1 NAME

perlreftut – Mark 的关于引用的简短指南

2 DESCRIPTION

Perl5 最重要的新特性之一就是可以管理复杂数据结构比如多维数组或者嵌套的哈 希。为了实现这一功能，Perl5 引进了一个新的特性称为“引用”，引用是管理 Perl 中复杂数据结构的关键。不幸的是，这其中有大量有意思的语法需要学习， 主体手册掌握起来有一定困难。虽然它相当完整，但是有时候也存在问题，它很难 讲明哪些是重要的，哪些不是。

幸运的是，你只需要了解主体手册的 10% 就可以得到 90% 的好处。本指南就将向 你展示这 10%。

3 谁需要复杂数据结构

一直以来 Perl4 中都存在着如何表示一个包含的列表值的哈希结构的问题。Perl4 当然也有哈希，但是它的值只能是标量，而不能是列表。

为什么需要一个包含列表的哈希呢？我们举一个简单的例子：你有一个包含城市和 国家名称的文件，如：

```
Chicago, USA
Frankfurt, Germany
Berlin, Germany
Washington, USA
Helsinki, Finland
New York, USA
```

你想产生这样一个输出，每个国家仅被提到一次，后面跟着按字母排序的该国城市 的名称

```
Finland: Helsinki.
Germany: Berlin, Frankfurt.
USA:  Chicago, New York, Washington.
```

很自然的方法就是用一个键值为国家名称的哈希来表示。与每个国家名称相关联的 是这个国家的城市列表。每次你读入一行输入，就将它分割成一个国家名称和城市 名称，在该国家已有的城市列表中查找名称或往列表中加入新的城市。当你读完输 入之后，按照平常的方式迭代遍历整个哈希，并将每个列表中的城市排序后输出。

如果哈希的取值不能是列表，你就没法实现了。在 Perl4 中，哈希的值不能是列 表，只能用字符串表示。你也许只能设法将所有城市组合成一个简单的字符串，然 后在输出的时候将字符串分割成列表，将列表排序，再转换回字符串。这种方法凌 乱而易出错，并且也很挫伤人，因为 Perl 本身有完美的列表可以解决这个问题， 只要你可以使用它。

4 解决方案在哪里？

在 Perl5 开始普及时，我们已经习惯于这样的设计：哈希的值必须是标量。这就 需要使用引用来解决这个问题。

引用是一个指向整个数组或整个哈希（或者其他任何东西）的标量。姓名是你已 经非常熟悉的引用之一。考虑一下美国总统：一副乱七八糟的皮肉。但是谈到他或 者在程序中表示他，你只需要一个简单的标量字符串“乔治·布什”。

Perl 中的引用就像数组和哈希的名字。他们是 Perl 的私有和内部名字，所以你 可以保证他们不会混淆。和“乔治·布什”不同的是，很多人都可以叫这个名字， 但是一个引用却只指向一个事物，并且你总是了解它指向什么。如果你拥有一个数 组的引用，那么你可以从中取得整个数组。如果你有一个哈希的引用，那么你也能 恢复整个哈希。但是引用仍然是一个简单、紧凑的标量值。

你不能得到一个包含数组值的哈希；哈希值只能是标量。我们坚持这个原则。但是 一个引用可以表示整个数组，并且引用是标量，所以你可以拥有一个包含哈希引用 的数组，它看起来就像是数组的哈希并且和数组的哈希一样有用。

我们待会儿再讨论城市-国家问题，在我们看一下管理引用的语法之后。

5 语法

有两种创建引用的方法，也只有两种方法可以使用它。

创建引用

创建规则一

如果你在变量前面加一个 \，那么你就得到了它的引用。

```
$aref = \@array;        # $aref 保存 @array 的引用
$href = \%hash;         # $href 保存 %hash 的引用
```

一旦引用被储存在一个像 $aref 或 $href 的变量中，你就可以像其他标量值一样 对它进行拷贝或储存。

```
$xy = $aref;            # $xy 保存 @array的引用
$p[3] = $href;          # $p[3] 保存 %hash的引用
$z = $p[3];             # $z 保存 %hash的引用
```

这些例子显示了如何从名字中创建引用。有时候你想要创建一个没有名字的数组或 哈希的引用，这个就像你可以使用 \n 这个字符串或者 80 这样的 ， 而你并没有将他们存储在任何变量中一样。

创建规则二

[ITEMS] 创建一个新的，匿名数组，并且返回该数组的引用。{ ITEMS } 创建一个新的匿名哈希，并返回这个哈希的引用。

```
$aref = [ 1, "foo", undef, 13 ];
# $aref 保存一个数组的引用

$href = { APR => 4, AUG => 8 };
# $href 保存一个哈希的引用
```

通过规则一创建的引用与通过规则二创建的引用效果完全相同：

```
# 下面这句
$aref = [ 1, 2, 3 ];

# 其实就相当于
@array = (1, 2, 3);
$aref = \@array;
```

第一行是下面两行的缩写，只不过它不创建多余的数组变量 @array。

如果你输入只输入 []，你就会得到一个新的空的匿名数组。 如果你只输入 {}，你就会得到一个新的空的匿名哈希。

使用引用

在得到引用之后你该如何使用它呢？这是一个标量，我们已经看到可以将它当作标 量存储起来并像一个标量一样重新取得它。此外还有两种使用它的方法。

用法规则一

你总是可以用花括号括起的数组引用来代替数组名称。例如，@{$aref} 可以用 来代替 @array。

这里有一些例子

数组：

```
@a            @{$aref}              一个数组
reverse @a     reverse @{$aref}       将数组翻转
$a[3]         ${$aref}[3]           数组的一个元素
$a[3] = 17;    ${$aref}[3] = 17       给一个元素赋值
```

每一行中的表达式都做同样的事情。左边的版本操作数组 @a。右边的版本操作 $aref 指向的数组。对于同一个数组来说，两种方式的效果是一样的。

使用哈希的引用也完全一样：

```
%h            %{$href}             一个哈希
keys %h        keys %{$href}        获取哈希的键值列表
$h{'red'}      ${$href}{'red'}      访问哈希的一个元素
$h{'red'} = 17  ${$href}{'red'} = 17  给哈希的元素赋值
```

不论你想要用引用来做什么，使用用法规则一都可以做到。你只需要象往常一样使 普通的数组或者哈希来写程序，然后再把数组或者哈希的名字换成 $reference 就行了。“我如何通过一个引用来遍历一个数组？”，很简单，如果你想遍历一个 数组，那你得这么写：

```
for my $element (@array) {
  ...
}
```

然后，把数组名 array 换成引用就可以了：

```
for my $element (@{$aref}) {
  ...
}
```

“我怎么样通过引用来打印出哈希的内容？”，你可以先写一个打印哈希的程序：

```
for my $key (keys %hash) {
  print "$key => $hash{$key}\n";
}
```

然后把哈希的名字换成引用就可以了：

```
for my $key (keys %{$href}) {
  print "$key => ${$href}{$key}\n";
}
```

用法规则二

有了用法规则一在大多数时候就已经够用了，因为它告诉了你一个以不变应万变 的引用用法。但是大多数时候我们只需要访问数组或者哈希的一个元素，这时候用 法规则一就显得太繁琐了。所以下面介绍一种简写方式：

考虑一下如何通过一个引用去访问数组的第 4 个元素？得这样：${$aref}[3]，可 是这种写法很难读懂，那你可以写成 $aref->[3] 来代替。

${$href}{red} 也很难懂，那我们可以用 $href->{red} 代替。

如果 $aref 保存了一个指向数组的引用，那么 $aref->[3] 的意思就是 取这个数组的第 4 个元素。千万不好和 $aref[3] 混淆，后者表示的是 @aref 这 个数组的第四个元素 ^_^。$ref 和 @aref 完全无关。

同样地，$href->{'red'} 表示哈希引用 $href 的一个元素。同样地，你 不能把它和 $href{'red'} 混淆，因为后者访问的是 %href 这个哈希的内容。忘记 书写 -> 将导致你的程序出现一些莫名其妙的问题。因为你访问的数组和哈 希其实都不是你想要访问的。

一个例子

让我们来看一个简短的例子。

首先，记住 [1，2，3] 将创建一个包含有 (1，2，3) 三个数字的匿名数组， 并且返回指向这个匿名数组的引用。

现在看看这个：

```
@a = ( [1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]
     );
```

@a 是一个包含有三个元素的数组，它的每个元素都是指向别的数组的引用。

$a[1] 是这些引用其中的一个，它指向一个数组，这个数组包含 (4，5，6)， 并且，因为它是一个数组的引用，所以根据用法规则二，我们可以用 $a[1]->[2] 来获取这个数组的第三个元素。所以 $a[1]->[2] 就是 6。

同样地，$a[0]->[1] 就是 2，这就好像我们有了一个二维数组一样，你可以 通过 $a[ ]->[ ] 来获取或者设置任意行任意列的元素的值。

这看起来仍然有些繁琐，所以下面再介绍一种更加简便的写法：

箭头规则

在两个下标之间，箭头可以省略。

于是，$a[1]->[2] 就可以写作 $a[1][2]，它们是同一个意思。 $a[0]->[1] = 23, 就可以写作 $a[0][1] = 23，它们也是同一个意思。

现在看上去真的和二维数组很像了！

箭头和省略箭头这两个规则非常重要。如果没有了它们，我们就必须把 $a[1][2] 写成是 ${$a[1]}[2]。三维数组 $x[2][3][5] 就必须写成是 ${${$x[2]}[3]}[5]，哦，那简直是一场恶梦。

6 解决方案

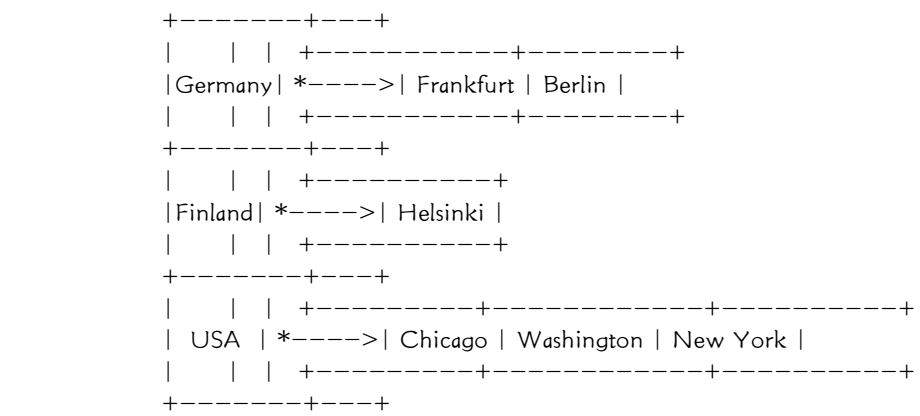
现在就可以回答开始提出的问题了。下面的程序就可以对国家・城市名重新整理：

```
1  my %table;
```

```
 2  while (<>) {
 3   chomp;
 4    my ($city, $country) = split /, /;
 5    $table{$country} = [] unless exists $table{$country};
 6    push @{$table{$country}}, $city;
 7  }

 8  foreach $country (sort keys %table) {
 9    print "$country: ";
10    my @cities = @{$table{$country}};
11    print join ', ', sort @cities;
12    print ".\n";
13  }
```

这个程序分两部分：第 2–7 行读入输入数据并且创建一个数据结构，然后第 8–13 行分析这些数据并产生报表。我们设计了一张哈希表，%table，它的每个键是 一个国家名，对应的值则是一个指向数组的引用，数组中存放城市名。整个数据结 构看起来就像是这样：



我们先看看输出部分。假如我们已经拥有了这样的一个数据结构，那么我们该怎样 打印它呢？

```
 8  foreach $country (sort keys %table) {
 9    print "$country: ";
10    my @cities = @{$table{$country}};
11    print join ', ', sort @cities;
12    print ".\n";
13  }
```

%table 是一个普通的哈希表，我们取出它所有的键，并且进行排序，然后遍历 这些键，这些和往常没有什么不同。这里仅仅只在第 10 行使用了引用。 $table{$country} 从哈希表中检索 $country 对应的值，这个值就是一个指 向这个国家城市列表的一个引用。根据用法规则一，我们可以用 @{$table{$country}} 还原这个数组。所以第 10 行就好像是：

```
@cities = @array;
```

唯一不同的地方是数组名 array 被 $table{$country} 代替了。@ 告诉 Perl 得到整个数组。然后我们对这个数组用 sort 排序、用 join 连接，最后打印它。

第 2–7 行负责构造数据结构。我们再来看看：

```
2   while (<>) {
3    chomp;
4     my ($city, $country) = split /, /;
5     $table{$country} = [] unless exists $table{$country};
6     push @{$table{$country}}, $city;
7   }
```

第 2–4 行得到城市名和国家名，第 5 行检查国家名对应的键值是否已经在哈希表 中存在。如果不存在，那么就用 [] 创建一个新的空匿名数组，并且把引用挂到 哈希表中国家名对应的条目中。

第 6 行把城市名加入到适当的数组中去。现在 `$table{$country}` 指向一个城市 名数组，所以第 6 行看上去就像是：

```
push @array, $city;
```

唯一不同的地方就是数组名 `array` 被 `{$table{$country}}` 代替了。`push` 运算符把城市名添加到引用指向的数组的末尾。

其实第 5 行完全是不必要的。也就是说，也可以写成这样：

```
2   while (<>) {
3    chomp;
4     my ($city, $country) = split /, /;
5   ####  $table{$country} = [] unless exists $table{$country};
6     push @{$table{$country}}, $city;
7   }
```

如国 `%table` 中已经存在了 `$country` 对应的条目，那么有没有第 5 行并没 有什么分别。第 6 行将定位到 `$table{$country}` 并且把 `$city` push 进去。 但是如果哈希表中并没有 `$country` 这样一个键，那么会发生什么事呢?

Perl 对这时应该发生的行为做了明确的定义。如果你想要 push `Athens` 到一个 数组，而那个数组并不存在，那么 Perl 会友好地帮你创建一个新的、空的、匿名 数组给你，并且把它挂在 `%table` 上，然后再把 `Athens` push 进去。这叫做 `autovivification'--bringing things to life automatically. (flw 注：这句 实在不知该如何翻译) 。Perl 看见那个键不在哈希中，于是就自动创建一个新的 哈希条目；Perl 看见你想要把这个条目作为一个数组使用，于是它又自动地创建 一个新的空数组并且自动把引用存储到哈希中。 And as usual, Perl made the array one element longer to hold the new city name.

7 临了

我曾经答应过用 10% 的内容给你 90% 的好处，这意味着，还有 90% 的内容我没 讲 ^_^。不过现在你已经对主要的东西有了一个大概的印象，因此你可以很容易地 看懂 perlref，来品尝 100% 的内容。

阅读 perlref 时应该注意以下几点：

- 你可以给任何东西创建引用，例如标量、函数、甚至是别的引用。
- 在用法规则一中，如果花括号中只有一个简单的引用的话，那么花括号就可以 省略。例如，`@$aref` 就相当于 `@{$aref}`，`$$aref[1]` 就相当于 `${$aref}[1]`。 If you're just starting out, you may want to adopt the habit of always including the curly brackets.
- 还有一点，这样并不能复制它们指向的数组：

```
$aref2 = $aref1;
```

这样你只能得到两个指向相同的数组的引用。如果你修改了 `$aref1->[23]` 那么 `$aref2->[23]` 会做同样的改变。

要想完全复制数组，你得这样：

```
$aref2 = [@{$aref1}];
```

符号 `[...]` 创建一个新的匿名数组，并且把引用赋值给 `$aref2`，这个新 数组已经用 `$aref1` 的内容初始化过了。

同样地，要想复制匿名哈希，你得这样：

```
$href2 = {%{$href1}};
```

- 因为引用本来就是一个标量，因此从表面上是无法知道一个标量是否就是一个引 用，这时你可以使用 `ref` 函数。如果传递给 ref 的参数是一个引用，它将返 回真值，事实上，如果是一个哈希引用，那么 ref 会返回 `HASH`，如果是一个 数组引用，ref 则返回 `ARRAY`
- 如果你把一个引用当作一个字符串来用的话，你将得到类似于这样的一个结果：

  ```
  ARRAY(0x80f5dec)  or  HASH(0x826afc0)
  ```

  如果你曾经看到过这样的字符串，那么说明你以前曾经打印过一个引用 ^_^

  这个特性的一个副作用就是你可以使用 `eq` 运算符来比较两个引用是否指向 同一个东西。但是建议你不要用 `eq`，而是用 `==`，因为后面的这个更快 一些。
- 你也可以把一个字符串当作一个引用来用。如果你把 `"foo"` 当作一个数组引 用，那么相当于是在访问 `@foo`。这个技术在术语里叫做 软引用 或者 符号引用。`use strict 'refs'` 声明可以关掉这个特性，因为它经常给人 们带来各式各样的烦恼。

你可能更喜欢看 perllol 而不是 perlref。因为它讨论“列表的列表” 以及多维数组的技术。看完它之后，你还可以看 perldsc。这是一个“数据 结构烹调指南》，它可以告诉你如何打印数组的哈希、哈希的数组等复杂的数据 结构。

8 摘要

任何人都需要复杂的数据结构，Perl 中通过引用来得到解决。关于引用有四条重 要规则：两条和创建引用有关的；两条和使用引用有关的。如果你了解了这些规 则，那么你就可以用引用来处理大多数重要的事情。

9 参与人名单

作者: Mark Jason Dominus, Plover Systems (`mjd-perl-ref+@plover.com`)

这篇文章最早出自 The Perl Journal (http://www.tpj.com/) 卷 3, #2. 允许再版。

本文的原是标题是 Understand References Today (现在就了解引用) 。

翻译

本文的开始部分（从开始截至到用法规则一）由 rogerz 翻译，其余部分由 flw（flw@cpan.org）翻译，翻译成果首次出现在 中国 Perl 协会（ http://www.perlchina.org）的协作开发平台上。

发布条件

译者声明

PerlChina.org 本着“在国内推广 Perl” 的目的，组织人员翻译本文。 读者可以在遵守原作者许可协议、尊重原作者及译作者劳动成果的前提下，任意 发布或修改本文。

如果你对本文有任何意见，欢迎来信指教。译作者欢迎此类行为但并不为此负责。