1、概念

Perl 是 Practical Extraction and Report Language 的缩写，可翻译为 "实用报表提取语言"。

Perl 是高级、通用、直译式、动态的程序语言。

Perl 最初的设计者为拉里·沃尔（Larry Wall），于1987年12月18日发表。

Perl 借用了C、sed、awk、shell脚本以及很多其他编程语言的特性。

2、简介

Perl，一种功能丰富的计算机程序语言，运行在超过100种计算机平台上，适用广泛，从大型机到便携设备，从快速原型创建到大规模可扩展开发。

Perl 语言的应用范围很广，除CGI以外，Perl被用于图形编程、系统管理、网络编程、金融、生物以及其他领域。由于其灵活性，Perl被称为脚本语言中的瑞士军刀。

3、优缺点

优点：相比C、Pascal这样的"高级"语言而言，Perl语言直接提供泛型变量、动态数组、Hash表等更加便捷的编程元素。

Perl具有动态语言的强大灵活的特性，并且还从C/C++、Basic、Pascal等语言中分别借鉴了语法规则，从而提供了许多冗余语法。

在统一变量类型和掩盖运算细节方面，Perl做得比其他高级语言(如：Python）更为出色。

由于从其他语言大量借鉴了语法，使得从其他编程语言转到Perl语言的程序员可以迅速上手写程序并完成任务，这使得Perl语言是一门容易用的语言。

Perl 是可扩展的，我们可以通过[CPAN（"the Comprehensive Perl Archive Network"全面的 Perl 存档网络）](http://cpan.perl.org/)中心仓库找到很多我们需要的模块。

Perl 的 [mod\_perl](http://perl.apache.org/) 的模块允许 Apache web 服务器使用 Perl 解释器。

缺点：也正是因为Perl的灵活性和"过度"的冗余语法，也因此获得了仅写（write-only）的"美誉"，因为Perl程序可以写得很随意（例如，变量不经声明就可以直接使用），但是可能少写一些字母就会得到意想不到的结果（而不报错），许多Perl程序的代码令人难以阅读，实现相同功能的程序代码长度可以相差十倍百倍，这就令程序的维护者（甚至是编写者）难以维护。

同样的，因为Perl这样随意的特点，可能会导致一些Perl程序员遗忘语法，以至于不得不经常查看Perl手册。

建议的解决方法是在程序里使用use strict;以及use warnings;，并统一代码风格，使用库，而不是自己使用"硬编码"。Perl同样可以将代码书写得像Python或Ruby等语言一样优雅。

很多时候，perl.exe进程会占用很多的内存空间，虽然只是一时，但是感觉不好。

perl -v 可查看perl版本

4、安装

**Unix 和 Linux 安装 Perl**

Unix/Linux 系统上 Perl 安装步骤如下：

通过浏览器打开 [http://www.perl.org/get.html。](http://www.perl.org/get.html)

下载适用于 Unix/Linux 的源码包。

下载 **perl-5.x.y.tar.gz** 文件后执行以下操作。

$ tar -xzf perl-5.x.y.tar.gz
$ cd perl-5.x.y
$ ./Configure -de
$ make
$ make test
$ make install

接下来我们如果 **perl -v** 命令查看是否安装成功。

安装成功后，Perl 的安装路径为 */usr/local/bin* ，库安装在 */usr/local/lib/perlXX*, XX 为版本号。

5、正式学习

[root@localhost soft]# perl -e 'print "Hello, World\n"'

Hello, World

**perl 中的单引号和双引号：**

（1）双中有双，单中有单都需要 **\** 转义。

（2）双中有单或单中有双均不需要转义。

（3）单引号直接了当，引号内是什么就显示什么，双引号则需要考虑转义或变量替换等。

数据类型

**标量**

标量是Perl语言中最简单的一种数据类型。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**

数组变量以字符"@"开头，索引从0开始，如：@arr=(1,2,3)

@arr=(1,2,3)

**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);

Perl 语言中常用的一些转义字符如下表所示：

|  |  |
| --- | --- |
| **转义字符** | **含义** |
| \\ | 反斜线 |
| \' | 单引号 |
| \" | 双引号 |
| \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的三个基本的数据类型：标量、数组、哈希。

**数组变量**

数组是用于存储一个有序的标量值的变量。

数组 @ 开始。

要访问数组的变量，可以使用美元符号($)+变量名，并指定下标来访问

**哈希变量**

哈希是一个 **key/value** 对的集合。

哈希 % 开始。

如果要访问哈希值，可以使用 **$ + {key}** 格式来访问：

**变量上下文**

所谓上下文：指的是表达式所在的位置。

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

|  |  |
| --- | --- |
| 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。 |
| 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。 |

**特殊字符**

以下我们将演示 Perl 中特殊字符的应用，如 \_\_FILE\_\_, \_\_LINE\_\_, 和 \_\_PACKAGE\_\_ 分别表示当前执行脚本的文件名，行号，包名。

**注意**： **\_\_** 是两条下划线，**\_\_FILE\_\_** 前后各两条下划线。

这些特殊字符是单独的标记，不能写在字符串中，

**实例**

#!/usr/bin/perl
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";

执行以上程序，输出结果为：

文件名 test.pl
行号 4
包名 main
\_\_FILE\_\_ \_\_LINE\_\_ \_\_PACKAGE\_\_

**Perl 数组**

Perl 数组一个是存储标量值的列表变量，变量可以是不同类型。

数组变量以 @ 开头。访问数组元素使用 **$ + 变量名称 + [索引值]** 格式来读取，实例如下：

**实例**

#!/usr/bin/perl
@hits = (25, 30, 40);
@names = ("google", "runoob", "taobao");
print "\$hits[0] = $hits[0]\n";
print "\$hits[1] = $hits[1]\n";
print "\$hits[2] = $hits[2]\n";
print "\$names[0] = $names[0]\n";
print "\$names[1] = $names[1]\n";
print "\$names[2] = $names[2]\n";

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。

@array = (1, 2, 'Hello');
@array = qw/这是 一个 数组/;

**数组序列号**

Perl 提供了可以按序列输出的数组形式，格式为 **起始值 + .. + 结束值**，实例如下：

**实例**

#!/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

**添加和删除数组元素**

Perl 提供了一些有用的函数来添加和删除数组元素。

如果你之前没有编程经验，可能会问什么是函数，其实我们之前使用的 **print** 即是一个输出函数。

下表列出了数组中常用的操作函数：

|  |  |
| --- | --- |
| **序号** | **类型和描述** |
| 1 | push @ARRAY, LIST将列表的值放到数组的末尾 |
| 2 | pop @ARRAY弹出数组最后一个值，并返回它 |
| 3 | 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。 |
| 4 | unshift @ARRAY, LIST将列表放在数组前面，并返回新数组的元素个数 |

**切割数组**

我们可以切割一个数组，并返回切割后的新数组：

**实例**

#!/usr/bin/perl
@sites = qw/google taobao runoob weibo qq facebook 网易/;
@sites2 = @sites[3,4,5];
print "@sites2\n";

数组索引需要指定有效的索引值，可以是正数后负数，每个索引值使用逗号隔开。

如果是连续的索引，可以使用 **..** 来表示指定范围：

**实例**

#!/usr/bin/perl
@sites = qw/google taobao runoob weibo qq facebook 网易/;
@sites2 = @sites[3..5];
print "@sites2\n";

**替换数组元素**

Perl 中数组元素替换使用 splice() 函数，语法格式如下：

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。

以下实例从第6个元素开始替换数组中的5个元素:

**实例**

#!/usr/bin/perl
@nums = (1..20);
print "替换前 - @nums\n";
splice(@nums, 5, 5, 21..25);
print "替换后 - @nums\n";

**将字符串转换为数组**

Perl 中将字符串转换为数组使用 split() 函数，语法格式如下：

split [ PATTERN [ , EXPR [ , LIMIT ] ] ]

参数说明：

PATTERN：分隔符，默认为空格。

EXPR：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。

**实例**

#!/usr/bin/perl
# 定义字符串
$var\_test = "runoob";
$var\_string = "www-runoob-com";
$var\_names = "google,taobao,runoob,weibo";
# 字符串转为数组
@test = split('', $var\_test);
@string = split('-', $var\_string);
@names = split(',', $var\_names);

**数组排序**

Perl 中数组排序使用 sort() 函数，语法格式如下：

sort [ SUBROUTINE ] LIST

参数说明：

SUBROUTINE：指定规则。

LIST：列表或数组。

**实例**

#!/usr/bin/perl
# 定义数组
@sites = qw(google taobao runoob facebook);
print "排序前: @sites\n";
# 对数组进行排序
@sites = sort(@sites);
print "排序前: @sites\n";

执行以上程序，输出结果为：

排序前: google taobao runoob facebook
排序前: facebook google runoob taobao

**合并数组**

数组的元素是以逗号来分割，我们也可以使用逗号来合并数组，如下所示：

**实例**

#!/usr/bin/perl
@numbers = (1,3,(4,5,6));
print "numbers = @numbers\n";

执行以上程序，输出结果为：

numbers = 1 3 4 5 6

也可以在数组中嵌入多个数组，并合并到主数组中：

**实例**

#!/usr/bin/perl
@odd = (1,3,5);
@even = (2, 4, 6);
@numbers = (@odd, @even);
print "numbers = @numbers\n";

执行以上程序，输出结果为：

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，如下所示：

**实例**

#!/usr/bin/perl
$var = (5,4,3,2,1)[4];
print "var 的值为 = $var\n"

执行以上程序，输出结果为：

var 的值为 = 1

同样我们可以在数组中使用 **..** 来读取指定范围的元素：

**实例**

#!/usr/bin/perl
@list = (5,4,3,2,1)[1..3];
print "list 的值 = @list\n";

**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，如下所示：

**实例**

#!/usr/bin/perl
$var = (5,4,3,2,1)[4];
print "var 的值为 = $var\n"

执行以上程序，输出结果为：

var 的值为 = 1

同样我们可以在数组中使用 **..** 来读取指定范围的元素：

**实例**

#!/usr/bin/perl
@list = (5,4,3,2,1)[1..3];
print "list 的值 = @list\n";

执行以上程序，输出结果为：

list 的值 = 4 3 2

**Perl 哈希**

哈希是 **key/value** 对的集合。

Perl中哈希变量以百分号 (%) 标记开始。

访问哈希元素格式：**${key}**。

访问哈希元素格式：**${key}**，实例如下：

**实例**

#!/usr/bin/perl
%data = ('google'=>'google.com', 'runoob'=>'runoob.com', 'taobao'=>'taobao.com');
print "\$data{'google'} = $data{'google'}\n";
print "\$data{'runoob'} = $data{'runoob'}\n";
print "\$data{'taobao'} = $data{'taobao'}\n";

列表中第一个元素为 key，第二个为 value。

%data = ('google', 'google.com', 'runoob', 'runoob.com', 'taobao', 'taobao.com');

也可以使用 **=>** 符号来设置 key/value:

%data = ('google'=>'google.com', 'runoob'=>'runoob.com', 'taobao'=>'taobao.com');

以下实例是上面实例的变种，使用 **-** 来代替引号：

%data = (-google=>'google.com', -runoob=>'runoob.com', -taobao=>'taobao.com');

**读取哈希值**

你可以像数组一样从哈希中提取值。

哈希值提取到数组语法格式：**@{key1,key2}**。

**实例**

#!/uer/bin/perl
%data = (-taobao => 45, -google => 30, -runoob => 40);
@array = @data{-taobao, -runoob};
print "Array : @array\n";

执行以上程序，输出结果为：

Array : 45 40

**读取哈希的 key 和 value**

**读取所有key**

我们可以使用 **keys** 函数读取哈希所有的键，语法格式如下：

keys %HASH

该函数返回所有哈希的所有 key 的数组。

**实例**

#!/usr/bin/perl
%data = ('google'=>'google.com', 'runoob'=>'runoob.com', 'taobao'=>'taobao.com');
@names = keys %data;
print "$names[0]\n";
print "$names[1]\n";
print "$names[2]\n";

类似的我们可以使用 **values** 函数来读取哈希所有的值,语法格式如下：

values %HASH

该函数返回所有哈希的所有 value 的数组。

**实例**

#!/usr/bin/perl
%data = ('google'=>'google.com', 'runoob'=>'runoob.com', 'taobao'=>'taobao.com');
@urls = values %data;
print "$urls[0]\n";
print "$urls[1]\n";
print "$urls[2]\n";

执行以上程序，输出结果为：

taobao.com
runoob.com
google.com

Perl 教程

[Perl 教程](http://www.runoob.com/perl/perl-tutorial.html)[Perl 简介](http://www.runoob.com/perl/perl-intro.html)[Perl 环境安装](http://www.runoob.com/perl/perl-environment.html)[Perl 基础语法](http://www.runoob.com/perl/perl-syntax.html)[Perl 数据类型](http://www.runoob.com/perl/perl-data-types.html)[Perl 变量](http://www.runoob.com/perl/perl-variables.html)[Perl 标量](http://www.runoob.com/perl/perl-scalars.html)[Perl 数组](http://www.runoob.com/perl/perl-arrays.html)[**Perl 哈希**](http://www.runoob.com/perl/perl-hashes.html)[Perl 条件语句](http://www.runoob.com/perl/perl-conditions.html)[Perl 循环](http://www.runoob.com/perl/perl-loops.html)[Perl 运算符](http://www.runoob.com/perl/perl-operators.html)[Perl 时间日期](http://www.runoob.com/perl/perl-date-time.html)[Perl 子程序(函数)](http://www.runoob.com/perl/perl-subroutines.html)[Perl 引用](http://www.runoob.com/perl/perl-references.html)[Perl 格式化输出](http://www.runoob.com/perl/perl-formats.html)[Perl 文件操作](http://www.runoob.com/perl/perl-files.html)[Perl 目录操作](http://www.runoob.com/perl/perl-directories.html)[Perl 错误处理](http://www.runoob.com/perl/perl-error-handling.html)[Perl 特殊变量](http://www.runoob.com/perl/perl-special-variables.html)[Perl 正则表达式](http://www.runoob.com/perl/perl-regular-expressions.html)[Perl 发送邮件](http://www.runoob.com/perl/perl-sending-email.html)[Perl Socket 编程](http://www.runoob.com/perl/perl-socket-programming.html)[Perl 面向对象](http://www.runoob.com/perl/perl-object-oriented.html)[Perl 数据库连接](http://www.runoob.com/perl/perl-database-access.html)[Perl CGI编程](http://www.runoob.com/perl/perl-cgi-programming.html)[Perl 包和模块](http://www.runoob.com/perl/perl-packages-modules.html)[Perl 进程管理](http://www.runoob.com/perl/perl-process-management.html)[Perl POD 文档](http://www.runoob.com/perl/perl-embedded-documentation.html)

 [Perl 数组](http://www.runoob.com/perl/perl-arrays.html)

[Perl 条件语句](http://www.runoob.com/perl/perl-conditions.html)

**Perl 哈希**

哈希是 **key/value** 对的集合。

Perl中哈希变量以百分号 (%) 标记开始。

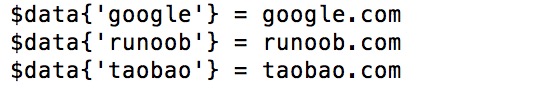
访问哈希元素格式：**${key}**。

以下是一个简单的哈希实例：

**实例**

#!/usr/bin/perl
%data = ('google', 'google.com', 'runoob', 'runoob.com', 'taobao', 'taobao.com');
print "\$data{'google'} = $data{'google'}\n";
print "\$data{'runoob'} = $data{'runoob'}\n";
print "\$data{'taobao'} = $data{'taobao'}\n";

执行以上程序，输出结果为：



**创建哈希**

创建哈希可以通过以下两种方式：

**一、为每个 key 设置 value**

$data{'google'} = 'google.com';
$data{'runoob'} = 'runoob.com';
$data{'taobao'} = 'taobao.com';

**二、通过列表设置**

列表中第一个元素为 key，第二个为 value。

%data = ('google', 'google.com', 'runoob', 'runoob.com', 'taobao', 'taobao.com');

也可以使用 **=>** 符号来设置 key/value:

%data = ('google'=>'google.com', 'runoob'=>'runoob.com', 'taobao'=>'taobao.com');

以下实例是上面实例的变种，使用 **-** 来代替引号：

%data = (-google=>'google.com', -runoob=>'runoob.com', -taobao=>'taobao.com');

使用这种方式 key 不能出现空格，读取元素方式为：

$val = $data{-google}
$val = $data{-runoob}

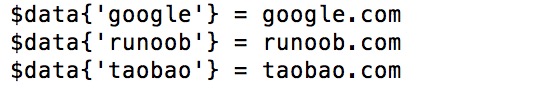
**访问哈希元素**

访问哈希元素格式：**${key}**，实例如下：

**实例**

#!/usr/bin/perl
%data = ('google'=>'google.com', 'runoob'=>'runoob.com', 'taobao'=>'taobao.com');
print "\$data{'google'} = $data{'google'}\n";
print "\$data{'runoob'} = $data{'runoob'}\n";
print "\$data{'taobao'} = $data{'taobao'}\n";

执行以上程序，输出结果为：



**读取哈希值**

你可以像数组一样从哈希中提取值。

哈希值提取到数组语法格式：**@{key1,key2}**。

**实例**

#!/uer/bin/perl
%data = (-taobao => 45, -google => 30, -runoob => 40);
@array = @data{-taobao, -runoob};
print "Array : @array\n";

执行以上程序，输出结果为：

Array : 45 40

**读取哈希的 key 和 value**

**读取所有key**

我们可以使用 **keys** 函数读取哈希所有的键，语法格式如下：

keys %HASH

该函数返回所有哈希的所有 key 的数组。

**实例**

#!/usr/bin/perl
%data = ('google'=>'google.com', 'runoob'=>'runoob.com', 'taobao'=>'taobao.com');
@names = keys %data;
print "$names[0]\n";
print "$names[1]\n";
print "$names[2]\n";

执行以上程序，输出结果为：

taobao
google
runoob

类似的我们可以使用 **values** 函数来读取哈希所有的值,语法格式如下：

values %HASH

该函数返回所有哈希的所有 value 的数组。

**实例**

#!/usr/bin/perl
%data = ('google'=>'google.com', 'runoob'=>'runoob.com', 'taobao'=>'taobao.com');
@urls = values %data;
print "$urls[0]\n";
print "$urls[1]\n";
print "$urls[2]\n";

执行以上程序，输出结果为：

taobao.com
runoob.com
google.com

**检测元素是否存在**

如果你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：

**实例**

#!/usr/bin/perl
%data = ('google'=>'google.com', 'runoob'=>'runoob.com', 'taobao'=>'taobao.com');
if( exists($data{'facebook'} ) ){
print "facebook 的网址为 $data{'facebook'} \n";
}
else
{
print "facebook 键不存在\n";
}

执行以上程序，输出结果为：

facebook 键不存在

以上代码中我们使用了 **IF...ELSE** 语句，在后面的章节我们会具体介绍。

**获取哈希大小**

哈希大小为元素的个数，我们可以通过先获取 key 或 value 的所有元素数组，再计算数组元素多少来获取哈希的大小，实例如下：

**实例**

#!/usr/bin/perl
%data = ('google'=>'google.com', 'runoob'=>'runoob.com', 'taobao'=>'taobao.com');
@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";

执行以上程序，输出结果为：

1 - 哈希大小: 3
2 - 哈希大小: 3

**哈希中添加或删除元素**

添加 key/value 对可以通过简单的赋值来完成。但是删除哈希元素你需要使用 **delete** 函数：

**实例**

#!/usr/bin/perl
%data = ('google'=>'google.com', 'runoob'=>'runoob.com', 'taobao'=>'taobao.com');
@keys = keys %data;
$size = @keys;
print "1 - 哈希大小: $size\n";
# 添加元素
$data{'facebook'} = 'facebook.com';
@keys = keys %data;
$size = @keys;
print "2 - 哈希大小: $size\n";
# 删除哈希中的元素
delete $data{'taobao'};
@keys = keys %data;
$size = @keys;
print "3 - 哈希大小: $size\n";

**获取哈希大小**

哈希大小为元素的个数，我们可以通过先获取 key 或 value 的所有元素数组，再计算数组元素多少来获取哈希的大小，实例如下：

**实例**

#!/usr/bin/perl
%data = ('google'=>'google.com', 'runoob'=>'runoob.com', 'taobao'=>'taobao.com');
@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";

**条件语句**

Perl 提供了下拉的条件语句：

|  |  |
| --- | --- |
| **语句** | **描述** |
| if 语句 | 一个 if 语句 由一个布尔表达式后跟一个或多个语句组成。 |
| if...else 语句 | 一个 if 语句 后可跟一个可选的 else 语句，else 语句在布尔表达式为假时执行。 |
| if...elsif...else 语句 | 您可以在一个 if 语句后可跟一个可选的 elsif 语句，然后再跟另一个 else 语句。 |
| unless 语句 | 一个 unless 语句 由一个布尔表达式后跟一个或多个语句组成。 |
| unless...else 语句。 | 一个 unless 语句 后可跟一个可选的 else 语句。 |
| unless...elsif..else statement | 一个 unless 语句 后可跟一个可选的 elsif 语句，然后再跟另一个 else 语句。 |
| switch 语句 | 在最新版本的 Perl 中，我们可以使用 switch 语句。它根据不同的值执行对应的代码块。 |

**三元运算符 ? :**

我们可以使用 **条件运算 ? :** 来简化 **if...else** 语句的操作。通常格式为：

Exp1 ? Exp2 : Exp3;

**循环**

Perl 语言提供了以下几种循环类型:

|  |  |
| --- | --- |
| **循环类型** | **描述** |
| while 循环 | 当给定条件为 true 时，重复执行语句或语句组。循环主体执行之前会先测试条件。 |
| until 循环 | 重复执行语句或语句组，直到给定的条件为 true。 循环主体执行之前会先测试条件。 |
| for 循环 | 多次执行一个语句序列，简化管理循环变量的代码。 |
| foreach 循环 | foreach 循环用于迭代一个列表或集合变量的值。 |
| do...while 循环 | 除了它是在循环主体结尾测试条件外，其他与 while 语句类似。 |
| 嵌套循环 | 您可以在 while、for 或 do..while 循环内使用一个或多个循环。 |

**循环控制语句**

循环控制语句改变了代码的执行顺序，通过它你可以实现代码的跳转。

Perl 提供了下列的循环控制语句:

|  |  |
| --- | --- |
| **控制语句** | **描述** |
| next 语句 | 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 语句 | 退出循环语句块，从而结束循环 |
| continue 语句 | continue 语句块通常在条件语句再次判断前执行。 |
| redo 语句 | redo 语句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 语句 | Perl 有三种 goto 形式：got LABLE，goto EXPR，和 goto &NAME。 |

**无限循环**

如果条件永远不为 false，则循环将变成无限循环。

for 循环在传统意义上可用于实现无限循环。

由于构成循环的三个表达式中任何一个都不是必需的，您可以将某些条件表达式留空来构成一个无限循环。

**实例**

#!/usr/bin/perl
for( ; ; )
{
printf "循环会无限执行。\n";
}

*你可以按下 Ctrl + C 键来终止循环。*

当条件表达式不存在时，它被假设为 true 。您也可以设置一个初始值和增量表达式，但是一般情况下，Perl 程序员偏向于使用 for(;;) 结构来表示一个无限循环。

**引号运算**

Perl 引号运算符如下表所示。

|  |  |  |
| --- | --- | --- |
| **运算符** | **描述** | **实例** |
| 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 |

**其他运算符**

除了以上我们提到的运算符外，Perl 还支持以下运算符：

|  |  |  |
| --- | --- | --- |
| **运算符** | **描述** | **实例** |
| . | 点号 (.) 用于连接两个字符串。 | 如果 $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。 | ('-' x 3) 输出为 ---。 |
| .. | .. 为范围运算符。 | (2..5) 输出结果为 (2, 3, 4, 5) |
| ++ | 自增运算符，整数值增加 1 | $a =10, $a++ will 输出为 11 |
| -- | 自减运算符，整数值减少 1 | $a =10, $a-- 输出为 9 |
| -> | 箭号用于指定一个类的方法 | $obj->$a 表示对象 $obj 的 $a 方法。 |

**运算符优先级**

下表列出了 Perl 语言的运算符优先级：

|  |  |
| --- | --- |
| **运算符符** | **结合性** |
| ++, -- | 无 |
| -, ~, ! | 从右到左 |
| \*\* | 从右到左 |
| =~, !~ | 从左到右 |
| \*, /, %, x | 从左到右 |
| +, -, . | 从左到右 |
| <<, >> | 从左到右 |
| -e, -r, | 无 |
| <, <=, >, >=, lt, le, gt, ge | 从左到右 |
| ==, !=, <=>, eq, ne, cmp | 从左到右 |
| & | 从左到右 |
| |, ^ | 从左到右 |
| && | 从左到右 |
| || | 从左到右 |
| .. | 从左到右 |
| ? and : | 从右到左 |
| =, +=, -=, \*=, | 从右到左 |
| 其他 | |
| , | 从左到右 |
| not | 从左到右 |
| and | 从左到右 |
| or, xor | 从左到右 |

Perl中处理时间的函数有如下几种：

**1、time() 函数：**返回从1970年1月1日起累计的秒数

**2、localtime() 函数：**获取本地时区时间

**3、gmtime() 函数：**获取格林威治时间

**当前时间和日期**

接下来让我们看下 **localtime()** 函数，该函数在没有参数的情况下返回当前的时间和日期。

以下 9 个符号代表不同的时间日期参数：

sec, # 秒， 0 到 61
min, # 分钟， 0 到 59
hour, # 小时， 0 到 24
mday, # 天， 1 到 31
mon, # 月， 0 到 11
year, # 年，从 1900 开始
wday, # 星期几，0-6,0表示周日
yday, # 一年中的第几天,0-364,365
isdst # 如果夏令时有效，则为真

time() localtiime() gmtime()

**格式化日期和时间**

我们可以使用 localtime() 函数的 9 个时间元素来输出需要制定的格式时间。格式化输出使用 printf() 函数：

**实例**

#!/usr/bin/perl
($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) = localtime();
printf("格式化时间：HH:MM:SS\n");
printf("%02d:%02d:%02d", $hour, $min, $sec);

以上实例执行输出结果为：

格式化时间：HH:MM:SS
11:35:23

**POSIX 函数 strftime()**

函数 strftime() 可以将时间格式化为我们想要的格式。

下表列出了一些格式化的符号，\* 号表示想要依赖本地时间：

|  |  |  |
| --- | --- | --- |
| **符号** | **描述** | **实例** |
| %a | 星期几的简称（ Sun..Sat） \* | Thu |
| %A | 星期几的全称（ Sunday..Saturday） \* | Thursday |
| %b | 月的简称（Jan..Dec） \* | Aug |
| %B | 月的全称（January..December） \* | August |
| %c | 日期和时间 \* | Thu Aug 23 14:55:02 2001 |
| %C | 年份除于100，并取整 (00-99) | 20 |
| %d | 一个月的第几天 (01-31) | 23 |
| %D | 日期, MM/DD/YY 相等于%m/%d/%y | 08/23/01 |
| %e | 一个月的第几天，使用空格填充个位数 ( 1-31) | 23 |
| %F | YYYY-MM-DD 的简写类似于 %Y-%m-%d | 2001-08-23 |
| %g | 年份的最后两位数 (00-99) | 01 |
| %g | 年 | 2001 |
| %h | 月的简称 \* (和%b选项相同) | Aug |
| %H | 24 小时制 (00-23) | 14 |
| %I | 12 小时制 (01-12) | 02 |
| %j | 一年的第几天 (001-366) | 235 |
| %m | 月 (01-12) | 08 |
| %M | 分钟 (00-59) | 55 |
| %n | 新行 ('\n') |  |
| %p | 显示出AM或PM | PM |
| %r | 时间（hh：mm：ss AM或PM），12小时 \* | 02:55:02 pm |
| %R | 24 小时 HH:MM 时间格式,相等于 %H:%M | 14:55 |
| %S | 秒数 (00-61) | 02 |
| %t | 水平制表符 ('\t') |  |
| %T | 时间（24小时制）（hh:mm:ss），相等于%H:%M:%S | 14:55 |
| %u | ISO 8601 的星期几格式，星期一为1 (1-7) | 4 |
| %U | 一年中的第几周，星期天为第一天(00-53) | 33 |
| %V | ISO 8601 第几周 (00-53) | 34 |
| %w | 一个星期的第几天（0代表星期天） (0-6) | 4 |
| %W | 一年的第几个星期，星期一为第一天 (00-53) | 34 |
| %x | 显示日期的格式（mm/dd/yy） \* | 08/23/01 |
| %X | 显示时间格式 \* | 14:55:02 |
| %y | 年，两位数 (00-99) | 01 |
| %Y | 年 | 2001 |
| %z | ISO 8601与UTC的时区偏移(1 minute=1, 1 hour=100) | +100 |
| %Z | 当前时区的名称,如"中国标准时间" \* | CDT |
| %% | % 符号 | % |

**实例**

#!/usr/bin/perl
use POSIX qw(strftime);
$datestring = strftime "%Y-%m-%d %H:%M:%S", localtime;
printf("时间日期 - $datestring\n");
# GMT 格式化时间日期
$datestring = strftime "%Y-%m-%d %H:%M:%S", gmtime;
printf("时间日期 - $datestring\n");

以上实例执行输出结果为：

时间日期 - 2016-06-12 12:15:13
时间日期 - 2016-06-12 04:15:13

**Perl 子程序(函数)**

Perl 子程序也就是用户定义的函数。

Perl 子程序即执行一个特殊任务的一段分离的代码，它可以使减少重复代码且使程序易读。

Perl 子程序可以出现在程序的任何地方，语法格式如下：

sub subroutine{
statements;
}

调用子程序语法格式：

subroutine( 参数列表 );

在 Perl 5.0 以下版本调用子程序方法如下：

&subroutine( 参数列表 );

在新版本上，虽然也支持该调用方法，但不推荐使用。

接下来我们来看一个简单是实例：

**实例**

#!/usr/bin/perl
# 函数定义
sub Hello{
print "Hello, World!\n";
}
# 函数调用
Hello();

执行以上程序，输出结果为：

Hello, World!

**向子程序传递哈希**

当向子程序传递哈希表时，它将复制到 @\_ 中，哈希表将被展开为键/值组合的列表。

**实例**

#!/usr/bin/perl
# 方法定义
sub PrintHash{
my (%hash) = @\_;
foreach my $key ( keys %hash ){
my $value = $hash{$key};
print "$key : $value\n";
}
}
%hash = ('name' => 'runoob', 'age' => 3);
# 传递哈希
PrintHash(%hash);

**子程序的私有变量**

默认情况下，Perl 中所有的变量都是全局变量，这就是说变量在程序的任何地方都可以调用。

如果我们需要设置私有变量，可以使用 **my** 操作符来设置。

**my** 操作符用于创建词法作用域变量，通过 **my** 创建的变量，存活于声明开始的地方，直到闭合作用域的结尾。

闭合作用域指的可以是一对花括号中的区域，可以是一个文件，也可以是一个 if, while, for, foreach, eval字符串

**实例**

#!/usr/bin/perl
# 全局变量
$string = "Hello, World!";
# 函数定义
sub PrintHello{
# PrintHello 函数的私有变量
my $string;
$string = "Hello, Runoob!";
print "函数内字符串：$string\n";
}
# 调用函数
PrintHello();
print "函数外字符串：$string\n";

**变量的临时赋值**

我们可以使用 local 为全局变量提供临时的值，在退出作用域后将原来的值还回去。

local 定义的变量不存在于主程序中，但存在于该子程序和该子程序调用的子程序中。定义时可以给其赋值，如：

**实例**

#!/usr/bin/perl
# 全局变量
$string = "Hello, World!";
sub PrintRunoob{
# PrintHello 函数私有变量
local $string;
$string = "Hello, Runoob!";
# 子程序调用的子程序
PrintMe();
print "PrintRunoob 函数内字符串值：$string\n";
}
sub PrintMe{
print "PrintMe 函数内字符串值：$string\n";
}
sub PrintHello{
print "PrintHello 函数内字符串值：$string\n";
}
# 函数调用
PrintRunoob();
PrintHello();
print "函数外部字符串值：$string\n";

以上程序执行输出结果为：

PrintMe 函数内字符串值：Hello, Runoob!
PrintRunoob 函数内字符串值：Hello, Runoob!
PrintHello 函数内字符串值：Hello, World!
函数外部字符串值：Hello, World!

**Perl 引用**

引用就是指针，Perl 引用是一个标量类型可以指向变量、数组、哈希表（也叫关联数组）甚至子程序，可以应用在程序的任何地方。

如果你不能确定变量类型，你可以使用 **ref** 来判断，返回值列表如下，如果没有以下的值返回 false：

SCALAR
ARRAY
HASH
CODE
GLOB
REF

实例如下：

**实例**

#!/usr/bin/perl
$var = 10;
$r = \$var;
print "r 的引用类型 : ", ref($r), "\n";
@var = (1, 2, 3);
$r = \@var;
print "r 的引用类型 : ", ref($r), "\n";
%var = ('key1' => 10, 'key2' => 20);
$r = \%var;
print "r 的引用类型 : ", ref($r), "\n";

执行以上实例执行结果为：

r 的引用类型 : SCALAR
r 的引用类型 : ARRAY
r 的引用类型 : HASH

**循环引用**

循环引用在两个引用相互包含时出现。你需要小心使用，不然会导致内存泄露，如下实例：

**实例**

#!/usr/bin/perl
my $foo = 100;
$foo = \$foo;
print "Value of foo is : ", $$foo, "\n";

**Perl 正则表达式**

正则表达式(regular expression)描述了一种字符串匹配的模式，可以用来检查一个串是否含有某种子串、将匹配的子串做替换或者从某个串中取出符合某个条件的子串等。

Perl语言的正则表达式功能非常强大，基本上是常用语言中最强大的，很多语言设计正则式支持的时候都参考Perl的正则表达式。

Perl的正则表达式的三种形式，分别是匹配，替换和转化:

匹配：m//（还可以简写为//，略去m）

替换：s/

转化：tr/

这三种形式一般都和 **=~** 或 **!~** 搭配使用， =~ 表示相匹配，!~ 表示不匹配。

**模式匹配修饰符**

模式匹配有一些常用的修饰符，如下表所示：

|  |  |
| --- | --- |
| **修饰符** | **描述** |
| i | 忽略模式中的大小写 |
| m | 多行模式 |
| o | 仅赋值一次 |
| s | 单行模式，"."匹配"\n"（默认不匹配） |
| x | 忽略模式中的空白 |
| g | 全局匹配 |
| cg | 全局匹配失败后，允许再次查找匹配串 |

**正则表达式变量**

perl处理完后会给匹配到的值存在三个特殊变量名:

**$`:**匹配部分的前一部分字符串

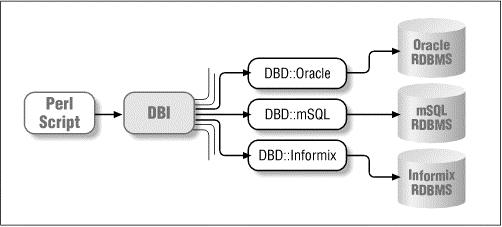
**$&:**匹配的字符串

**$':**还没有匹配的剩余字符串

如果将这三个变量放在一起,你将得到原始字符串

**DBI 结构**

DBI 和具体数据库平台无关，我们可以将其应用在Oracle, MySQL 或 Informix, 等数据库中。



图表中 DBI 获取所有 API（Application Programming Interface：应用程序接口） 发送过来的 SQL 数据，然后分发到对应的驱动上执行，最后再获取数据返回。

**变量名约定**

以下设置了比较常用的变量名命名方法：

$dsn 驱动程序对象的句柄
$dbh 一个数据库对象的句柄
$sth 一个语句或者一个查询对象的句柄
$h 通用的句柄 ($dbh, $sth, 或 $drh)，依赖于上下文
$rc 操作代码返回的布什值（true 或 false）
$rv 操作代码返回的整数值
@ary 查询返回的一行值的数组（列表）
$rows 操作代码返回的行数值
$fh 文件句柄
undef NULL 值表示未定义
\%attr 引用属性的哈希值并传到方法上

**实例**

#!/usr/bin/perl -w
use strict;
use DBI;
my $host = "localhost"; # 主机地址
my $driver = "mysql"; # 接口类型 默认为 localhost
my $database = "RUNOOB"; # 数据库
# 驱动程序对象的句柄
my $dsn = "DBI:$driver:database=$database:$host";
my $userid = "root"; # 数据库用户名
my $password = "123456"; # 数据库密码
# 连接数据库
my $dbh = DBI->connect($dsn, $userid, $password ) or die $DBI::errstr;
my $sth = $dbh->prepare("SELECT \* FROM Websites"); # 预处理 SQL 语句
$sth->execute(); # 执行 SQL 操作
# 注释这部分使用的是绑定值操作
# $alexa = 20;
# my $sth = $dbh->prepare("SELECT name, url
# FROM Websites
# WHERE alexa > ?");
# $sth->execute( $alexa ) or die $DBI::errstr;
# 循环输出所有数据
while ( my @row = $sth->fetchrow\_array() )
{
print join('\t', @row)."\n";
}
$sth->finish();
$dbh->disconnect();