



神奇的perl-最佳PERL入门读物

作者: deepfuture <http://deepfuture.javaeye.com>

神奇的Perl系列为刘兴原创

(<http://deepfuture.javaeye.com/>)

版权所有 未经作者书面授权请勿转载

更多章节请看 [HTTP://DEEPFUTURE.JAVAEYE.COM/CATEGORY/99810](http://DEEPFUTURE.JAVAEYE.COM/CATEGORY/99810)

目 录

1. 神奇的Perl

1.1 拥抱Perl 3

1.2 Perl处理输入输出-1 5

1.3 Perl处理输入输出-2 11

1.4 word文件批量另存为txt 15

1.5 Perl的语法-1 16

1.6 perl语法-2 24

1.7 神奇的Perl-哈希与数组(1) 38

1.8 神奇的Perl-哈希与数组(2) 42

1.9 神奇的Perl-哈希与数组(3) 47

1.10 神奇的Perl-哈希与数组(4) 50

1.11 perl处理输入和输出-3 54

1.12 神奇的perl-哈希和数组(5) 59

1.13 神奇的Perl-哈希和数组(6) 61

1.14 神奇的Perl-哈希和数组(7) 66

1.15 神奇的perl-正则表达式(1) 75

1.16 神奇的perl-正则表达式(2) 77

1.17 神奇的Perl-正则表达式(3) 81

1.18 神奇的Perl-最佳Perl入门读物 84

1.1 拥抱Perl

发表时间: 2010-03-24

声明：本神奇的Perl系列为刘兴
(<http://deepfuture.javaeye.com/>)原创，未经笔者授权，任何人和机构不能转载

拥抱Perl

Perl是什么？

Perl最初的设计者为Larry Wall，Perl借取了C、sed、awk、shell scripting以及很多其他程序语言的特性。Perl一般被称为“实用报表提取语言”（Practical Extraction and Report Language），虽然有时被称做“病态折中垃圾列表器”（Pathologically Eclectic Rubbish Lister）。

Perl是神奇的

Perl的神奇之处在于具有C语言一样的强大能力和灵活性，但却比C简单很多。

你不用先学习所有Perl的东西就可以开始写有用的程序，这对于那些急于完成任务却不得不为此编写一堆代码的人来可谓雪中送炭，从一开始，Perl就设计成可以把简单工作简单化，同时又不失去处理困难问题能力的语言，Perl既强大又好用，所以它被广泛地用于日常生活的方方面面，从宇航工程到分子生物学，从数学到语言学，从图形处理到文档处理，从数据库操作到网络管理。

对于不想为程序设计语言买单的人来说，Perl也不失一种选择，因为Perl的解释程序是开放源码的免费软件，使用Perl不必担心费用，Perl也能在绝大多数操作系统运行，可以方便地向不同操作系统迁移。

安装Perl运行环境和IDE

笔者推崇使用以下组合：

运行环境：activeperl，ActivePerl一个可以让你任意执行Perl程序的工具软件，其包含了包括有Perl for Win32、Perl for ISAPI、PerlScript、Perl Package Manager四套开发工具程序。

下载地址：<http://www.activestate.com/activeperl/>

IDE: Open Perl IDE，OpenPerlIDE是免费且开源的，也是绿色软件，使用时直接运行PerlIDE.exe文件即可。

下载地址：<http://sourceforge.net/projects/open-perl-ide/files/>

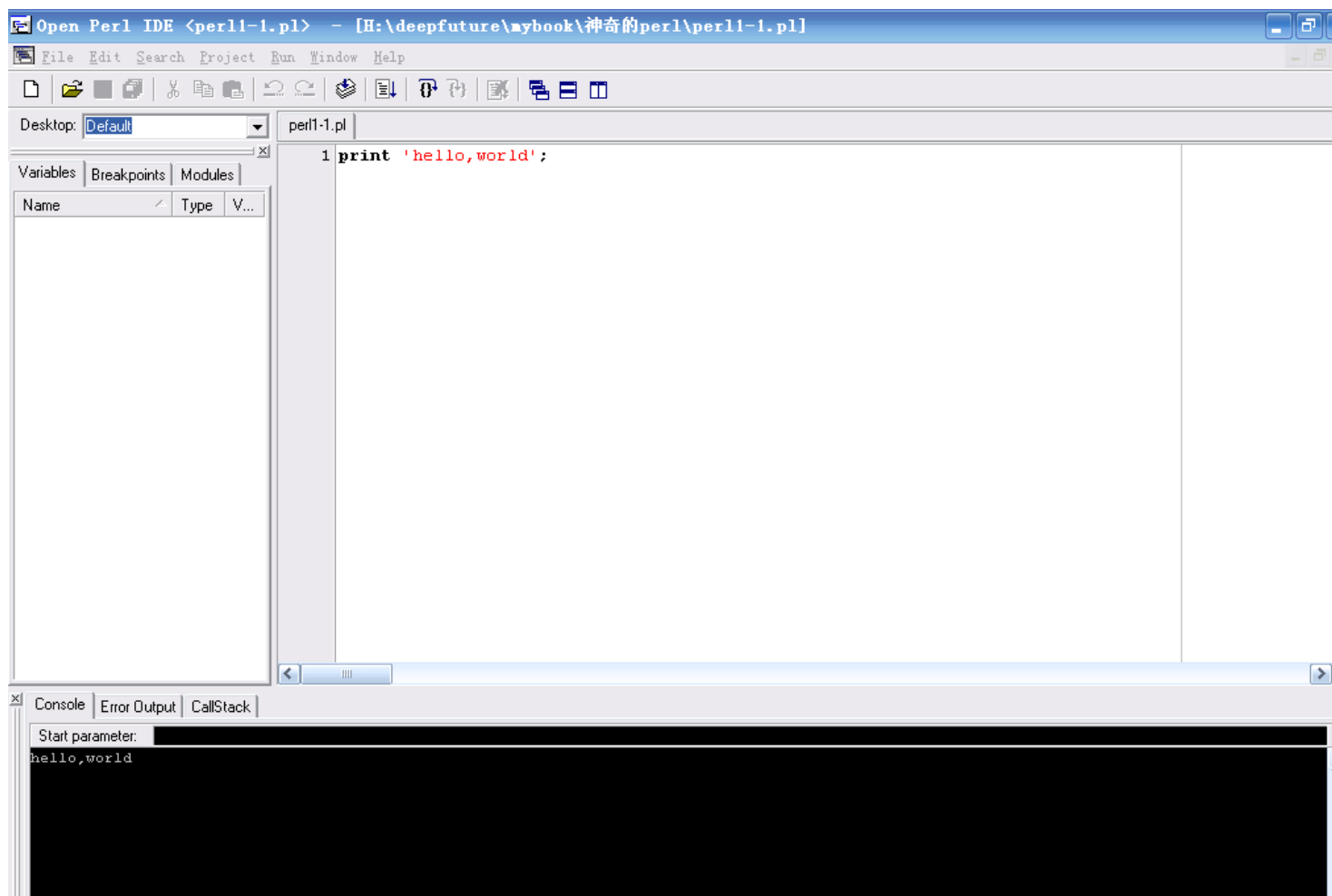
使用说明：OpenPerlIDE是绿色软件，直接运行PerlIDE.exe文件即可。如果你觉得OpenPerlIDE不好用，可以结合ultraedit一起使用。

hello,world

下面我们开始perl的helloworld之旅。打开Open Perl IDE，新建一个名为perl1-1.pl的文件，内容如下：

```
print 'hello,world';#太简单了，就一句
```

选择RUN菜单下的RUN运行该程序，在监视窗口的Console部分，hello,world已经向我们招手了。



1.2 Perl处理输入输出-1

发表时间: 2010-03-24

声明：本神奇的Perl系列为刘兴(<http://deepfuture.javaeye.com/>)原创，未经笔者授权，任何人和机构不能转载

1.第一个Perl任务

我们先放松想像一下：你是公司一个普通的开发人员，一大早，阳光明媚，你急不可待打开QQ跟最近泡上的MM聊天。此时项目组长急匆匆过来了，交给你一个紧急任务，将perl1-2.txt中的人员按城市分别存为几个文本文件，文件名是城市名，文本内容如下：

刘欢欢,20,长沙#冯军,25,上海#李兵,21,北京#李军,23,北京#李志,27,北京#黄王兵,29,长沙#赵兵,22,上海#李强兵,25,上海。。。。。。

你不禁倒吸一口凉气，天呀，又要编写一大堆代码来完成这个任务，而且还要读写文件，好久没操作文件IO了，有些命令都忘了。你也许足够聪明，想到干脆把perl1-2.txt导入到数据库中，然后运行SQL查询，按城市生成不同的表再导出，很快你打消这个念头，工作量大，如果城市众多，手工将生成的表导出几乎不可能。用C、C++、Pascal、java等高级语言确实完成这个任务，但代码量较大，使用Perl，10多行就能搞定。请允许我在此卖个关子，我们先了解一下Perl的输入输出基础，回头再来解决这个任务。

Perl语言具有以下基本特点：

- 1、Perl用#表示后面的内容为注释。
- 2、Perl语句用;表示结束。
- 3、Perl程序通常以pl结尾，如果是做为Apache容器中的cgi程序来执行脚本，必须为脚本文件名提供.pl或.cgi扩展名。
- 4、Perl对变量的表示方式非常简洁：
 - 1) Perl不要求变量有类型。
 - 2) 对于数组、列表，采用以@后接变量名表示,比如@names。
 - 3) 对于保存单个值的变量，采用\$后接变量名表示，称之为标量，比如\$name。
 - 4) 对于哈希变量，采用%后接变量名表示。

5、你可以选择这种方式执行Perl程序，在命令行中输入：

```
perl 程序名
```

2. Perl的文件处理原则

Perl的文件处理很简单，只需掌握以下2个原则：

1、打开文件使用open，关闭文件使用close,使用print向文件输出内容。

2、>表示写，<表示读，>>表示在原有内容上增加，Perl使用这几个符号来表示对文件的处理方式。

比如：

1)读取文件（FH表示文件句柄，以下几个例子均是）：

```
open (FH, "<文件名" );
```

2)写入文件,写入之前，如果文件已经存在，则删除它，重新建立一个新的。

```
open (FH, ">文件名" );
```

3)打开一个文件并增加内容。如果文件不存在，则建立一个文件再打开。如果已经存在则直接打开。

```
Open (FH," >>文件名" );
```

3、关闭一个文件

```
Close (FH);
```

3. 读取文本文件

我们首先试试看能不能按每人每行来读取并显示文件内容

```
use 5.010;
```

```
open FH,'<.\perl1-2.txt';
```

```
$mytext=<FH>;
```

```
@peoples=split '#',$mytext;
```

```
foreach $people(@peoples){  
    say $people;  
}  
close FH;
```

运行一下，效果不错

```
刘欢欢,20,长沙  
冯军,25,上海  
李兵,21,北京  
李军,23,北京  
李志,27,北京  
黄王兵,29,长沙  
赵兵,22,上海  
李强兵,25,上海
```

我们来解释一下这段perl程序:

第1行：use后接perl的版本号，表示使用perl5.1。

第2行：打开一个文件，并给予一个文件句柄，Perl对文件的处理方式是操作文件句柄，而不直接操作文件名，你可以把它理解为一个文件的别名。根据Perl的文件处理的第2个原则，perl1-2.txt表示处理的文件名，<是处理方式为读取。open的调用方式为open 句柄,处理方式后接文件名

第3行：这一行表示打开FH文件句柄所代表的文件，然后，把文件内容赋值给\$mytext变量。

第4行：从变量名的前缀@可以看出@peoples是一个数组，那么split('#', \$mytext)是什么？split是一个函数，这个函数的功能是从\$mytext变量的文本中取得被#分割的部分，即每个人的具体情况，每个部分做为数组的一个元素。

第5行：foreach表示从一个循环，在循环中每次从@peoples数组中取得一个元素值给\$people，

第6行：循环体，唯一的任务就是将每个取得的元素打印出来，因为这些元素已经放在\$people中了。say和print的不同之处在于：say在输出时会在结尾处加上一个换行，print不会。如果使用print，可以将say \$people;改成print "\$people\n"；

细心的你一定发现了 "\$people\n" 这种写法，\$people不是变量名吗，怎么能直接放在双引号中呢，Perl管这个叫变量内插，在双引号的变量最后会以变量值表示。

第7行：循环结束

第8行：表示关闭文件句柄FH。

4.单引号与双引号

等等，还有问题，单引号呢？除了双引号，单引号也可以表示字符串呀，答案是单引号内的变量不会内插，会直接以变量名表示。最后一个小提示，在Perl字符串的连接可以使用小数点符号.解决。如：“hello,” . “ ’ world” 表示 “ ’ helloworld” 。

这节结束之前我们把上面代码改一下，看看单引号和双引号的区别：

```
use 5.010;

$|=1;

open FH,'<.\perl1-2.txt';

$mytext=<FH>;

@peoples=split '#',$mytext;

foreach $people(@peoples){

    print '$people:.'"$people\n";

}

close FH;
```

从以下运行结果可以看出，单引号直接将变量名显示出来了。

```
$people:刘欢欢,20,长沙
$people:冯军,25,上海
$people:李兵,21,北京
$people:李军,23,北京
$people:李志,27,北京
$people:黄王兵,29,长沙
$people:赵兵,22,上海
$people:李强兵,25,上海
```

从上面程序中，你也许看出来了：

1、“=”这个符号太神奇了，后面接文件句柄，就能读取文件

```
$mytext=<FH>;
```


但是，上面这个语句只适于文件只有一行的情况（所谓一行是指从行首一直到换行符为止），因为\$mytext是标量，只能保存一个元素。文件超过一行，应使用数组方式：

@mytext=<FH>;#Perl返回所有内容给@mytext，数组每个元素是文件的一行。

2、foreach更神奇了，可以自动从数组中抽取元素。

```
foreach $people(@peoples){  
    say $people;  
}
```

5.小试牛刀

学以致用，我们尝试一下：读取ActivePerl目录下的Copyright.html。

```
#perl1-4.pl  
#笔者的ActivePerl安装在d盘下  
use 5.010;  
open FH,'<D:/Perl/html/Copyright.html';  
@mytexts=<FH>;  
  
foreach $mytext(@mytexts){  
    print $mytext;  
}
```

网页文件成功显示了：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
```

```
<html>
```

```
<head>

<!-- saved from url=(0017)http://localhost/ -->

<script language="JavaScript" src="./displayToc.js"> </script>

<script language="JavaScript" src="./tocParas.js"> </script>

<script language="JavaScript" src="./tocTab.js"> </script>

<title>Welcome to ActivePerl</title>

<link rel="stylesheet" href="Active.css" type="text/css">

</head>
```

1.3 Perl处理输入输出-2

发表时间: 2010-03-29

声明：本神奇的Perl系列为刘兴(<http://deepfuture.javaeye.com/>)原创，未经笔者授权，任何人和机构不能转载

6.say、print、<STDIN>与Chomp

2个新疑问产生了，这里为什么不用say，好吧，我们通过下面的小例子解释这一切

新建一个perl1-4.txt的文本文件，内容如下：

第1行

第2行

第3行

第4行

我们分别用say和print完成对这个文件的读取

say版本：

```
#perl1-4.pl
```

```
#笔者的ActivePerl安装在d盘下
```

```
use 5.010;
```

```
open FH,'<./perl1-4.txt';
```

```
@mytexts=<FH>;
```

```
foreach $mytext(@mytexts){
```

```
    say $mytext;
```

```
}
```

程序运行结果如下：

第1行

第2行

第3行

第4行

print版本：

```
#perl1-5.pl
```

```
#笔者的ActivePerl安装在d盘下
```

```
use 5.010;
```

```
open FH,'<./perl1-4.txt';
```

```
@mytexts=<FH>;
```

```
foreach $mytext(@mytexts){
```

```
    print "$mytext";
```

```
}
```

程序运行结果如下：

第1行

第2行

第3行

第4行

say版本在每行后输出一个回车换行，因为perl1-4.txt文件本身每行有一个换行符，因此，每读取文件一行就会多输出1行，这也是say和print的主要区别。某些情况下，我们并不需要换行符，即使有换行符也要去掉，有一个函数能帮我们这个忙：chomp。我们打开 perl1-5.txt，可以看到内容如下：

西红柿
黄瓜
丝瓜
冬瓜
白菜
鸡蛋

我们完成一个小任务，将这些人名在一行内输出

```
#perl1-6.pl
use 5.010;
open FH,'<./perl1-4.txt';
@mytexts=<FH>;
foreach $mytext(@mytexts){
    chomp($mytext);
    print "$mytext," ;
}
```

输出如下：

西红柿 黄瓜 丝瓜 冬瓜 白菜 鸡蛋

因为foreach循环一次，从@mytexts中取得下一个元素放在\$mytext中，这个过程并不包括删除每个元素的换行符，这个例子中每个元素就是一行，自然包括换行符，所以必须要使用chomp函数去掉末尾的换行符：

chomp(\$mytext)。

Chomp还有一个地方用得最多，就是接受用户输入时。Perl处理用户键盘输入的语句形式如下：

变量=<STDIN>

我们先看一段程序，这段程序的本意是接受用户输入，并将用户和用户的表妹的年纪在一行内显示出来。

```
#perl1-7.pl
print "你表妹多大了？";
$myinput1=<STDIN>;
print "你今年多大了？";
$myinput2=<STDIN>;
print "你的年纪是$myinput1,";
print "你表妹的年纪是$myinput2。";
$myinput=<STDIN>;
```

上面程序中的\$myinput1=<STDIN>表示接受输入，并将输入的内容赋值给\$myinput1。输出结果如下：

你表妹多大了？15

你今年多大了？22

你的年纪是15

,你表妹的年纪是22

程序并没有按想像中的运行，用户和其表妹的年纪在2行内显示出来，因为Perl接受输入，但并不包括清除用户完成输入最后敲入的换行符。因为应去掉\$myinput1所含的换行符。程序修改如下：

```
#perl1-8.pl
```

```
print "你表妹多大了？";
$myinput1=<STDIN>;
print "你今年多大了？";
$myinput2=<STDIN>;
chomp($myinput1);
print "你的年纪是$myinput1,";
print "你表妹的年纪是$myinput2";
```

输出结果如下，任务完成。

你表妹多大了？15

你今年多大了？22

你的年纪是15,你表妹的年纪是22。

7.读取每行的多列数据

上面的例子涉及的都是单列数据，而多数情况下，我们需要读取文件的多列数据。

我从朋友那得到一份北京现代汽车的报价清单，做为消费者，想从中挑选出10-15万的车型，并显示出来。

清单文件为perl1-9.csv，内容如下：

```
伊兰特 悦动1.6GL手动舒适型2010款,9.98
伊兰特 悦动1.6GL自动舒适型2010款,10.88
伊兰特 悦动1.6GLS手动豪华型2010款,10.98
伊兰特 悦动1.6GLS自动豪华型2010款,11.98
伊兰特 悦动1.8GLS手动豪华型2010款,12.18
伊兰特 悦动1.8GLS自动豪华型2010款,12.98
```

.....

.....

仔细观察可以发现，型号和价格是用“，”分隔，我们再次使用split函数。

```
#perl1-9.pl
use 5.010;
open CAR,'<.\perl1-9.csv';
foreach $carmes(<CAR>){
    ($name,$price)=split(',',$carmes);
    chomp($price);
    say "$name:$price万" if $price<15 and $price>10;
}
close CAR;
```

程序运行结果如下(.....表示以下有若干行省略没列出)：

```
伊兰特 悦动1.6GL自动舒适型2010款:10.88万
伊兰特 悦动1.6GLS自动舒适型2010款:10.98万
伊兰特 悦动1.6GLS自动豪华型2010款:11.98万
伊兰特 悦动1.8GLS手动豪华型2010款:12.18万
伊兰特 悦动1.8GLS自动豪华型2010款:12.98万
```

.....

.....

1.4 word文件批量另存为txt

发表时间: 2010-03-30

声明：本神奇的Perl系列为刘兴(<http://deepfuture.javaeye.com/>)原创，未经笔者授权，任何人和机构不能转载

```
use warnings;
use strict;
use Win32::OLE;
use Win32::OLE::Const 'Microsoft Word';
my $word = CreateObject Win32::OLE 'Word.Application' or die $!;
$word->{'Visible'} = 1;
my $document = $word->Documents->Open('H:\deepfuture\深未来\文本挖掘\相关代码\word操作\
xxx.DOC');
my $txtfilename = 'H:\deepfuture\深未来\文本挖掘\相关代码\word操作\xx.txt';
$document->saveas({FileName=>$txtfilename,FileFormat=>wdFormatText});
$word->quit();
```

1.5 Perl的语法-1

发表时间: 2010-05-02

声明：本神奇的Perl系列为刘兴(<http://deepfuture.javaeye.com/>)原创，

未经笔者授权，任何人和机构不能转载

1. 语句

1.1 结构

单个语句用;表示结束，结构如下：

语句;注释

例如：

```
print "hello,word" ;#输出hello,word
```

1.2 注释

#表示后面的内容为注释。

1.3 语句块

- 1) 语句块用{和}包围。
- 2) 一行可以有多个语句。

2. 执行及扩展名

2.1 程序扩展名

程序通常以pl结尾，如果是做为Apache容器中的cgi程序执行脚本，必须为脚本文件名提供.pl或.cgi扩展名。

2.2 程序执行

在命令行中输入:perl 程序名

3. 变量与常量

3.1 变量的表示

变量的表示非常简洁：

- 1) 不要求变量有类型。
- 2) 对于数组，采用以@后接变量名表示,比如@names。
- 3) 对于保存单个值的变量，采用\$后接变量名表示，称之为标量，比如\$name。
- 4) 对于哈希变量，采用%后接变量名表示。
- 5) 变量在第一次赋值前有一个特殊值 undef，即没有任何值。

3.2 变量的声明

1) 变量可以不声明，直接使用。

2) 声明语法是：

作用域 变量名

3.3 变量的作用域

1) 变量前不加作用域限定，表示该变量为全局变量。

2) 主要有my作用域和local作用域。

my作用域主要用来声明局部变量。my声明的变量仅作用于声明该的层次及同一层次上定义的其它子程序。

local作用域只影响声明的层次以及以内的层次，仅能在local作用域声明的层次及以内层次修改变量值。

3.4 预定义变量

预定义变量	用途
<code>\$_</code>	在执行输入和模式搜索操作时使用的默认空格变量
<code>\$.</code>	文件中最后处理的当前行号
<code>\$@</code>	由最近一个eval()运算符提供的Perl语法报错信息
<code>\$!</code>	获取当前错误信息值，常用于die命令
<code>\$0</code>	获取当前执行脚本的程序名
<code>\$\$</code>	正在执行脚本的Perl进程号
<code>\$PERL_VERSION /z</code>	Perl解释器的版本、子版本和修订版本信息
<code>@ARGV</code>	获取命令行参数列表
<code>ARGV</code>	一个特殊的文件句柄，用于遍历@ARGV中出现的所有文件名
<code>@INC</code>	库文件的搜索路径
<code>@_</code>	在子程序中，@_变量含有传给该子程序的变量内容
<code>%ENV</code>	关联数组型变量%ENV含有当前环境信息
<code>%SIG</code>	关联数组型变量%SIG含有指向信号内容的句柄

3.5 常量

4. 选择控制

在Perl中，选择控制包括if和unless，条件表达式值为空串或0时，为假，否则为真。

4.1 如果为真if

表示如果为真。

语法一：

```
if(条件表达式)
{
```

表达式为真时的语句块;

```
}
```

或者为以下语法(注意不是语句块，是单个语句)：

表达式为真时的语句块 if条件表达式;

范例1：

```
#perl0-1.pl
```

```
$x=1;
```

```
if ($x>0){
```

```
    print "ok" ;
```

```
}
```

```
print "ok" if $x>0;#和本程序中的上一个if功能完全一样
```

范例2:

```
#perl0-2.pl
```

```
$x=1;
```

```
if ($x) {#条件表达式非0，即真
```

```
    print "x:yes\n" ;
```

```
}
```

```
$y=0;
```

```
if ($y) {#条件表达式为0，即假
```

```
    print "y:yes\n" ;
```

```
}
```

```
$z=0.0;
```

```
if ($z) {#条件表达式为0，即假，注意0.0也作为数值表达式等于0
```

```
    print "z:yes\n" ;
```

```
}
```

```
$a=' 0' ;

if ($a) {#条件表达式为0，即假

    print "a:yes\n" ;

}

$b=' 00' ;

if ($b) {#条件表达式非0，即真

    print "b:yes\n" ;

}

$c=' 0.0' ;

if ($c) {#条件表达式非0，即真,注意' 0.0' 做为字符表达式不等于0

    print "c:yes\n" ;

}

$d=' ' ;

if ($d) {#条件表达式为空串，即假

    print "d:yes\n" ;

}

$e=' ' ;

if ($e) {#条件表达式为一个空格，不为空串，即真

    print "e:yes\n" ;

}
```

运行结果如下：

x:yes

b:yes

c:yes

e:yes

语法二:

```
if(条件表达式){  
    条件表达式为真时的语句块;  
}  
else{  
    条件表达式为假时的语句块;  
}
```

范例:

```
#perl0-3.pl  
  
$x=5;  
if($x>0)  
{  
    print " 正数!\n" ;  
}  
else{  
    print " 0或负数!\n" ;  
}
```

运行结果如下：

正数!

语法三:

```
if (条件表达式一)  
{  
    条件表达式一为真时的语句块;  
}  
elseif(条件表达式二){  
    条件表达式二为真时的语句块;  
}  
elseif(条件表达式三){  
    条件表达式三为真时的语句块;  
}
```

...

...

...

```
elsif(条件表达式n){
```

 条件表达式n为真时的语句块;

```
}else{
```

 所有条件表达式为假时的语句块;

```
}
```

范例:

```
#perl0-4.pl
```

```
$x=<STDIN>;
```

```
chomp($x);
```

```
if($x>=10000)
```

```
{
```

```
    print " 大于1万的正数!\n" ;
```

```
}elsif ($x>0){
```

```
    print " 小于1万的正数0!\n" ;
```

```
}
```

```
elsif ($x==0){
```

```
    print " 0!\n" ;
```

```
}
```

```
else{
```

```
    print "负数!\n" ;
```

```
}
```

运行几次程序，依次输入测试数字100、300000、-200、0。

Perl perl0-4.pl

100

小于1万的正数0!

Perl perl0-4.pl

300000

大于1万的正数!

Perl perl0-4.pl

-200

负数!

Perl perl0-4.pl

0

0!

1.6 perl语法-2

发表时间: 2010-05-17

声明：本神奇的Perl系列为刘兴(<http://deepfuture.javaeye.com/>)原创，未经笔者授权，任何人和机构不得转载

1.1 如果非真unless

表示如果非真，即：如果条件表达式不为真。

语法一：

```
unless(条件表达式) {  
    条件表达式为假执行的语句块;  
}
```

也可以写成：

条件表达式为假执行的语句块unless (条件表达式);

范例:

```
#perl0-5.pl
```

```
##%为除余操作符
```

```
$x=6;
```

```
unless($x%2){  
    print "x是偶数\n" ;  
}
```

```
print "x是偶数\n" unless($x%2);
```

输出结果为：

```
x是偶数
```

```
x是偶数
```

语法二：


```
unless(条件表达式)
{
    条件表达式为假时执行的语句块;
}
else{
    条件表达式为真时执行的语句块;
}
```

范例:

```
#perl0-6.pl
```

##为除余操作符

```
$x=9;
```

```
unless($x%2){
    print "x是偶数\n" ;
}
else{
    print "x是奇数\n" ;
}
```

输出结果为：

```
x是奇数
```

1.2 更简洁的控制方式

4.3.1 替代if-else结构的三目操作符?:

?: 操作符可以替代if-else结构，语法格式如下：

条件表示式?表达式1:表达式2

表示如果条件表示式为真则求表达式1的值，否则求表达式2的值。

范例:

```
#perl0-7.pl
```

```
##为除余操作符
```

```
$x=9;
```

```
$x%2?print "x是奇数\n":print "x是偶数\n";
```

输出结果为：

x是奇数

4.3.2 替代if结构的&&

.....
&&可替代if结构，语法格式如下：

条件表达式 && 条件表达式为真时执行的语句块

范例：

```
#perl0-8.pl
```

```
##为除余操作符
```

```
$x=8;
```

```
$x%2&& print "x是奇数\n";
```

因为8不是奇数，所以print语句不会执行，无输出结果。

1.

2.

3.

.....
1.

2.

4.3.3 替代unless结构的||

.....
||可替代unless结构，语法格式如下：

条件表达式 || 条件表达式为假执行的语句块

```
#perl0-9.pl
```

#%为除余操作符

```
$x=8;
```

```
$x%2|| print "x是偶数\n";
```

因为8是偶数，所以输出结果如下：

x是偶数

1. 循环控制

Perl循环控制很丰富，其中某些控制语法在处理列表和数组方面相当实用和高效。

1

2

3

4

5

1

2

3

4

5

5.1 while

语法一：

```
while(条件表达式) {
```

语句块:

```
}
```

范例:

```
#perl0-10.pl
```

```
#计算1到10的连乘结果
```

```
$x=1;
```

```
$jg=1;
```

```
while($x<=10)
```

```
{
```

```
$jg=$jg*$x;
```

```
$x++;
```

```
}
```

```
print "$jg\n" ;
```

输出结果如下：

```
3628800
```

语法二:

语句块while(判别运算式);

范例:

```
#perl0-11.pl
```

```
#计算1到10的连乘结果
```

```
$x=1;
```

```
$jg=1;
```

```
{
```

```
$jg*=$x;
```

```
$x++;
```

```
} while($x<=10)
```

```
print "$jg\n" ;
```

请注意\$*jg**=\$*x*这种写法，学过C语言的都知道：

\$*jg**=\$*x*与\$*jg*=\$*jg**\$*x*的结果一样，但编译效率更高。

输出结果如下：

```
3628800
```

5.2 do while

与while相比，do while保证语句块至少执行一次。

语法:

```
do
{
    语句块;
}while(条件表达式)
```

范例:

```
#perl0-12.pl

#求和1到100

$jg=0;

$x=1;

do{
    $jg+=$x;
    $x++;
}while($x<=100);
print "$jg\n";
```

输出结果如下：

```
5050
```

5.3 until

until 直到...才终止循环

语法:

until(条件表达式)

```
{  
    语句块;  
}
```

也可以写成: 语句块 until(条件表达式);

范例:

```
#perl0-13.pl
```

```
#从1开始求和，直到和大于1000终止
```

```
$jg=0;
```

```
$x=1;
```

```
until ($jg > 1000)
```

```
{  
    $jg+=$x;  
    $x++;  
}
```

```
print "1->$x:";
```

```
print "$jg\n";
```

输出结果如下：

```
1->46:1035
```

5.4 do ...until

do ...until直到...才终止循环

语法:

```
do{  
    语句块;  
}until (条件表达式);
```

范例:

```
#perl0-14.pl
```

```
#从1开始求和，直到和大于1000终止
```

```
$jg=0;  
  
$x=1;  
  
do  
  
{  
  
$jg+=$x;  
  
$x++;  
  
} until ($jg >1000);  
  
print "1->$x:";  
  
print "$jg\n";输出结果如下：
```

```
1->46:1035
```

5.5 for

循环

语法一:

```
for (初始表达式;条件表达式;循环过程运算式)  
{  
  
    语句块;  
  
}
```

范例一:

```
#perl0-15.pl
```

```
#从1开始求和，直到和大于1000终止
```

```
$jg=0;
```

```
$x=1;
```

```
for ($jg=0,$x=1; $jg <=1000;$x++)
```

```
{
```

```
    $jg+=$x;
```

```
}
```

```
print "1->$x:";
```

```
print "$jg\n";
```

输出结果如下：

```
1->46:1035
```

语法二:

每次循环依次将数组变量的元素指定给标量

```
for 标量(数组变量)
```

```
{
```

```
    语句块;
```

```
}
```

范例：

```
#perl0-16.pl
```

```
#从数组中挑出奇数
```

```
@myarr=(2,5,7,10,23,33,18);
```

```
print"奇数如下:\n";
```

```
for $num(@myarr)
```

```
{
```

```
    $num%2&&print "$num ";
```



```
}
```

```
print"\n";
```

输出结果如下：

奇数如下:

5 7 23 33

语法三:

每次循环依次将数组变量的元素指定给预定义变量\$_

```
for (数组变量)
```

```
{
```

```
    读取$_ , 处理数组的相关语句块;
```

```
}
```

范例：

```
#perl0-17.pl
```

```
#从数组中挑出奇数
```

```
@myarr=(2,5,7,10,23,33,18);
```

```
print"奇数如下:\n";
```

```
for (@myarr)
```

```
{
```

```
    $_%2&&print "$_ ";
```

```
}
```

```
print"\n";
```

输出结果如下：

奇数如下:

5 7 23 33

5.6 foreach

每次循环依次将数组变量的元素指定给标量

语法一:

```
foreach 标量(数组变量)
{
    语句块;
}
```

如果把\$variable变量省略的话,就会将数组@array的元素——指定给\$_这个内定的输出变量.

范例:

```
#perl0-18.pl

#从数组中挑出奇数

@myarr=(2,5,7,10,23,33,18);

print"奇数如下:\n";

foreach $num(@myarr)
{
    $num%2&&print "$num ";
}

print"\n";
```

输出结果如下 :

奇数如下:

5 7 23 33

语法二:

每次循环依次将数组变量的元素指定给预定义变量\$_

```
foreach (数组变量)
{
```

语句块:

```
}
```

范例:

```
#perl0-19.pl
```

```
#从数组中挑出奇数
```

```
@myarr=(2,5,7,10,23,33,18);
```

```
print"奇数如下:\n";
```

```
foreach (@myarr)
```

```
{
```

```
$_%2&&print; #使用无参数的print表示输出$_
```

```
print" ";
```

```
}
```

```
print"\n";
```

输出结果如下：

奇数如下:

5 7 23 33

5.7 last退出循环

退出循环

语法:

last

范例:

```
#perl0-20.pl
```

```
#从1开始累加，直到和大于500为止
```

```
$x=1;
```

```
$jg=0;
```

```
while (1)
{
    $jg+=$x;

    $jg>500&&last;#求和>500退出循环

    $x++;
}

print"sum(1:$x)=$jg\n";
```

输出结果如下：

```
sum(1:32)=528
```

5.8 next终止本次循环

终止本次循环，进入下一循环。

语法:

```
next
```

范例:

```
#perl0-21.pl
```

```
#求出1-100内的奇数之和
```

```
foreach ($x=1,$jg=0;$x<=100;$x++){

    $x%2||next;#如果是偶数则结束本次循环，不累加，继续下次循环

    $jg+=$x;

    $x++;

}

print "$jg\n";
```


1.7 神奇的Perl-哈希与数组(1)

发表时间: 2010-06-01

**声明：本神奇的Perl系列为刘兴
(<http://deepfuture.javaeye.com/>)原创，未经
笔者授权，任何人和机构不能转载**

神奇的Perl- 哈希与数组

1. use strict和use warnings

1.1 要求

通常来说，这2个命令总是同时使用，并且书写在程序的最前面。

1.2 作用

程序员有时会输错变量名、函数名等，use strict和use warnings对程序员进行了约束，避免了这类简单错误发生，也许算是懒惰的程序员们的恶梦吧，毕竟会导致多敲N次键盘。

笔者认为，与其将大量时间花在程序调试排错上，不如约束程序员，使他们少犯简单错误，在工程规模较大的情况下，这种结束效果非常明显。比如以下test.pl的程序(因为程序员疏忽，将\$sum+= \$l中的\$i输成了\$l)

```
#test.pl
for ($i=1;$i<100;$i++){
    $sum+= $l;
}
print $sum;
```

程序输出0，但这并不是程序需要的求和结果。

使用use strict和use warnings后,程序改成如下：

```
#test.pl
use strict;
use warnings;
my $i;
my $sum;
for ($i=1;$i<100;$i++){
    $sum+= $l;
}
print $sum;
```

程序提示错误：

Global symbol "\$l" requires explicit package name at test.pl line 7.

Execution of test.pl aborted due to compilation errors.

得益于use strict和use warnings，不习惯使用变量前事先声明的懒惰程序员犯的错误被轻易发现了。

1. 哈希

2.1 什么是哈希

哈希是一种数据结构，以数字、字符串等为索引将值存放到其中，或者从中取回值。哈希把任意长度的输入(键)，通过散列算法，变换成固定长度的散列值输出，将输入视为索引，将输出视为给索引分配的唯一内存地址，地址里存放的是这个索引代表的内容。

2.2 访问哈希

2.2.1 访问语法

2.2.1.1 读取哈希(hash) 元素

语法如下：

`$hash变量名{哈希键}`

如：

`$studentage=$age{张三};`

2.2.1.2 修改或增加哈希(hash)元素

语法如下：

`$hash变量名{哈希键}=键值。`

如：

`$age{张三}=28;`

2.2.1.3 访问不存在的hash键值

不存在的hash键值的访问结果是undef，undef即未定义。

2.2.2 哈希变量声明

对于使用了use strict语句的程序，必须事先声明变量，语法如下：

变量范围 %哈希变量名

如：

```
my %studentage;
```

2.2.3 哈希拷贝与反转

哈希之间的拷贝语法如下：

%新变量名 = %被拷贝的变量名;

如：

```
%newstudentage=%studentage;
```

哈希反转语法如下：

```
%newstudentage=reverse %studentage;
```

哈希的反转指哈希键和哈希值调换，例如：

```
my %student;
```

```
my %newstudent;
```

```
%student=("张三"=>"上海","李四"=>"广州","王五"=>"北京");
```

```
print "$student{'张三'}\n";
```

```
%newstudent=reverse %student;
```

```
print "$newstudent{'上海'}\n";
```

输出如下：

上海

张三

上述代码中，未反转之前，张三为键，上海为值，反转后，上海为键，张三为值。

2.2.4 哈希赋值

使用大箭头符号(=>)对HASH赋值，箭头左边是键，右边是值。

语法如下：

```
my %变量名=(
```

```
  键名1=>键值1,
```

```
  键名2=>键值2,
```

```
  .....
```

```
  .....
```

```
  键名n=>键值n
```

```
)
```

如：

```
#perl3-1.pl
```

```
use strict;
```

```
use warnings;
```

```
#哈希赋值
```

```
my %studentage;
```

```
%studentage=(
```

```
  "张三"=>19,
```

```
  "李四"=>22
```

```
);
```

```
#访问哈希
```

```
print "张三:$studentage{'张三'}";
```

```
print "\n";
```

1.8 神奇的Perl-哈希与数组(2)

发表时间: 2010-06-05

声明：本神奇的Perl系列为刘兴(<http://deepfuture.javaeye.com/>)原创，未经笔者授权，任何人和机构不能转载

1.1 哈希内插哈希

很多情况下，需要几层哈希来存储一个数据表，比如以下学生成绩表:

姓名	物理	化学	数学
张三	85	79	90
李四	77	96	79
王五	86	81	93

这个成绩表需要2层哈希，所谓哈希内插哈希。第1层哈希存储行，即学生姓名。第2层哈希，存储每行的信息，即学生的每门成绩。2层哈希如何结合起来存储学生成绩表，并提供相关访问呢？

首先，我们来解决存储问题。

#第1层哈希

my %student;

#第2层哈希

```
%student=(  
    "张三" =>{ "物理" =>85, "化学" =>79, "数学" =>90},  
    "李四" =>{ "物理" =>77, "化学" =>96, "数学" =>79},  
    "王五" =>{ "物理" =>86, "化学" =>81, "数学" =>93},  
);
```

在上述代码中，%student由3个键组成：张三、李四、王五，与这些键相关的值则位于花括号中，并含有嵌套的哈希（即：键值对）。张三键的相应值中含有3个嵌套的键：物理、化学、数学，其值分别是85、79、90。李四键的相应值中含有3个嵌套的键：物理、化学、数学，其值分别是77、96、79，王五也同理。

上述代码还可写成：

#第1层哈希

```
my %student;
```

#第2层哈希

```
$student{ "张三" }={ "物理" =>85, "化学" =>79, "数学" =>90};
```

```
$student{ "李四" }={ "物理" =>77, "化学" =>96, "数学" =>79};
```

```
$student{ "王五" }={ "物理" =>86, "化学" =>81, "数学" =>93};
```

其次，访问这2层哈希，比如访问王五同学的数学成绩：

第一种方式：

```
print $student{ "王五" }{ "数学" };
```

第二种方式:

```
my $studentreport=$student{ "王五" };#取得第一层HASH
```

```
print ${$studentreport }{ "数学" };#访问第二层HASH
```

总结以上方式形成以下2个程序：

1) perl3-2.pl:

```
#perl3-2.pl
#第1层哈希
my %student;
#第2层哈希
%student=(
    "张三"=>{"物理"=>85,"化学"=>79,"数学"=>90},
    "李四"=>{"物理"=>77,"化学"=>96,"数学"=>79},
    "王五"=>{"物理"=>86,"化学"=>81,"数学"=>93},
);
print $student{"李四"}{"化学"};
```

2) perl3-3.pl:

```
#perl3-3.pl
#第1层哈希
my %student;
#第2层哈希
```

```
$student{"张三"}={"物理"=>85,"化学"=>79,"数学"=>90};  
$student{"李四"}={"物理"=>77,"化学"=>96,"数学"=>79};  
$student{"王五"}={"物理"=>86,"化学"=>81,"数学"=>93};  
#访问  
my $studentreport=$student{"王五"};  
print ${$studentreport}{"物理"};
```

其实还有第3种方法，通过PERL的硬引用操作符\完成内嵌哈希的赋值。改写以上代码如下：

#第1层哈希

```
my %student;
```

#第2层哈希

```
my %cj=("物理"=>85,"化学"=>79,"数学"=>90);
```

```
$student{"张三"}=\%cj;
```

```
my %cj=("物理"=>77,"化学"=>96,"数学"=>79);
```

```
$student{"李四"}= \%cj;
```

```
my %cj=("物理"=>86,"化学"=>81,"数学"=>93);
```

```
$student{"王五"}=\%cj;
```

#访问

```
my $studentreport=$student{"王五"};
```

```
print ${$studentreport}{"物理"};
```

1. 数组

3.1 列表

列表是标量的有序集合，但不能把列表看成一个变量，数组才是变量，数组中存储列表。

列表的构造方法有以下2种：

1) (元素1,元素2 , 元素3)

构造由 3个元素组成的列表。元素类型可以不一致，比如： ('abc' ,123,' xyz')

2) qw(字符串 1 字符串 2 字符串3)

构造由 3个字符串组成的列表，做为列表元素的字符串，不必加上引号，且以空格隔开。 qw构造的列表， Perl会将其中元素视为加上单引号的字符串， 比如：

qw(abc xyz)

qw构造中的()实际为定界符，如果你愿意，也可以写成：

qw/abc xyz/ 以 "/" 为定界符

数组声明与赋值

数组声明格式如下：

my @数组变量名;

数组赋值格式如下：

@数组变量名=(元素1 , 元素2 , 元素3 , ...,元素n);

比如，声明学生姓名数组：

my @student;

@student=("李明" , " 张亮" , " 王波" , " 李安");

那么对于二维数组，如何处理呢？ 比如，一段曲线坐标点集合，可如下声明赋值：

my @coordinate;

@coordinate=(

[1.2,3.4],

[1.3,3.2],

[1.6,3.9]

);

数组不要求其元素的数据类型一致，以下赋值方式是合法的：

```
my @myinfo=( "李四" , "湛江" , " 广东" ,26,198.22);
```

对于范围数组的赋值，使用“起点..终点” 的格式

比如，赋值元素是从0到10。

```
my @mynum=(0 .. 10);
```

```
my @myalpha=( 'a' .. 'z' );
```

```
my @myalpha=( 'A' .. 'Z' );
```

数组元素访问和修改

语法格式如下：

\$数组变量名[下标]

其中下标从0开始计数。

例如：

```
my @mynum=(0 .. 10);
```

```
print $mynum[3];#访问数组元素
```

```
$mynum[3]=100;#修改数组元素
```

```
print $mynum[3];#输出修改后的数组元素
```

输出结果如下：

3

100

1.9 神奇的Perl-哈希与数组(3)

发表时间: 2010-06-06

声明：本神奇的Perl系列为刘兴(<http://deepfuture.javaeye.com/>)原创，未经笔者授权，任何人和机构不能转载

1. 哈希内嵌数组

语法格式如下：

声明及定义

```
my %哈希变量=(
```

```
    哈希键1=>[数组元素1，数组元素2，...，数组元素n],
```

```
    哈希键2=>[数组元素1，数组元素2，...，数组元素n],
```

```
    .....
```

```
    .....
```

```
    哈希键n=>[数组元素1，数组元素2，...，数组元素n]
```

```
);
```

访问内嵌数组的单个元素：

```
$哈希变量名{哈希键}->[数组下标]
```

访问内嵌数组：

```
@{$哈希变量名{哈希键}}
```

例如，游戏地图上有一组建筑物的坐标，分别如下：

武器店：123，35

修练场：85，196

防具店：67,96

魔法店：128,45

用哈希内嵌数组的方式来存储和访问这些数据。

1) 存储

```
my %game=(  
    "武器店" =>[123,35],  
  
    "修练场" =>[85,196],  
  
    "防具店" =>[67,96],  
  
    "魔法店" =>[128,45]  
);
```

2) 访问

访问防具店的坐标值:

```
printf ("%d,%d",@{$game{"防具店"}});
```

访问魔法店的坐标Y值:

```
print $game{ "防具店" }->[1];
```

可使用硬引用符来完成内嵌数组的赋值。

```
my %game;  
my @array=(123,35);  
$game{"武器店"}=\@array;  
my @array=(85,196);  
$game{"修练场"}=\@array;  
my @array=(67,96);  
$game{"防具店"}=\@array;  
my @array=(67,96);  
$game{"魔法店"}=\@array;
```

```
printf ("%d,%d",@{$game{"防具店"}});
```


1.10 神奇的Perl-哈希与数组(4)

发表时间: 2010-06-21

声明：本神奇的Perl系列为刘兴(<http://deepfuture.javaeye.com/>)原创，未经笔者授权，任何人和机构不能转载

2. 数组内嵌哈希

声明及定义

```
my @数组变量名=(  
  
    {  
  
        数组第1个元素的哈希键1=>值1 ,  
  
        数组第1个元素的哈希键2=>值2,  
  
        .....  
  
        数组第1个元素的 哈希键n=>值n  
  
    },  
  
    {  
  
        数组第2个元素的哈希键1=>值1 ,  
  
        数组第2个元素的哈希键2=>值2,  
  
        .....  
  
        数组第2个元素的哈希键n=>值n  
  
    }  
  
    ....  
  
    {  
  
        数组第n个元素的哈希键1=>值1 ,  
  
        数组第n个元素的哈希键2=>值2,
```

.....

数组第n个元素的哈希键n=>值n

```
}  
  
);
```

访问内嵌哈希的键值：

`$数组变量名[数组下标]{哈希键}`

例如，我们使用数组内嵌哈希来存储和访问某文具店的库存:

品名	价格	数量
钢笔	8.06	50
2B铅笔	2.10	300
胶水	5.80	60
剪刀	12.20	28

存储：

```
my @wenju=(  
  
{name=>" 钢笔" ,price=>8.06, amount =>50},  
  
{name=>" 2B铅笔" ,price=>2.10,amont=>300},  
  
{name=>" 胶水" ,price=>5.80,amont=>60},  
  
{name=>" 剪刀" ,price=>12.20,amont=>28}  
  
);
```

访问品名和数量：

```
print $wenju[2]{ name };#品名  
  
print $wenju[2]{ amont };#数量
```

2. 哈希的遍历



7.1 第二个任务

同事小王是某BBS驾游板块的版主，最近想在成员最多的城市组织聚会。小王将该版块发言成员信息导出为qqyou.txt文本文件，其中每行存放一位成员的信息，第一列是姓名，第二列是城市，每列以空格分隔，内容如下：

王兴 湛江

张华 深圳

李光 广州

王安 湛江

张美 深圳

张江美 深圳

胡海明 深圳

黄国刚 深圳

刘灿 湛江

李发 广州

哈希的遍历可通过while循环完成，每次循环从哈希变量中取出一对键值。语法如下：

```
while (( $键名,$键值)=each %哈希变量名){  
    #对取出的键名值进行操作  
}
```

我们来完成第二个任务，代码如下：

```
#perl3-4.pl  
  
use strict;  
  
use warnings;  
  
my %zjycity;  
my $mycity;  
my $cityinfo;  
my $name;  
my $city;  
my $zjycity;  
my $citycount;  
my $pcount=0;  
my @citys,;
```

```
open CITY,"<perl3-4.txt";
```

```
#从perl3-4.txt文件中读成员的姓名及城市信息
```

```
foreach $cityinfo(<CITY>){
```

```
($name,$city)=split(' ', $cityinfo);  
$zjycity{$city}++;  
}  
#循环遍历哈希，并找到成员数最多的城市  
while (($zjycity,$citycount)=each %zjycity){ # $zjycity为城市名， $citycount为成员数  
    if ($citycount>$pcount) {  
        $pcount=$citycount;  
        $mycity=$zjycity;  
    }  
}  
print "最佳聚会地点：$mycity\n";
```

以上代码中，%zjycity为哈希变量，首先从文本文件perl3-4.txt读取数据，并生成城市统计哈希%zjycity，键为城市名，键值为城市的成员数目，然后通过循环遍历哈希变量%zjycity，找到成员数最多的城市，即找到最大键值对应的键名。

1.11 perl处理输入和输出-3

发表时间: 2010-06-27

声明：本神奇的Perl系列为刘兴(<http://deepfuture.javaeye.com/>)原创，未经笔者授权，任何人和机构不能转载

这段程序有几处耐人寻味：

1、`($name,$price)=split(',',$carmes);`

`($name,$price)`表示一个列表，列表可以理解为标量的有序集合，列表存储着数据的集合，而数组可理解为存储着列表的变量。列表可用用(元素1，元素2，元素3,...,元素n)的方式表示。这句代码完成一个功能把等号右边分隔的每个元素赋值给左边的列表，其中第1个元素给`$name`，第二个元素给`$price`。

2、`chomp($price);`

`$price`难道会包括换行符？这是初学者最容易忽视的问题：文件中每行只有2个元素，最后一个元素（也就是第二个元素）的末尾包括换行符。

3、`foreach $carmes(<CAR>)`

从CAR文件句柄中读取数据给`$carmes`变量，每循环一次读取一行，也可以如下书写：

```
@mytext=<CAR>;
```

```
foreach $carmes(@mytext){
```

```
.....
```

```
}
```

4、`say "$name:$price万" if $price<15 and $price>10;`

想必大家对if语句并不陌生，这是if的一种特殊用法，此语句的含义表示：如果`$price<15`并且`$price>10`，则执行`say "$name:$price万"`语句。关于if，我们将在下一章详述。

8.写文件

Perl写文件的方式非常简洁，使用以下2种形式：

```
say 文件句柄 内容
```

```
print 文件句柄 内容
```

把Perl1-9的代码稍加修改，将10万到15万之间车写入另一个CSV格式文件中。

```
#perl1-10.pl
```

```
use 5.010;
```

```
open CAR,'<.\perl1-9.csv';
```

```
open CAROUT,'>.\perl1-10.txt';#以写入方式打开文件perl1-10.txt
```

```
foreach $carmes(<CAR>){
```

```
    ($name,$price)=split(',',$carmes);
```

```
    chomp($price);
```

```
    say CAROUT "$name:$price万" if $price<15 and $price>10;#写文件
}
close CAR;
```

9.带格式输出Sprintf和printf

Sprintf返回格式化后的字符串，语法如下：

sprintf(“格式”,要格式的字符串列表)

格式主要有以下几种：

%%	百分号
%c	把给定的数字转化为字符
%s	字符串
%d	带符号整数，十进制
%u	无符号整数，十进制
%o	无符号整数，八进制
%x	无符号整数，十六进制
%e	浮点数，科学计算法
%f	浮点数，用于固定十进制计数
%g	浮点数，包括%e和%f

printf等同于以下语句：

[print](#) sprintf(“格式”,要格式的字符串列表);

我们使用sprintf，以元为单位输出10万到15万的车的价格列表。

```
#perl1-11.pl
use 5.010;
open CAR,'<.\perl1-9.csv';
foreach $carmes(<CAR>){
    ($name,$price)=split(',',$carmes);
    chomp($price);
    say $name.':'.sprintf( "%d" , $price*10000) if $price<15 and $price>10;
# $price*10000完成将万元转换成元
}
close CAR;
```

10.join

split和join函数在处理文本时经常使用，join的功能与split相反，它将列表中用逗号分隔的字符串元素连接成单个字符串，并返回这个字符串，调用格式如下：

变量=join(分隔符,字符串元素列表);

我们将perl1-11.pl修改一下，从perl1-9.csv中提取价格在10万到12万之间的汽车，输出为另一个CSV格式文件。

```
#perl1-12.pl
use 5.010;
open CAR,'<.\perl1-9.csv';
open CAROUT,'>.\perl1-11.csv' ;以写入方式打开文件perl1-11.csv
foreach $carmes(<CAR>){
    ($name,$price)=split(',',$carmes);
    chomp($price);
    say CAROUT join( ' ', $name,sprintf( "%d" , $price*10000))
    if $price<12 and $price>10;
# $price*10000完成将万元转换成元
}
close CAR;
close CAROUT;
```

注意下面这个语句：

```
say CAROUT join( ' ', $name,sprintf( "%d" , $price*10000))
if $price<12 and $price>10;
```

因为csv格式每行元素之间以逗号分隔，所以使用逗号做为分隔符，将\$name和格式转换后的\$price连接为单个字符串输出到CAROUT文件句柄中。

11.转义字符表示

Perl的主要转义字符如下：

结 构 含 义

\n 换行

\r 回车

\t 水平置表符

\f 换页符

\b 退格

\v 垂直置表符

\a 响铃

\e Esc

\007 任一八进制ASCII值(这里007表示bell)

\x7f 任一十六进制ASCII值

\cC 任一“控制”字符

\\ 反斜杠

\" 双引号

\l 下一字母小写

\L 以后所有字母小写直到\E

\u 下一字母大写

\U 以后所有字母大写直到\E

\E 结束\L和\U

12.硬引用、指针

Perl没有像C语言一样的可直接操纵内存的指针，但有类似于指针的引用(也称为硬引用)。引用是一种标量，含有其他类型数据的地址，可把它理解为一个内容为地址值的变量。

在perl里面使用“\”来创建引用；使用\$解引用后，可使用引用指向的标量；使用@解引用，使用引用指向的数组；使用%解引用后，使用引用指向的哈希。这些使用方式具备修改权力，如：

```
#perl1-13.pl

$hello="hello";

$myhello=\$hello;

print "$myhello\n";#输出$hello地址

print "$$myhello\n";#输出$hello变量内容

$$myhello="hello!";#通过$myhello引用修改$hello变量的值

print "$hello\n";#输出$hello变量内容
```

输出结果如下：

```
SCALAR(0x182a6f4)

hello

hello!
```

这个程序中，\$myhello存有\$hello的地址，存有\$hello变量的地址，通过\$myhello能直接操作\$hello，而不是操作\$hello的复制版本。

符号引用（也称为软引用）操作符为\$，相当于宏替换，如以下代码中，\$\$var相当于取\$var的内容为变量名，完成宏替换，最终表示\$name。

```
$var="name";

$$var="张三";

print "$name\n";
```

输出结果为

```
张三
```

13.完成第一个任务

这一章要结束了，输入输出基础介绍完了，能完成本章开始时提出的任务了，将Perl1-2.txt中的人员按城市分别存为几个文本文件，文件名是城市名。

```
#perl1-14.pl

use 5.010;
```

```
open PEOPLE,'<.\perl1-2.txt';
$mytext= <PEOPLE>;
@peoples=split('#', $mytext);
foreach $peoplemes(@peoples){
    ($name,$age,$city)= split(',',$peoplemes);
    chomp($city);
    open CITY,">>$city.txt";
    say CITY join(',',$name,$age,$city);
    close CITY;
}
close PEOPLE;
```

请大家注意这个语句：open CITY,">>\$city.txt";

打开文件的方式使用的是增加方式。根据Perl的文件处理原则，>>表示在原有内容上增加。

1.12 神奇的perl-哈希和数组(5)

发表时间: 2010-06-27

声明：本神奇的Perl系列为刘兴(<http://deepfuture.javaeye.com/>)原创，未经笔者授权，任何人和机构不能转载

1.1 第三个任务

BOSS交给小张一个文本文件，为去年销售人员季度完成的销售额，要求小张统计出前三名做为金牌销售员人选。

张三 25 9 10 60

李四 17 16 79 25

黄三 5 10 5 2

黄光 10 3 6 9

刘发 6 5 9 12

以上文件第一列是姓名，第二列是第一季度销售额，第三列是第二季度销售额，第四列是第三季度销售额，第五列是第四季度销售额，每列用tab分隔。

完成这个任务需要用到内嵌哈希。

内嵌哈希的遍历语法如下：

```
while (( $键名,$键值)=each %哈希变量名){  
    #对取出的键名值进行操作，其中键值是内嵌哈希的引用。  
    #访问内嵌哈希键值的语法为：${$键值}{$内嵌哈希的键名};  
}
```

我们先计算每个人的全年的总销售额并输出：

```
#perl3-5.pl
```

```
use strict;
```

```
use warnings;
```

```
use 5.010;
```

```
my %p;
```

```
open PINFO,"<perl3-5.txt";
```

#从perl3-5.txt文件中读成员的姓名及城市信息

```
foreach my $pinfo(<PINFO>){

    chomp($pinfo);

    my %info;

    (my $name, my $jd1,my $jd2,my $jd3,my $jd4)=split('\t',$pinfo);

    $info{jd1}= $jd1;

    $info{jd2}= $jd2;

    $info{jd3}= $jd3;

    $info{jd4}= $jd4;

    $info{all}= $jd1+$jd2+$jd3+$jd4;

    $p{$name}=\%info;

}

close PINFO;

while ((my $key,my $value)=each %p){

    say "$key=>${$value}{all}";

}
```

结果如下：

刘发=>32

张三=>104

李四=>137

黄光=>28

黄三=>22

1.13 神奇的Perl-哈希和数组(6)

发表时间: 2010-07-15

声明：本神奇的Perl系列为刘兴(<http://deepfuture.javaeye.com/>)原创，未经笔者授权，任何人和机构不能转载

1.1.1 each、values、keys、sort

1、 each

each一般作用于哈希和数组，以2元素的列表形式返回哈希的键值对和数组的索引、值对。语法如下：
each %hash

each @array

比如：

```
while (($key, $value) = each %hash) {#$key取得键, $value取得值
```

```
print    "$key:$value\n";
```

```
}
```

```
while (($index, $value) = each @array) {#$index取得索引, $value取得值
```

```
print    "$index:$value\n";
```

```
}
```

2、 values

values函数以列表形式返回哈希和数组的所有值。语法如下：

values %hash

values @array

比如：

```
for $value(values %hash)
```

```
{ #输出hash的所有值(注意不是键值对)。
```

```
print    "$value\n" ;
```

```
}
```

3、 keys

keys函数以列表形式返回哈希和数组的所有键。语法如下：

```
keys %hash
```

```
keys @array
```

比如：

```
for $key(keys %hash)
```

```
{ #输出hash的所有键(注意不是键值对)。
```

```
    print "$key\n";
```

```
}
```

以一段程序为例说明keys和values的用法：

```
$hash{A}=0;
```

```
$hash{B}=1;
```

```
$array[0]="a";
```

```
$array[1]="b";
```

```
for $value(values %hash)
```

```
{ #输出hash的所有值(注意不是键值对)。
```

```
    print "$value\n";
```

```
}
```

```
for $key(keys %hash)
```

```
{ #输出hash的所有键(注意不是键值对)。
```

```
    print "$key\n";
```

```
}
```

```
for $value(values @array)
```

```
{ #输出数组的所有值，注意不是索引、值对)。
```

```
    print "$value\n";
```

```
}
```

```
for $index(keys @array)
```

```
{ #输出数组的所有索引，注意不是索引、值对)。
```

```
    print "$index\n";
```

```
}
```

输出结果如下：

0
1
A
B
a
b
0
1

4、 sort

sort实现对列表排序后返回列表，主要有以下几种用法：

1) sort列表

按普通字符串比较排序.比如：

```
my @myarr;  
  
@myarr= sort @myarr;
```

2) sort 子程序名 列表

如果使用了子程序名，可以实现自定义排序，主要分为数字排序、字符串排序和综合排序。\$a和\$b代表需要比较列表的2个元素。

首先编写排序用函数。

A) 数字排序

```
sub by_number{$a<=>$b};#<=>是飞碟操作符，比较2个数字返回-1,0,1
```

B) 字符串排序

```
sub by_str{$a cmp $b};
```

C) 综合排序

定义一个子程序，返回1、0、-1。经过子程序判断，第一个参数(\$a)位于第二个参数(\$b)之前，返回-1,否则返回1,如果顺序不分先后，返回0。

```
sub by_num{#定义一个数字升序排序  
  
if ($a<$b) {-1} elsif ($a>$b) {1} else {0};  
  
}
```

然后实施排序，以数字排序为例。

```
my @myarr;  
  
@myarr= sort by_number @myarr;
```

1.1.2 完成第三个任务

我们利用sort排序完成内嵌哈希值的排序

哈希排序利用

```
#perl3-6.pl
```

```
use strict;
```

```
use warnings;
```

```
use 5.010;
```

```
my %po;
```

```
open PINFO,"<perl3-5.txt";
```

```
#从perl3-5.txt文件中读成员的姓名及城市信息
```

```
foreach my $pinfo(<PINFO>){
```

```
    chomp($pinfo);
```

```
    my %info;
```

```
    (my $name, my $jd1,my $jd2,my $jd3,my $jd4)=split('\t',$pinfo);
```

```
    $info{jd1}= $jd1;
```

```
    $info{jd2}= $jd2;
```

```
    $info{jd3}= $jd3;
```

```
    $info{jd4}= $jd4;
```

```
    $info{all}= $jd1+$jd2+$jd3+$jd4;
```

```
    $po{$name}=\%info;
```

```
}
```



```
foreach my $key (reverse sort {${$po{$a}}{all}<=>${$po{$b}}{all}} keys %po){  
  
    print "$key=>${$po{$key}}{all}\n";  
  
}
```

1.14 神奇的Perl-哈希和数组(7)

发表时间: 2010-07-17

声明：本神奇的Perl系列为刘兴(<http://deepfuture.javaeye.com/>)原创，未经笔者授权，任何人和机构不能转载

1.1 push、pop、shift、unshift

1、push在数组末尾处增加元素，pop就是从数组的末尾取出元素。具体语法如下：

push ARRAY, (以逗号分隔的要增加的值列表)
push ARRAY,要增加的值

pop ARRAY

比如：

```
my @myarr;

push @myarr,-1;

push @myarr,0;

push @myarr,1;

print pop @myarr;

print "\n";

print pop @myarr;

print "\n";

print pop @myarr;

print "\n";
```

输出如下：

1

0

-1

2、unshift在数据的起始增加元素，shift在数组的起始取出一个元素。shift弹出数组的值，并返回它，unshift的队列中增加值。具体语法如下：

unshift (ARRAY,(以逗号分隔的要增加的值列表))

unshift ARRAY,要增加的值

shift ARRAY

比如：

输出如下：

1

0

-1

3、通过对push、pop、shift、unshift使用组合，可以实现堆栈和队列的操作。

1) 堆栈，后进先出，通过push和shift组合实现

```
my @myarr;  
  
push @myarr,-1;  
  
push @myarr,0;  
  
push @myarr,1;  
  
print pop @myarr;  
  
print "\n";  
  
print pop @myarr;  
  
print "\n";  
  
print pop @myarr;  
  
print "\n";
```

输出如下：

1

0

-1

2)队列，先进先出，通过push和shift组合实现

```
my @myarr;  
  
push @myarr,-1;  
  
push @myarr,0;  
  
push @myarr,1;  
  
print shift @myarr;  
  
print "\n";  
  
print shift @myarr;  
  
print "\n";  
  
print shift @myarr;  
  
print "\n";
```

输出如下：

-1

0

1

1.2 第四个任务

BOSS要求编写一个脚本，搜索某目录下所有以PHP扩展名结尾的源代码文件，并生成一个HTML文件，其内容是按目录归类的文件列表。

我们先简单介绍一下如何操作目录及文件：

通常使用`opendir`、`closedir`、`readdir`函数操作目录，格式如下：

```
opendir(DH, $some_dir) || die "can't opendir $some_dir: $!";
```

`#opendir`打开目录，将目录句柄赋值给`DH`。`$some_dir`为要打开的目录名。

`#开始循环读取目录下的文件`

```
foreach $file(readdir DH){#读取目录下的文件,readdir返回文件列表
```

```
    if ( -d $file){
```

```
        # -d 文件名称 检测$file储存的文件名称是否为目录。
```

```
        #$file是子目录，对子目录进行处理
```

```
    }
```

```
    else {
```

```
        #$file是文件，对文件进行处理。
```

```
    }
```

```
}
```

```
closedir DH;#关闭目录句柄
```

我们的思路是：

- 1) 打开要读取的目录。
- 2) 循环读取目录中所有文件，将新发现的子目录存储在数组中。
- 3) 继续取出数组中的目录，如果数组中没有元素，转到第4步，否则转到第1步。
- 4) 退出。

存放目录的数组采用堆栈的方式操作，可模拟深度优先搜索，较好地展示目录树结构。代码如下：

```
#perl3-8.pl
use 5.010;

use warnings;

use strict;

my $dirs='E:\TDDOWNLOAD\latest';# 初始路径

my $lsfile;

my @filedir;

my %filediryj;#已经访问过的目录 ，哈希键值对为：目录名=>文件数目

my $flcount;

my $ppfile;#文件匹配符 以正则方式表达的

open MYTXT,">phpfiles.htm";
```

#输出HTML文件头

```
say MYTXT '<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR'
```

```
say MYTXT '<html xmlns="http://www.w3.org/1999/xhtml">    ';
```

```
say MYTXT '<head> ';
```

```
say MYTXT '<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />    ';
```

```
say MYTXT '<title>查找某目录下的某类文件</title>    ';
```

```
say MYTXT '</head> ';
```

```
say MYTXT '<body> ';
```

```
$ppfile='php$'; #要寻找文件扩展名为php，php后紧跟符号$表示文件名以php结尾。
```

```
push @filedir,$dirs;
```

```
$|=1;
```

```
my $filedir;
```

```
while ($filedir=shift(@filedir)){
```

```
    opendir DH,$filedir or die "不能打开$filedir"; #打开目录
```

```
    say MYTXT "<br>$filedir<br>";
```

```
    $flcount=0;
```


}

```
#输出HTML文件尾
```

```
say MYTXT '</body>  ';
```

```
say MYTXT '</html>  ';
```

1.15 神奇的perl-正则表达式(1)

发表时间: 2010-07-26

声明：本神奇的Perl系列为刘兴(<http://deepfuture.javaeye.com/>)原创，未经笔者授权，任何人和机构不能转载

第五章 正则表达式

1. PERL正则基础

正则表达式是一种序列或字符模式，负责在搜索和替换文本时对文本内容进行字符串匹配。Perl中的正则表达式由待匹配字符串或模式串或2者混合而成，一般以斜杠（/）作为定界符。

Perl正则模式串主要有以下几种：

. 匹配单个除换行符以外的字符

a? 匹配 0 次或一次 a 字符(a为任意字符，如:a*、b*、2*等)

a* 匹配 0 次或多次 a 字符(a为任意字符，如:a*、b*、2*等)

a+ 匹配 1 次或多次 a 字符(a为任意字符，如:a+、b+、2+等)

. * 匹配 0 次或一次的任何字符

. + 匹配 1 次或多次的任何字符

{m} 匹配刚好是 m 个 的指定字符

{m,n} 匹配在 m个 以上 n个 以下的指定字符

{m,} 匹配 m个 以上 的指定字符

[] 匹配符合 [] 内的字符

[^] 匹配不符合 [] 内的字符

(x|y) 匹配x或y

(x) 匹配x模式串

[0-9] 匹配单个数字

[a-z] 匹配单个小写字母

[^0-9] 匹配单个非数字字符

[^a-z] 匹配单个非小写字母字符

^ 匹配字符开头的字符

\$ 匹配字符结尾的字符

`\d` 匹配单个数字的字符，和 `[0-9]` 语法一样

`\d+` 匹配多个数字字符，和 `[0-9]+` 语法一样

`\D` 单个非数字字符

`\D+` 匹配多个非数字字符

`\w` 单个英文字母或数字的字符，

`\W` 单个非英文字母或数字的字符

`\W+` 匹配多个非英文字母或数字字符

`\s` 空格等同于 `[\n\t\r\f]`

`\s+` 等同于 `[\n\t\r\f]+`

`\S` 单个非空格字符

`\S+` 多个非空格字符

我们来看几个例子:

`/abc/` 匹配所有包括abc的字符串,比如，匹配 `"123abcxxx"`、`"abcde"`。

`/ab(c|d)/` 匹配所有包括abc或abd的字符串,比如，匹配 `"123abcxxx"`、`"abde88"`

`/abc\d+/` 匹配所有包括abc后接至少一个数字的字符串,比如，匹配 `"xxabc456"`、`"abc678de"`。

`/^abc\d+/` 匹配所有以abc后接至少一个数字开头的字符串,比如，匹配 `"abc456"`、`"abc678de"`，不匹配 `"xxabc456"`、`"xxbc456"`。

`/^abc\d+$/` 匹配所有以abc开头，至少一个数字结尾，且abc后接至少一个数字的字符串,换句话说，仅匹配符合abc后接至少一个数字，不包括任何其它字符。比如，匹配 `"abc456"`、`"abc1"`，不匹配 `"abc456x"`、`"bc456"`、`"abc"`。

`/[^0-9]abc\d+/` 匹配所有包括非数字字符后接abc及至少一个数字的字符串，比如，匹配 `"xyabc456"`、`"aabc1"`，不匹配 `"456abc98"`、`"bdc456"`、`"0abc1"`。

`/^[^0-9]abc\d+/` 匹配所有以非数字字符后接abc及至少一个数字的形式开头的字符串，比如，匹配 `"aabc456"`、`"xabc1you"`，不匹配 `"8abc98ww"`、`"bd456"`、`"0abc1"`。

1.16 神奇的perl-正则表达式(2)

发表时间: 2010-07-27

声明：本神奇的Perl系列为刘兴(<http://deepfuture.javaeye.com/>)原创，未经笔者授权，任何人和机构不能转载

1. 匹配

2.1 基础语法

1、Perl使用正则表达式可实现字符串的查询和匹配。

2、语法格式(regex为模式串)：

\$字符串变量名=~m/<regex>/

也可以简写为\$字符串变量名=~ /<regex>/

通常如下使用:

```
If ($字符串变量名=~m/<regex>/){
```

```
    #匹配成功，执行成功后的语句
```

```
}
```

3、示例

```
#perl4-0.pl
my $name=" zhangsang" ;
if ($name=~m/san/){
#匹配成功 print  "ok" ;
}
```

很显然, zhangsang是能够匹配的, 因此输出为:

ok

2.1 匹配修饰

匹配修饰符在模式定界符之后使用

i : 忽略大小写, 如:/abc/i

g : 查找所有匹配值, 返回一个匹配到的列表。如: /abc/g

例如:

```
#perl4-1.pl
my $name="hello!Beijing";
#我们首先使用 i, 用大写 "LLO" 串进行匹配, 由于忽略大小写, 匹配是成功的。
print "使用 i\n";
if ($name=~m/LLO/i){
#使用i, 匹配成功
print "使用 i-ok\n";
}
#使用 g, 用 "e" 进行全局匹配, e 出现了2次, 因此返回2个e组成的列表
print "使用 g\n";
my @result=($name=~m/e/g);
print @result;#输出匹配列表数据
print "\n";
```

```
print scalar(@result);#返回@result数组大小(全局匹配成功数目)为2
。 scalar(@数组变量名)返回数组大小
print "\n";
if ($name=~m/e/g){
    #使用g , 匹配成功2处
    print "使用 g-ok\n";
}
#同时使用 g 和i进行匹配
    print "使用 g 和i\n";
if ($name=~m/L/gi){
    #使用了g和i匹配成功
    print "使用 g 和i-ok\n";
}
```

输出结果如下:

使用 i

使用 i-ok

使用 g

ee

2

使用 g-ok

使用 g 和i

使用 g 和i-ok

1.17 神奇的Perl-正则表达式(3)

发表时间: 2010-07-28

声明：本神奇的Perl系列为刘兴(<http://deepfuture.javaeye.com/>)原创，未经笔者授权，任何人和机构不能转载

2.1 特殊字符匹配

在Perl 正则中，某些字符具有特殊含义，譬如 “?”、“*”、“.”、“+”等。如果模式串需要把这些符号当作纯文本来处理的话，就必须在它前面加上反斜杠(\)。

比如:

```
#perl4-2.pl

my $name="你叫什么名字?";

my $filename="plane.jpg";

if ($name=~m/\?/){

    print "$name是疑问句\n";#匹配疑问句

}

if ($filename=~m/\.jpg$/){

    print "$filename为JPG类型图像文件\n";#文件扩展名为jpg

}
```

输出结果如下：

你叫什么名字?是疑问句

plane.jpg为JPG类型图像文件

2.2 第五个任务公司新开发的小型ERP系统试运行时，经常出故障，小黄奉命查出故障来源，系统记录用户操作保存在run.log的日志文件中。

run.log的结构大致如下，每列信息以#相隔。

```
2010-07-01#09:12:10#zhangsang login success
```

```
2010-07-01#09:15:30#zhangsang save success
```

```
2010-07-01#09:16:50#lisi login error#系统错误#用户不存在
```

```
2010-07-01#09:18:05#zhangsang exit success
```

```
2010-07-01#13:19:10#wangwu login error#系统错误#密码错误
```

```
2010-07-02#09:20:10#lisi login success
```

```
2010-07-02#09:31:32#lisi save error#SQL执行失败#insert error
```

```
2010-07-02#09:28:32#lisi exit success
```

由于日志文件过于巨大，小黄无法人工查找错误，只能用perl编写以下脚本筛选错误,并将错误输出到单独的文本文件runerr.txt中。

```
#perl4-3.pl
use strict;
use warnings;

open LOG,"<run.log";
open ERRLOG,">runerr.txt";
foreach my $line(<LOG>){
    if ($line=~m/error/i) {#只匹配日志中的错误事件，错误事件都包括“error”字符
        chomp($line);#去除换行符
        (my $mydate,my $mytime,my $mysj,my $myerr,my $errinfo)=split("#",$line);#获取事件信息
        print ERRLOG "-----\n";
        print ERRLOG "日期:$mydate\n时间:$mytime\n事件:$mysj\n错误类型:$myerr\n错误信息:$errinfo\n"
    }
}
```

```
close LOG;  
close ERRLOG;
```

脚本执行完毕后，打开runerr.txt，错误信息如下，一目了然：

日期:2010-07-01

时间:09:16:50

事件:lisi login error

错误类型:系统错误

错误信息:用户不存在

日期:2010-07-01

时间:13:19:10

事件:wangwu login error

错误类型:系统错误

错误信息:密码错误

日期:2010-07-02

时间:09:31:32

事件:lisi save error

错误类型:SQL执行失败

错误信息:insert error

1.18 神奇的Perl-最佳Perl入门读物

发表时间: 2010-07-29

Perl脚本语言最佳入门读物

刘兴[QQ:1477022617] [HTTP://DEEPPUTURE.JAVAEYE.COM/](http://deepfuture.javaeye.com/) 版权所有,未经作者书面授权请勿转载

目录

第一章 拥抱Perl

1. Perl是什么?
2. Perl是神奇的
3. Perl运行环境
 - 3.1 Activeperl
 - 3.2 IDE
4. hello,world

第二章 Perl语法

1. 语句
 - 1.1 结构
 - 1.2 注释
 - 1.3 语句块
2. 执行及扩展名
 - 2.1 程序扩展名
 - 2.2 程序执行
3. 变量与常量
 - 3.1 变量的表示
 - 3.2 变量的声明
 - 3.3 变量的作用域
 - 3.4 预定义变量
 - 3.5 使用print

- 3.5.1 基本用法
- 3.5.2 输出缓冲
- 3.6 defined、undef与delete
- 3.7 exists
- 3.8 常量
- 4. 子程序
- 5. 基本操作符
 - 5.1 算术操作符
 - 5.2 自增与自减
 - 5.3 比较操作符
 - 5.4 字符串操作符
 - 5.5 逻辑运算符
 - 5.6 位操作符
 - 5.7 赋值操作符
- 6. 选择控制
 - 6.1 如果为真if
 - 6.2 如果非真unless
 - 6.3 更简洁的控制方式
 - 6.3.1 替代if-else结构的三目操作符?:
 - 6.3.2 替代if结构的&&
 - 6.3.3 替代unless结构的||
- 7. 循环控制
 - 7.1 while
 - 7.2 do while
 - 7.3 until
 - 7.4 do ...until
 - 7.5 for
 - 7.6 foreach
 - 7.7 last退出循环
 - 7.8 next终止本次循环

第三章 Perl处理输入输出

1. 第一个Perl任务
2. Perl的文件处理原则
3. 分割字符串split
4. 读取文本文件
5. 单引号与双引号
6. 小试牛刀
7. say、print、<STDIN>与Chomp
8. 读取每行的多列数据
9. 写文件
10. 带格式输出Sprintf和printf
11. join
12. 转义字符表示
13. 引用、符号引用、指针
14. 在子程序中使用引用传参
15. 完成第一个任务

第四章 哈希与数组

1. use strict和use warnings
 - 1.1 要求
 - 1.2 作用
2. 哈希
 - 2.1 什么是哈希
 - 2.2 访问哈希
 - 2.2.1 访问语法
 - 2.2.2 哈希变量声明
 - 2.2.3 哈希拷贝与反转
 - 2.2.4 哈希赋值

- 2.3 哈希内嵌哈希
- 3. 数组
 - 3.1 数组声明与赋值
 - 3.2 元素访问与修改
- 4. 哈希内嵌数组
- 5. 数组内嵌哈希
- 6. 数组内嵌数组
- 7. 删除、清空哈希和数组
- 8. 哈希的遍历
 - 8.1 第二个任务
 - 8.2 第三个任务

8.2.1 遍历内嵌哈希

.....

解压密码：<http://deepfuture.javaeye.com/>

例子解压前请先看readme.txt

附件下载:

- 神奇的perl例子.zip (30.6 KB)
- dl.javaeye.com/topics/download/c1498486-e5a8-32cc-847f-e69c4c54c617
- 神奇的perl-最佳Perl入门.pdf (998.1 KB)
- dl.javaeye.com/topics/download/06e66544-cd9f-3a8d-bbd5-5d76732a48e1