable';
```

若要使这个代码变得短一些，可以用一个列表对该哈希结构进行初始化。该列表应该包含成对的关键字与值，如下所示：

```
%food=('apple', 'fruit', 'pear', 'fruit', 'carrot', 'vegetable');
```

这个例子看上去与第 4学时介绍的数组初始化的例子有些相似。实际上，当你学到本学时后面部分的内容时，就会知道哈希结构可以在许多上下文中作为一种特殊类型的数组来处理。

当你对哈希结构进行初始化时，要想跟踪大型列表中的哪些项目是关键字，哪些项目是值，这是很容易搞混的。 Perl有一个特殊的运算符，称为逗号箭头运算符，即 =>。使用=>运算符，同时利用Perl忽略白空间的特性，就能够编写下面这样的哈希结构的初始化代码：

```
%food=( 'apple' => 'fruit',
     'pear'   => 'fruit',
     'carrot' => 'vegetable',
    );
```

可以使用两个辅助的快捷方式来进行哈希结构的初始化 。=>运算符的左边将是个简单的字符串，不需要用引号括起来。另外，花括号中的单个单词的哈希关键字会自动加上引号。因此，前面显示的初始化代码将变成下面的形式：

```
$Books{Dune}='Frank Herbert';
%food=( apple => 'fruit',  pear => 'fruit',  carrot => 'vegetable' );
```

逗号箭头运算符之所以叫做这个名字，原因是它的作用类似于逗号（当它用于分隔列表的项目时），并且它看上去像个箭头。

## 7.2 从哈希结构中取出数据

若要从哈希结构中取出单个元素，只需要使用一个 $、哈希结构的名字和你想要检索的关键字。请看下面这个例子：

```
%Movies=( 'The Shining' => 'Kubrick', 'Ten Commandments' => 'DeMille',
        Goonies=> 'Spielberg');
print $Movies{'The Shining'};
```

这些代码行用于输出哈希结构%Movies中的元素The Shining。这个例子将输出Kubrick。

有时查看哈希结构中的所有元素是非常有用的。如果哈希结构中的所有关键字都是已知的，那么你可以像上面显示的那样按照关键字来访问它们。但是，大多数情况下，按照名字来访问每个关键字很不方便。你可能不一定知道所有关键字的名字，也可能关键字的数量太大，无法一一列举。

可以使用keys函数来检索作为列表返回的哈希结构的所有关键字，然后可以查看该列表，找出哈希结构的所有元素。在哈希结构的内部，它的关键字并不按照特定的顺序进行存放，keys函数返回的关键字也不使用特定的顺序。若要输出该哈希结构中的所有电影名字，可以使用下面的代码：

```
foreach $film (keys %Movies) {
    print "$film\n";
}
```

这里，$film使用keys %Movies返回的列表的每个元素的值。如果除了电影的名字外，还想输出所有导演的名字，那么可以输入下面的代码：

```
foreach $film (keys %Movies) {
    print "$film was directed by $Movies{$film}.\n";
}
```

这个代码段输出的结果如下：

```
Ten Commandments was directed by DeMille.
The Shining was directed by Kubrick.
Goonies was directed by Spielberg.
```

由于$film包含一个哈希关键字的值，因此 $Movies{$film}将检索该关键字代表的哈希结构的元素值。可以将它们同时输出，以便观察哈希结构中的关键字与值之间的关系。(请记住，你的输出可能以不同的顺序出现，因为keys函数返回的关键字不是按照特定的顺序排列的。)

Perl还提供了另一个函数values，用于检索哈希结构中存放的所有值。如果仅仅检索值，通常是没有什么用处的，因为你不知道哪个关键字与哪个值相关联。返回的哈希结构的值的顺序与keys函数返回的关键字的顺序是相同的。现在请观察下面这个例子：

```
@Directors=values %Movies;
@Films=keys %Movies;
```

在这个例子中，@Directors和@Films的每个下标都包含了一个对来自 %Movies的相同关键字值对的引用，包含在$Directors[0]中的导演名字对应于存放在$Films[0]中的电影名字，等等。

有时，需要按值而不是按关键字从哈希结构中检索各个元素。按值来检索元素的最好方法是对哈希结构进行切换，也就是说，所有关键字变成值，所有值变成新哈希结构的关键字。下面就是一个例子：

```
%Movies=( 'The Shining' => 'Kubrick', 'Ten Commandments' => 'DeMille',
        Goonies=> 'Spielberg');
%ByDirector=reverse %Movies;
```

这是什么呢？当你对哈希结构使用 reverse函数时，Perl就将哈希结构转换成一个简单的列表，也许类似于下面这个列表：

```
('The Shining', 'Kubrick', 'Ten Commandments', 'DeMille', 'Goonies',
'Spielberg')
```

然后Perl对该列表中的元素顺序进行倒序，得到下面这个输出：

```
('Spielberg', 'Goonies', 'DeMille', 'Ten Commandments', 'Kubrick', 'The
Shining')
```

请注意，现在所有的关键字值对的顺序都倒了过来（值放在了前面）。当你将这个列表赋予%ByDirector时，产生的哈希结构将与原始哈希结构相同，只不过现在所有的关键字变成了值，而所有的值则变成了关键字。不过你应该知道，如果由于某个原因你的哈希结构拥有相重复的值，如果该值（将要变成关键字）不是唯一的，那么你得到的哈希结构拥有的元素将比原先要少。由于在新的哈希结构中，重复的值会发生冲突，因此老的关键字将被新的关键字代替。

## 7.3 列表与哈希结构

当我们介绍如何对哈希结构进行初始化时，曾经提到哈希结构与数组之间有着一定的相关性。每当哈希结构用于列表环境中时， Perl会将哈希结构重新变为由关键字和值组成的普通列表。该列表可以被赋予数组，这与其他任何列表的情况是一样的，如下所示：

```
%Movies=( 'The Shining' => 'Kubrick', 'Ten Commandments' => 'DeMille',
        Goonies=> 'Spielberg');
@Data=%Movies;
```

这时， @Data是个包含6个元素的数组，偶数元素（包含 0的元素）是导演的名字，奇数元素是电影名字。可以对 @Data进行任何通常的数组操作，然后将数组赋予 %Movies，如下所示：

```
%Movies=@Data;
```

Perl显然是以随机顺序来存放哈希关键字的，这个随机顺序只对Perl 有用。Perl并不设法记住放入哈希结构中的关键字的顺序，当检索关键字时，也不按照任何特定的顺序来放置它们。如果要按照某个顺序来显示它们，那么就必须对它们进行排序（参见本学时后面部分中的“ 用哈希结构进行的有用操作 ”这一节的内容），否则你应该记住它们插入时的顺序（参见本学时结尾处的“ 专家答疑 ”的内容）。

就其他方面来说，数组与哈希结构是相似的。若要拷贝一个哈希结构，只需要像下面这样将这个哈希结构赋予另一个哈希结构即可：

```
%New_Hash=%Old_Hash;
```

当你将%Old_Hash置于哈希初始化代码的右边时（ Perl通常希望右边是个列表或数组），Perl便将哈希结构转换成一个列表。然后该列表用于对 %New_Hash进行初始化。同样，可以像处理列表那样，将几个哈希结构组合起来并对它进行操作，如你下面看到的那样：

```
%Both=(%First, %Second);
%Additional=(%Both, key1=> 'value1', key2 => 'value2');
```

上面代码中的第一行将两个哈希结构 %First和%Second组合成第三个哈希结构 %Both。对于这个例子，你应该记住的是，如果 %First的有些关键字也出现在 %Second中，那么第二次出现的关键字值对就取代 %Both中的第一个关键字值对。在第二行代码中， %Both显示为一个放在括号中的关键字值对的列表，另外两个关键字值对也放在括号中。然后整个列表用于对%Additional进行初始化。

## 7.4 关于哈希结构的补充说明

如果你刚刚开始学习Perl，那么对哈希结构进行某些操作时可能会遇到困难。由于哈希结构的特殊性质，有两个常用操作需要一些专门的函数，而这些函数对于标量和数组来说是不必要的。

### 7.4.1 测试哈希结构中的关键字

若要测试哈希结构中是否存在某个关键字，需要使用下面的代码句法：

```
if ( $Hash{keyval} ) {        # WRONG, in this case
    :
}
```

由于几方面的原因，这个代码是不能满足需要的。首先，这个代码段并不是用来测定keyval是否是哈希结构中的一个关键字，它实际上是测试哈希结构中的 keyval关键字的值。如果像下面这样定义了该关键字，那么它能否达到上面的测试要求呢？

```
if ( defined $Hash{keyval} ) {    # WRONG, again in this case
    :
}
```

同样，这个代码也是不行的。这个代码段仍然只是测试与关键字 keyval相关的数据，而不是测试是否存在该关键字。 undef是个完全有效的值，用于与哈希关键字相关联，如下面的代码所示：

```
$Hash{keyval}=undef;
```

这个已经定义的测试将返回假，因为它们并没有测试哈希结构中是否存在某个关键字，它测试的是与关键字相关的数据。那么正确的方法应该是什么呢？ Perl有一个专门用于这个目的的函数，称为exists。下面显示的exists函数可以用于测试哈希结构中是否存在哈希关键字，如果存在，便返回真，否则返回假：

```
if ( exists $Hash{keyval} ) {    # RIGHT!
    :
}
```

### 7.4.2 从哈希结构中删除关键字

你可以进行的另一个操作是从哈希结构中删除关键字。正如你在前面看到的那样，仅仅将哈希元素设置为undef是不行的。若要删除单个哈希关键字，可以使用 delete函数，如下所示：

```
delete $Hash{keyval};
```

若要从哈希结构中删除所有关键字和值，只需要将哈希结构初始化为一个空的列表即可，如下所示：

```
%Hash=( );
```

## 7.5 用哈希结构进行的有用操作

由于许多方面的原因，在Perl中使用哈希结构，其目的不仅仅是为了按关键字来存储记录，供以后检索。使用哈希结构的优点是可以迅速访问各个关键字，并且哈希结构中的所有关键字都是惟一的。由于具备这些特性，因此哈希结构对于数据操作是非常有用的。毫不奇怪，由于数组和哈希结构非常相似，因此你用哈希结构进行的许多有趣的操作属于数组操作。

### 7.5.1 确定频率分布

在第6学时中，你学习了如何取出一行文本，然后将它分割成单词。请观察下面这个代码段：

```
while(<>) {
   while ( /(\w[\w-]*)/g  ) {    # Iterate over words, setting $1 to each.
      $Words{$1}++;
   }
}
```

第一行代码每次读取一行标准输入，为每一行设置 $_。

然后，第二行的 while（）循环对 $_中的每个单词进行迭代操作。让我们回顾一下第 6学时介绍的内容，在标量上下文中使用带有修饰符 g的模式匹配运算符（//），将返回匹配的每个模式，直到不再剩下匹配的模式为止。寻找的模式是个单词字符 \w，后随0个或多个单词字符或者连字符[\w-]*。在这个例子中使用了括号，以便记住特殊变量 $1中匹配的字符串。

下一行虽然很短，但是它是该代码段中令人感兴趣的部分。 $1依次设置为第二行上的模式匹配的每个单词。该单词用作哈希结构 %Words的关键字，该关键字值对的值开始时没有定义。通过对它进行递增，在第一次看到该单词时， Perl将该值设置为1。第二次看到一个单词时，哈希结构 %Words中已经存在关键字（该单词），同时它的值从 1递增为2。这个过程将继续进行下去，直到不再有遗漏的输入为止。

当你完成操作后，哈希结构 %Words将包含读入的单词的频率分布情况。若要查看该频率分布，可以使用下面这个代码：

```
foreach( keys %Words ) {
        print "$_ $Words{$_}\n";
}
```

### 7.5.2 在数组中寻找惟一的元素

上面这个代码中展示的方法也可以用来寻找数组中只出现一次的元素。假设已经将输入的全部单词放入一个数组而不是哈希结构，同时没有专门采取措施来保证在将一个单词放入列表之前，该列表中还没有这个单词。在这种情况下，列表中可能存在许多重复的单词。

如果输入的文本的开始行是 One Fish，Two Fish，那么该列表看上去如下所示：

```
@fishwords=('one', 'fish', 'two', 'fish', 'red', 'fish', 'blue', 'fish');
```

如果你被赋予这个单词列表（在 @fishwords中），同时你只需要该列表的独一无二的元素，那么使用哈希结构就非常适合你的需要，如程序清单 7-1所示：

**程序清单7-1 寻找数组中的惟一的元素**

```
1:   %seen=();
2:   foreach(@fishwords) {
3:      $seen{$_}=1;
4:   }
5:   @uniquewords=keys %seen;
```

第1行：用于对临时哈希结构 %seen进行初始化，该哈希结构用于存放你的所有单词。

第2行：对单词列表进行迭代操作，依次将 $_设置为每个单词。

第3行：用于创建哈希结构 %seen中的关键字，以 $_中的该单词作为关键字，并为该数据创建一个名义上的值。

第5行 ：只是从哈希结构中取出所有关键字，并将它们存放在 @uniquewords中。哈希结构中的任何重复单词（例如fish）将互相改写，然后只以一个关键字出现。

### 7.5.3 寻找两个数组之间的交汇部分和不同部分

对数组经常要进行的一项操作是寻找两个数组之间的交汇部分（即它们的重叠部分）和两个数组之间的不同部分（它们不重叠的部分）。在这个例子中，你有两个列表，一个是包含电影明星的列表，另一个是包含政治家的列表。你的任务是找出是电影明星的所有政治家。下面是你的两个（非常不完整的）数组：

```
@stars=('R. Reagan', 'C. Eastwood', 'M. Jackson', 'Cher', 'S. Bono');
@pols  = ('N. Gingrich', 'S. Thurmon', 'R. Reagan',
                    'S. Bono', 'C. Eastwood', 'M. Thatcher');
```

程序清单7-2显示了寻找交汇部分的代码。

**程序清单7-2 寻找两个数组的交汇部分**

```
1:   %seen=();
2:   foreach (@stars) {
3:        $seen{$_}=1;
4:   }
5:   @intersection=grep($seen{$_}, @pols);
```

第1行：用于对哈希结构 %seen进行初始化。这个临时哈希结构用于存放所有电影明星的名字。

第2行：对电影明星的列表进行迭代操作，依次将 $_设置为每个名字。

第3行：用电影明星的名字填入哈希结构 %seen的各个关键字，并将值设置为 1，这个值可以是你想要的任何真值。

第5行：这一行看起来比它实际上更复杂一些。 @pols中的Grep函数对政治家的列表进行迭代操作，依次将$_设置给每个政治家。然后，在哈希结构 %seen中寻找该名字。如果该名字返回真，那么它就位于哈希结构中，表达式 $seen{$_}计算的结果为真。如果该表达式计算的结果是真，那么grep返回$_的值，然后该值被放入@intersection中。这个过程将重复进行，直到@pols被grep全部查看完毕。当该代码段运行结束时， @intersection便包含既是@stars又是@pols的所有成员的名字。

用于寻找两个数组之间的不同部分（即在一个数组中存在，而在另一个数组中不存在的那些元素）的代码与上面这个代码几乎是相同的，可以使用程序清单 7-3来查找不是电影明星的所有政治家。

**程序清单7-3 寻找两个数组之间的不同部分**

```
1:   %seen=();
2:   foreach (@stars) {
3:        $seen{$_}=1;
4:   }
5:   @difference=grep(! $seen{$_}, @pols);
```

惟一有变化的一行是第 5行。它仍然用于查找哈希结构 %seen中的每个政治家的名字，但是，现在如果它找到了政治家的名字，则返回假。相反，如果没有找到政治家的名字，则返回真。出现在哈希结构 %seen中的所有政治家的名字并不返回给 @difference。如果你想要找到不是政治家的所有电影明星，使用的代码几乎完全一样，但是必须将 @stars切换成@pols。

### 7.5.4 对哈希结构进行排序

许多情况下，仅仅按照默认顺序来检索哈希结构中的关键字是不够的（默认顺序是非常随机的）。这是许多情况中的一种。可以用两种方法来输出你创建的频率分布，一种是按单词的字母顺序，另一种是按频率的顺序。由于 keys函数能够返回一个简单的列表，因此可以像下面这样使用sort函数对该列表进行排序：

```
foreach( sort keys %Words ) {
    print "$_ $Words{$_}\n";
}
```

按频率给列表排序并没有太大的差别。第 4学时中我们讲过，按照默认设置， sort函数只是用ASCII顺序给既定的列表排序。但是，如果需要进行比较复杂的排序，可以在代码块中调用sort函数，以便设定排序顺序。下面这个代码显示了按照值该哈希结构进行排序的情况：

```
foreach ( sort  { $Words{$a} <=> $Words{$b} } keys %Words ) {
    print "$_ $Words{$_}\n";
}
```

也许你还记得，与 sort一道使用的BLOCK被sort函数反复调用， $a和$b被设置为sort函数需要进行排序的每一对值。在这种情况下， $a和$b被设置为哈希结构 %words中的各个不同关键字。该代码不是直接比较 $a与$b，而是查看哈希结构 %Words中的这些关键字的值，然后对它们进行比较。

## 7.6 练习：用Perl创建一个简单的客户数据库

当你打电话给客户服务中心，并且最后接通对方的电话时，对方问你的第一个问题是你的电话号码是什么。确实，每次几乎都是这样。有时，客户服务代表想要你的客户号码，甚至你的社会保险号。对方需要的是计算机能够用来识别你的一个惟一标志。这些号码可以作为在数据库中检索关于你的信息时使用的关键字，这就像 Perl的哈希结构，是不是？

在这个练习中，你将要搜索一个客户数据库。这个程序假设数据库已经存在，并且尚无更新数据库的任何手段。在这个数据库中，你将允许用户搜索一个或两个不同的域。

在开始做这个练习时，需要一些数据。请打开文本编辑器，键入文本（或类似的某些数据），并将它保存为customers.tet。不必担心列与列之间的空格数量，也不必考虑使它们对齐，你只需要在每一列之间留一个空格。

```
Smith,John    (248)-555-9430 jsmith@aol.com
Hunter,Apryl  (810)-555-3029 april@showers.org
Stewart,Pat   (405)-555-8710 pats@starfleet.co.uk
Ching,Iris    (305)-555-0919 iching@zen.org
Doe,John      (212)-555-0912 jdoe@morgue.com
Jones,Tom     (312)-555-3321 tj2342@aol.com
Smith,John    (607)-555-0023 smith@pocahontas.com
Crosby,Dave   (405)-555-1516 cros@csny.org
Johns,Pam     (313)-555-6790 pj@sleepy.com
Jeter,Linda   (810)-555-8761 netless@earthlink.net
Garland,Judy  (305)-555-1231 ozgal@rainbow.com
```

在同一个目录中，键入程序清单 7-5中的短程序，并将它保存为 customer。务必按照第1学时中的说明使该程序成为可执行程序。

当完成上述操作后，键入下面这个命令行，设法运行该程序。

```
perl Customer
```

程序清单7-4显示了Customer程序的输出。

**程序清单7-4 Customer程序的示例输出**

```
1:  Type 'q' to exit
2:  Number? <return>
3:  E-Mail? cros@csny.org
4:  Customer: Crosby, Dave    (405)-555-1516  cros@csny.org
5:
6:  Number? (305)-555-0919
7:  Customer: Ching,Iris   (305)-555-0919  iching@zen.org
8:
9:  Number? q
10:
11: All done.
```

**程序清单7-5 Customer程序的完整清单**

```
1:  #!/usr/bin/perl -w
2:
3:  open(PH, "customers.txt") or die "Cannot open customers.txt: $!\n";
4:  while(<PH>) {
5:          chomp;
6:          ($number, $email)=(split(/\s+/, $_))[1,2];
7:          $Phone{$number}=$_;
8:          $Email{$email}=$_;
9:  }
10:  close(PH);
11:
12:  print "Type 'q' to exit\n";
13:  while(1) {
14:          print "\nNumber? ";
15:          $number=<STDIN>;  chomp($number);
16:          $address="";
17:          if (! $number ) {
18:                  print "E-Mail? ";
19:                  $address=<STDIN>;  chomp($address);
20:          }
21:
22:          next if (! $number and ! $address);
23:          last if ($number eq 'q' or $address eq 'q');
24:
25:          if ( $number and exists $Phone{$number} ) {
26:                  print "Customer: $Phone{$number}\n";
27:                  next;
28:          }
29:
30:          if ($address and exists $Email{$address} ) {
31:                  print "Customer: $Email{$address}\n";
32:                  next;
33:          }
34:          print "Customer record not found.\n";
35:          next;
36:  }
36:  print "\nAll done.\n";
```

第1行：这一行包含到达解释程序的路径（可以修改这个路径，使之适合系统的需要）和开关-w。请始终使警告特性处于激活状态。

第3行：文件句柄PH上的customers.txt文件被打开。当然，对它的错误进行了检查，并且作了报告。

第4~5行：文件句柄PH被读取，每一行均被赋予$_。$_则使用chomp命令删除结尾处的换行符。

第6行：$_中的这一行在白空间处（\s+）被分割。spilt语句的前后加上一组括号，后面是方括号。由于你只对每一行上的电话号码和电子邮件地址感兴趣，所以从分割的语句中取出一部分返回值。这两个值被赋予$number和$email。

第7~8行：%Email用于存放客户记录，关键字是电子邮件地址。 %Phone用于存放与客户相关的电子邮件地址。

第10行：这一行用于关闭文件句柄。

第13行：该while循环包含了需要重复运行的这部分代码。语句 while（1）是个Perl的习惯用语，意思是“ 永远循环 ”。若要退出该循环，最终需要使用 last语句。

第14~15行：电话号码被读取，换行符被删除。

第17~20行：如果没有电话号码，那么这些代码行提示你输入一个电子邮件地址。

第22~23行：如果没有输入任何信息，这一行便重复运行该循环。如果输入一个 g，则该循环退出。

第25~28行：如果输入了一个号码，并且是个有效的号码，那么第26行将输出该客户记录。控制权重新传递给顶部带有 next语句的代码块。

第30~33行：输入了一个地址，并且是个有效的地址，那么输出客户记录。控制权重新传递给顶部带有next的代码块。

第34~35行：如果输入了一个地址或电话号码，但是发现它是个无效的地址或号码，那么这两行便输出一条消息，并重复运行带有 next的代码块。

这个例子展示了 Perl的几个特性。哈希结构可以用于根据关键字来迅速查找数据。由于Perl能够非常有效地实现哈希结构，即使该程序拥有哈希结构中的成千上万的记录，查询的响应时间也不会变得很长。另外，这个程序也展示了使用简单 BLOCK的程序控制流，而不是其他的控制结构（如while、do和until等）。

## 7.7 课时小结

哈希结构为程序员提供了许多非常有用的工具。除了简单的记录存储和检索工具外，哈希还提供了对数据进行转换和分析的有用机制。数组操作、记录存储和检索的公式将会给你带来许多好处。在后面的几个学时中，哈希结构将为你提供一个进一步学习 DBM文件处理、复杂数据结构的操作，以及与你的系统环境打交道等内容的途径。

## 7.8 课外作业

### 7.8.1 专家答疑

**问题：**如果需要用一个关键字来存储若干个数据（一个列表），我能否在哈希结构中存储多个数据呢？

**解答：**可以。这需要使用两个基本方法。第一个方法（也是最笨的方法）是将哈希元素

中的值的部分格式化为可以识别的东西，比如用一个用逗号分隔的列表。每当你存储哈希元素时，可以使用join函数，将该列表组合到一个标量中，每当你从哈希结构中检索一个值时，可以使用split函数，将标量重新分割成列表。这种方法比较麻烦，也容易出错。

另一种方法是使用一个引用项。使用引用项后，你就能够创建数组的哈希结构，哈希结构中的哈希结构和其他复杂的数据类型。当你掌握了它的诀窍后，使用引用项来创建复杂数据结构是非常容易的。关于这方面的内容，我们将在第 13学时中介绍。

**问题：**我应该如何按照将关键字赋予哈希结构时的顺序来保持它的顺序？

**解答：**同样你可以使用两种方法来保持它们的顺序。第一种方法比较难，你要跟踪你的插入顺序。方法是使用一个对哈希结构进行镜像的数组。当将新元素放入哈希结构时，可以使用push将相同的关键字放入一个数组。当需要查看插入顺序时，只需要使用该数组而不是keys函数。这种方法比较复杂，并且容易出错。

更好的方法是使用模块Tie::IxHash。这个模块可以根据你的需要，使keys函数按照插入顺序返回哈希关键字。在第14学时中，我们将要介绍如何使用这个模块的方法。

**问题：**你能介绍一种将哈希结构写入文件时可以使用的简便方法吗？

**解答：**当然可以。Data::Dumper或Storable之类的模块能够将哈希和数组等数据类型重新格式化为便于存放的标量值，这些标量值可以写入文本文件。这些模块还拥有一些函数，它们能够利用那些格式化的标量并重新创建你存储的原始结构。

在第15学时中，我们将要介绍一种将哈希写入文件的非常方便的方法，这就是使用 DBM文件。使用DBM文件，你可以将你的哈希结构与一个磁盘文件关联起来。当你改变哈希结构时，磁盘文件也会相应地变更。该磁盘文件能够使你的哈希结构一直保留着，只要文件保持不变。

### 7.8.2 思考题

1) 对于本学时中你编写的 Customer程序来说，尤其是如果客户列表非常长的话，为什么name是个不合适的搜索关键字？

   a. 姓和名字的组合超出了Perl的哈希结构允许的长度。
   b. 人的名字不是独一无二的关键字。
   c. 没有人想要按照名字来搜索客户数据库。

2) 相关性数组与哈希结构之间的区别是什么？

   a. 没有区别。
   b. 相关性数组用于更加正式的数据集，比如帐单记录。
   c. 在Perl中，哈希不是真正的相关性数组，因此它们拥有不同的名字。

3) 哪些种类的数据最适合哈希结构？

   a. 简单的项目列表。
   b. 普通常用数据。
   c. 关键字值对的列表。

### 7.8.3 解答

1) 答案是b。Perl的哈希结构的大小实际上是不受限制的，用户必然要求按名字进行搜索。

但是按照人的名字来搜索是不合适的，因为名字不是独一无二的。电话簿中有许多的重复名字，比如John Smith和Robert Jones等。

2) 答案是a。哈希结构与相关性数组是同一个东西。惟一的区别是哈希说起来顺口，拼写容易。

3) 答案是c。选择c是正确的，不过组织得很好的普通数据哈希结构也是可以的。

### 7.8.4 实习

- 修改Customer程序，使得它可以按照名字进行搜索。由于你不能将name用作哈希关键字，因此必须通过哈希中的值来进行搜索。
- 修改Customer程序，使得它可以按照关键字的某个部分（比如电话号码的一部分或者电子邮件地址的一部分）来进行搜索。可以使用正则表达式来搜索模式。请记住应该找到多个结果，并且返回每个结果。