# perl基本语法

## **perl**代码的开头

```
#!/usr/bin/perl -w
use strict;
```

## **perl**语言转义以及输出：

```
print("Hello, world\n");
print "Hello, world\n";
print "Hello, world","\n";
print "Hello,
        world","\n";
print 'Hello, world\n',"\n";     #单引号
print \n,"\n";  #结果为SCALAR(0xe80da8)是一个地址，会改变
$a = 10;     #给变量赋值
print "a = $a\n";
print 'a = $a\n';
```

需要注意的是：

- 在“”之间也可以直接增加“，”来直接实现输出的延长。
- “”内的换行还有TAB都会在输出得到显示。
- 单引号和双引号并不等效，双引号可以正常解析一些转义字符与变量，而单引号无法解析会原样输出。

上面的代码输出如下：

```
Hello, world
Hello, world
Hello, world
Hello,
        world
SCALAR(0xe80da8)
a = 10
a = $a\n
```

## 注释

注释可以使用#，也可以用=pod和=cut进行连接。

```
# 这是一个单行注释

=pod 注释
这是一个多行注释
这是一个多行注释
=cut
```

## **Here**文档：

这是一种在命令行shell（如sh、csh、ksh、bash、PowerShell和zsh）和程序语言（像Perl、PHP、Python和Ruby）里定义一个字串的方法。

## 使用概述：

- 必须后接分号，否则编译通不过。
- END可以用任意其它字符代替，只需保证结束标识与开始标识一致。
- 结束标识必须顶格独自占一行(即必须从行首开始，前后不能衔接任何空白和字符)。
- 开始标识可以不带引号号或带单双引号，不带引号与带双引号效果一致，解释内嵌的变量和转义符号，带单引号则不解释内嵌的变量和转义符号。
- 当内容需要内嵌引号（单引号或双引号）时，不需要加转义符，本身对单双引号转义，此处相当于*q*和*qq*的用法。

示例：

```
#!/usr/bin/perl

$a = 10;
$var = <<"EOF";
Here 文档实例，使用双引号。
例如：a = $a
EOF
print "$var\n";

$var = <<'EOF';
Here 文档实例，使用单引号。
例如：a = $a
EOF
print "$var\n";
```

此时会将定义的EOF之前的内容全部输出，还是一样，双引号内会转意，单引号内不会。

```
Here 文档实例，使用双引号。
例如：a = 10

Here 文档实例，使用单引号。
例如：a = $a
```

在双引号中输出特殊字符和C一样使用转义符"\"

# perl数据类型

**Perl** 是一种弱类型语言，所以变量不需要指定类型，**Perl** 解释器会根据上下文自动选择匹配类型。

基本的数据类型：标量、数组、哈希。

## 标量：

这种数据类型的变量可以是数字，字符串，浮点数，不作严格的区分。在使用时在变量的名字前面加上一个 $，表示是标量。例如：

```
$myfirst=123;       #数字123
$mysecond="123";    #字符串123
```

## 数组：

数组变量以字符 @开头，索引从 0 开始，如：$@arr = (1, 2, 3)$

```
@arr=(1,2,3); #数组索引
print $arr[2],"\n"; #索引结果为3
```

```
3
```

## 哈希：

哈希是一个无序的 $key/value$ 对集合。可以使用键作为下标获取值。哈希变量以字符%开头。

```
%h=('a'=>1,'b'=>2);
print $h{'a'},"\n"; #索引结果为1
```

```
1
```

## 数字字面量

```perl
#整型变量与运算
$x = 1217;
if ($x + 116 == 1333) {
    # 执行代码语句块
}
#浮点型变量与运算
$value = 9.01e+21 + 0.01 - 9.01e+21;
print ("第一个值为：", $value, "\n");
# 第一个值为：0
$value = 9.01e+21 - 9.01e+21 + 0.01;
print ("第二个值为:", $value, "\n");
# 第二个值为:0.01
```

## 字符串

**Perl** 双引号和单引号的区别：双引号可以正常解析一些转义字符与变量，而单引号无法解析会原样输出。

## 单引号与双引号定义字符串

```perl
#!/usr/bin/perl
$var='这是一个使用

多行字符串文本

的例子';
print($var);
```

```
这是一个使用

多行字符串文本

的例子
```

| 转义字符 | 含义 |
| --- | --- |
| \\ | 反斜线 |
| ' | 单引号 |
| " | 双引号 |

| 转义字符 | 含义 |
| --- | --- |
| \a | 系统响铃 |
| \b | 退格 |
| \f | 换页符 |
| \n | 换行 |
| \r | 回车 |
| \t | 水平制表符 |
| \v | 垂直制表符 |
| \0nn | 创建八进制格式的数字 |
| \xnn | 创建十六进制格式的数字 |
| \cX | 控制字符，x可以是任何字符 |
| \u | 强制下一个字符为大写 |
| \l | 强制下一个字符为小写 |
| \U | 强制将所有字符转换为大写 |
| \L | 强制将所有的字符转换为小写 |
| \Q | 将到\E为止的非单词（non-word）字符加上反斜线 |
| \E | 结束\L、\U、\Q |

# Perl变量

**Perl** 为每个变量类型设置了独立的命令空间，所以不同类型的变量可以使用相同的名称，你不用担心会发生冲突。 例如： *$foo* 和 *@foo* 是两个不同的变量。

## 创建变量

Perl变量不需要显式声明类型，在变量赋值后，解释器会自动分配匹配的类型空间。

变量使用等号(=)来赋值。

但是，我们可以在程序中使用 **use strict** 语句让所有变量需要强制声明类型。

```
$age = 25;              # 整型
$name = "runoob";       # 字符串
$salary = 1445.50;      # 浮点数
```

## 标量变量

```
#!/usr/bin/perl

$age = 25;              # 整型
$name = "runoob";       # 字符串
$salary = 1445.50;      # 浮点数

print "Age = $age\n";
print "Name = $name\n";
print "Salary = $salary\n";
```

```
Age = 25
Name = runoob
Salary = 1445.5
```

## 数组变量

```
#!/usr/bin/perl

@ages = (25, 30, 40);
@names = ("google", "runoob", "taobao");

print "\$ages[0] = $ages[0]\n";
print "\$ages[1] = $ages[1]\n";
print "\$ages[2] = $ages[2]\n";
print "\$names[0] = $names[0]\n";
print "\$names[1] = $names[1]\n";
print "\$names[2] = $names[2]\n";
```

```
$ages[0] = 25
$ages[1] = 30
$ages[2] = 40
$names[0] = google
$names[1] = runoob
$names[2] = taobao
```

## 哈希变量

```
#!/usr/bin/perl

%data = ('google', 45, 'runoob', 30, 'taobao', 40);

print "\$data{'google'} = $data{'google'}\n";
print "\$data{'runoob'} = $data{'runoob'}\n";
print "\$data{'taobao'} = $data{'taobao'}\n";
```

```
$data{'google'} = 45
$data{'runoob'} = 30
$data{'taobao'} = 40
```

## 变量上下文

上下文是由等号左边的变量类型决定的，等号左边是标量，则是标量上下文，等号左边是列表，则是列表上下文。

Perl 解释器会根据上下文来决定变量的类型。

```
#!/usr/bin/perl

@names = ('google', 'runoob', 'taobao');

@copy = @names;    # 复制数组
$size = @names;    # 数组赋值给标量，返回数组元素个数

print "名字为 : @copy\n";
print "名字数为 : $size\n";
```

*copy*会拷贝数组，而*size*会返回数组元素个数

```
名字为 : google runoob taobao
名字数为 : 3
```

| 序号 | 上下文及描述 |
| --- | --- |
| 1 | 标量 -赋值给一个标量变量，在标量上下文的右侧计算 |
| 2 | 列表 -赋值给一个数组或哈希，在列表上下文的右侧计算。 |

| 序号 | 上下文及描述 |
| --- | --- |
| 3 | 布尔 –布尔上下文是一个简单的表达式计算，查看是否为 true 或 false。 |
| 4 | Void –这种上下文不需要关系返回什么值，一般不需要返回值。 |
| 5 | 插值 –这种上下文只发生在引号内。 |

# Perl中标量的注意事项

---

## Perl 中特殊字符的应用

```
#!/usr/bin/perl
#这些特殊字符是单独的标记，不能写在字符串中
#__FILE__, __LINE__, 和 __PACKAGE__
print "文件名 ". __FILE__ . "\n";
print "行号 " . __LINE__ ."\n";
print "包名 " . __PACKAGE__ ."\n";

# 无法解析
print "__FILE__ __LINE__ __PACKAGE__\n";
```

```
文件名 test.pl
行号 4
包名 main
__FILE__ __LINE__ __PACKAGE__
```

## v字符串

以 **v** 开头,后面跟着一个或多个用句点分隔的整数,会被当作一个字串文本。

```perl
#!/usr/bin/perl

$smile  = v9786;
#真有一个笑脸
$foo    = v102.111.111;
$martin = v77.97.114.116.105.110;

print "smile = $smile\n";
print "foo = $foo\n";
print "martin = $martin\n";
```

```
Wide character in print at learn01.pl line 7.
smile = ☺
foo = foo
martin = Martin
```

# Perl中的数组

## 创建数组

数组变量以 @ 符号开始，元素放在括号内，也可以以 **qw** 开始定义数组。

```perl
@array = (1, 2, 'Hello');
#array=[1,2,Hello]
@array = qw/这是 一个 数组/;
#array=[这是，一个，数组]
```

## 访问数组元素可以逆序

```perl
@sites = qw/google taobao runoob/;
print "$sites[-1]\n";     # 负数，反向读取
```

```
runoob
```

## 数组序列号

起始值 + .. + 结束值，可用于数字和字符

```
#!/usr/bin/perl

@var_10 = (1..10);
@var_20 = (10..20);
@var_abc = ('a'..'z');

print "@var_10\n";    # 输出 1 到 10
print "@var_20\n";    # 输出 10 到 20
print "@var_abc\n";   # 输出 a 到 z
```

```
1 2 3 4 5 6 7 8 9 10
10 11 12 13 14 15 16 17 18 19 20
a b c d e f g h i j k l m n o p q r s t u v w x y z
```

## 数组大小

```
@array = (1,2,3);
print "数组大小: ",scalar @array,"\n";
#scalar @array == 3;
```

```
#!/uer/bin/perl

@array = (1,2,3);
$array[50] = 4;

$size = @array;
$max_index = $#array;

print "数组大小:  $size\n";
print "最大索引: $max_index\n";
```

```
数组大小:  51
最大索引: 50
```

## 添加和删除元素

```
push(@ARRAY, LIST);
#列表的值放到数组的末尾
$ele = pop(@ARRAY);
#删除数组的最后一个值,返回值是这个元素
$ele = shift(@ARRAY);
#弹出数组第一个值，并返回它。数组的索引值也依次减一。
unshift(@ARRAY, LIST);
#将列表放在数组前面，并返回新数组的元素个数。
```

## 数组切割

```
#!/usr/bin/perl

@sites = qw/google taobao runoob weibo qq facebook 网易/;

@sites2 = @sites[3,4,5];

print "@sites2\n";
#@sites2=(weibo,qq,facebook)
```

## 替换数组元素

```
splice @ARRAY, OFFSET [ , LENGTH [ , LIST ] ]
```

参数说明：

- @ARRAY：要替换的数组。
- OFFSET：起始位置。
- LENGTH：替换的元素个数。
- LIST：替换元素列表。

```
#!/usr/bin/perl

@nums = (1..20);
print "替换前 - @nums\n";

splice(@nums, 5, 5, 21..25);
print "替换后 - @nums\n";
```

注意，这里可以超过或少于原有的元素个数进行替换

## 字符串到数组，字符串到数组

```
split [ PATTERN [ , EXPR1 [ , LIMIT ] ] ]
join [EXPR2, LIST]
```

参数说明：

- PATTERN：分隔符，默认为空格。
- EXPR1：指定字符串数。
- LIMIT：如果指定该参数，则返回该数组的元素个数。
- EXPR2：连接符。
- LIST：列表或数组。

```
$var_names = "google,taobao,runoob,weibo";
@names  = split(',', $var_names);
#$names[3]=weibo
$string2 = join( ',', @names );
#google,taobao,runoob,weibo
```

## 排序

```
#!/usr/bin/perl

# 定义数组
@sites = qw(google taobao runoob facebook);
#排序前: google taobao runoob facebook

# 对数组进行排序
@sites = sort(@sites);
#排序后: facebook google runoob taobao
```

注意：数组排序是根据 *ASCII* 数字值来排序。所以我们在对数组进行排序时最好先将每个元素转换为小写后再排序。

## 特殊变量： $[

$[ 表示数组的第一索引值，一般都为 0 ，如果我们将$[设置为 1，则数组的第一个索引值即为 1，第二个为 2，以此类推。

一般情况我们不建议使用特殊变量 $[，在新版 Perl 中，该变量已废弃。

## 合并数组

```
#!/usr/bin/perl

@numbers = (1,3,(4,5,6));
print "numbers = @numbers\n";

@odd = (1,3,5);
@even = (2, 4, 6);
@numbers = (@odd, @even);
print "numbers = @numbers\n";
```

```
numbers = 1 3 4 5 6
numbers = 1 3 5 2 4 6
```

## 从列表中选择元素

一个列表可以当作一个数组使用，在列表后指定索引值可以读取指定的元素。

```
#!/usr/bin/perl

$var = (5,4,3,2,1)[4];
@list = (5,4,3,2,1)[1..3];
@list2 = @list[0..1];
print "var 的值为 = $var\n"
print "list 的值 = @list\n";
print "list2 的值 = @list2\n";
```

```
var 的值为 = 1
list 的值 = 4 3 2
list2 的值 = 4 3
```

### 重点与C语言的不同之处

- 数组的两种定义方式$@array = （）$,$@array = qw/ele1\ ele2\ ele3/$
- 直接给未定义位置的数组索引赋值
- 逆序访问数组
- 顺序数列与字母列
- 直接输出数组
- ...

# Perl哈希

---

### 创建有两种方式：

- 为每个 key 设置 value：

```
$data{'google'} = 'google.com';
```

- 通过列表设置：

```
#!/usr/bin/perl

%data = ('google', 'google.com', 'runoob', 'runoob.com');
%data = ('google'=>'google.com', 'runoob'=>'runoob.com');
#$val = $data{'google'}
%data = (-google=>'google.com', -runoob=>'runoob.com');
#$val = $data{-google}
#索引的方式不一样
```

### 读取哈希的 key 和 value

```
@names = keys %data;
@urls = values %data;
```

这样就读取了data中的所有key或者value

## 检测元素存在

在哈希中读取不存在的 key/value 对，会返回 undefined 值，且在执行时会有警告提醒。为了避免这种情况，我们可以使用 exists 函数来判断key是否存在，存在的时候读取：

```
if( exists($data{'facebook'} ) ){
   print "facebook 的网址为 $data{'facebook'} \n";
}
else
{
   print "facebook 键不存在\n";
}
```

```
facebook 键不存在
```

## 大小

```
@keys = keys %data;
$size = @keys;
print "1 - 哈希大小: $size\n";

@values = values %data;
$size = @values;
print "2 - 哈希大小: $size\n";
#即先转化成数组，再读取数组大小
```

## 添加或删除元素

```
# 添加元素
$data{'facebook'} = 'facebook.com';
@keys = keys %data;
$size = @keys;
print "哈希大小: $size\n";

# 删除哈希中的元素
delete $data{'taobao'};
@keys = keys %data;
$size = @keys;
print "哈希大小: $size\n";
```

### 迭代哈希

关键词是foreach和while：

```
foreach my $key (keys %data){
    print "$data{$key}\n";
}

while(my ($key, $value) = each(%data)){
    print "$data{$key}\n";
}
```

## 条件语句

---

### 经典运算符

**Perl**中数字 **0,** 字符串 **'0'** 、 **"" ,** 空 **list() ,** 和 **undef** 为 **false** ， 其他值均为 **true**。 **true** 前面使用 **!** 或 **not**则返回 **false** 。
可以使用的语句有：

```
if(boolean_expression 1){
   # 在布尔表达式 boolean_expression 1 为 true 执行
}
elsif( boolean_expression 2){
   # 在布尔表达式 boolean_expression 2 为 true 执行
}
else{
   # 布尔表达式的条件都为 false 时执行
}

unless(boolean_expression 1){
   # 在布尔表达式 boolean_expression 1 为 false 执行
}
elsif( boolean_expression 2){
   # 在布尔表达式 boolean_expression 2 为 true 执行
}
else{
   #  没有条件匹配时执行
}

```

```
use Switch;
#需要使用Switch模块
switch(argument){
   case 1            { print "数字 1" }
   case "a"          { print "字符串 a" }
   case [1..10,42]   { print "数字在列表中" }
   case (\@array)    { print "数字在数组中" }
   case /\w+/        { print "正则匹配模式" }
   case qr/\w+/      { print "正则匹配模式" }
   case (\%hash)     { print "哈希" }
   case (\&sub)      { print "子进程" }
   else              { print "不匹配之前的条件" }
}
```

值得注意的是，perl中Switch有一个next的用法可以多个匹配。

```
#!/usr/bin/perl

use Switch;

$var = 10;
@array = (10, 20, 30);
%hash = ('key1' => 10, 'key2' => 20);

switch($var){
   case 10           { print "数字 10\n"; next; }  # 匹配后继续执行
   case "a"          { print "string a" }
   case [1..10,42]   { print "数字在列表中" }
   case (\@array)    { print "数字在数组中" }
   case (\%hash)     { print "在哈希中" }
   else              { print "没有匹配的条件" }
}
```

```
数字 10
数字在列表中
```

三元运算符

```
Exp1 ? Exp2 : Exp3;
```

# perl循环

## 循环的示例

```
while(condition){statement(s);}
until(condition){statement(s);}
for ( init; condition; increment ){statement(s);}
foreach $var (@list) {statement(s);}
do{statement(s);}while(condition);
```

## 循环的控制

| 语句 | 功能 |
| --- | --- |
| next | 停止执行从next语句的下一语句开始到循环体结束标识符之间的语句，转去执行continue语句块，然后再返回到循环体的起始处开始执行下一次循环 |
| last | 退出循环语句块，从而结束循环 |
| continue | continue 语句块通常在条件语句再次判断前执行 |
| redo | 直接转到循环体的第一行开始重复执行本次循环，redo语句之后的语句不再执行，continue语句块也不再执行 |
| goto | 三种 goto 形式：got LABLE，goto EXPR，和 goto &NAME |

# 运算符

| 运算符 | 描述 |
| --- | --- |
| + | 加 |
| - | 减 |
| * | 乘 |
| / | 除 |

| 运算符 | 描述 |
| --- | --- |
| % | 求余 |
| ** | 求幂 |
| 关系运算 | 描述 |
| ==/eq | 值相等 |
| !=/ne | 值不等 |
| <=>/cmp | 判断大小，相等为0，右大为-1.左大为1 |
| >/ge | 大于 |
| </lt | 小于 |
| >=/ge | 大于等于 |
| <=/le | 小于等于 |
| 赋值运算符 | 所有预算均可以+=**= |
| 位运算 | 描述 |
| & | 与 |
| \| | 或 |
| ^ | 异或 |
| ~ | 反 |
| << | 二进制左移，*2 |
| >> | 二进制右移，/2 |
| 逻辑运算 | 描述 |
| and | 与 |
| or | 或 |
| not | 非 |
| && | 与 |
| \|\| | 或 |

## 引号运算与其他运算

| 运算符 | 描述 | 示例 |
| --- | --- | --- |
| q{ } | 为字符串添加单引号 | q{abcd} 结果为 'abcd' |
| qq{} | 为字符串添加双引号 | qq{abcd} 结果为 "abcd" |
| qx{} | 为字符串添加反引号 | qx{abcd} 结果为 `abcd` |

| 运算符 | 描述 | 示例 |
| --- | --- | --- |
| . | 点号 (.) 用于连接两个字符串 | $a="run", $b="oob" ， $a.$b 结果为 "runoob" |
| x | x运算符返回字符串重复的次数 | ('-'x3) 输出为 --- |
| .. | .. 为范围运算符 | (2..5) 输出结果为 (2, 3, 4, 5) |
| ++ | 自增 | |
| -- | 自减 | |
| -> | 指定一个类的方法 | $obj->$a 表示对象 $obj 的 $a 方法 |

```
# 使用 unix 的 date 命令执行
#qx可能有特殊用途
$t = qx{date};
print "qx{date} = $t\n";
```

```
qx{date} = 2016年 6月10日 星期五 16时22分33秒 CST
```