🡪

Perl的模块使用.pm作为扩展名。

🡪

在Perl脚本中使用use引入模块，修改Perl模块引入的路径可以在环境变量中设置PERL5LIB或者在PERL脚本中动态使用use lib路径改变，或者加入路径到@INC中。

🡪

调用模块中的方法使用模块名：：方法名。

🡪

Perl中的Try Catch 语句来捕获以及处理异常，需要注意的是需要引入Try: Tiny模块才可以使用。

🡪

Perl在子函数中返回时候，可以返回字典，数组本身，也可以返回字典引用，或者数组引用。

🡪

在perl中用 {} 修饰变量名的作用是限制变量名，从而不引起歧义。

例如：

sub test {  
    my $head = "abc";  
    my $tail = "def";  
    my $full = "${head}\_${tail}";  
    print $full, "\n";  
}

直接写成下面这样，在strict模式下是无法通过的。

复制代码如下:

my $full = "$head\_$tail";

🡪

Perl数组的创建读取以及赋值

@array\_2 = ("beijing", "tianjin", "chongqin"); # 数组的定义

@array\_2 = qw/Beijing Tianjin chongqin/; # 数组的定义方式2， 使用qw关键字

$array\_2[0] # 获取数组的值

push (array\_2, “wuhan”) # 向数组的结尾添加一个元素

pop(@array\_2) # 将数组最后一个元素弹出

unshift (@array\_2, “wuhan”) # 在数组的开始位置添加一个元素

shift(@array\_2) # 弹出数组的第一个元素

$testRef = [“Beijing”, “Tianjin”, “chongqin”]; # 使用 [] 时候代表返回数组的引用

${$testRef}[2] # 使用数组引用获取值，为chongqin

@{$testRef}[2] # 使用数组引用获取值，为chongqin

使用Perl构建数据结构

创建字典列表时候使用()

获取列表值使用[]

获取字典值使用{}

直接使用[]申明列表将返回引用

🡪

Perl字典的创建，读取，以及赋值。

my $obj = $json->decode($jsontext); # $obj为字典的引用

%test = ("1"=>"BBBBBB","3"=>"4"); #创建字典，方式1通过=>符号创建。注意为(), 不是{}

%test = ("1", "BBBBBB","3","4"); #创建字典，方式2通过列表创建。注意为(), 不是{}

$test {"5"} = "GAAAAAAAA"; #对字典赋值，非引用

$test2 = \%test; #获取字典引用

print $test {"1"}."\n"; # 对于非字典引用，读取字典值

print $test2-> {"5"}."\n"; # 对于字典引用，方式1，使用->读取字典引用的值

print $$test2{"5"}."\n”; # 对于字典引用，方式2，使用$$读取字典引用的值

print ${$test2}{"5"}."\n”; # 对于字典引用，方式3，使用$$读取字典引用的值

print %{$test2}.” \n” #从字典引用中，取到字典

🡪

将字典引用放入列表当中（放引用）

%testHash = ("1"=>"BBBBBB","3"=>"4"); # 定义一个字典

@testList = (1,2,3); # 创建列表

push (@testList, \%testHash); #将字典引用放入数组中

print Dumper(\@testList);

print $testList[3] -> {"1"}; # 通过字典引用获取字典对应值，为”BBBBBB”

🡪

将列表放入字典中（放引用）

%testHash = (); # 创建testHash字典

@testList = (1,2,3); 创建testList列表

$testHash{"MappingList"} = \@testList; # 将testList列表引用放入testHash字典中

print Dumper(\%testHash);

print $testHash{"MappingList"}; # 打印列表引用

print @{$testHash{"MappingList"}}[2]; # 使用列表反引用获取数字值，值为3

🡪

构建字典中含有列表，列表中再含有字典的数据结构

%testHash = (); # 构建testHash字典

@testList = (); # 构建testList列表

%test2Hash = (# 构建testHash2字典

"A" => "B",

"C" => "D"

);

$testHash{"MappingList"} = \@testList; # 将testList列表引用放入testHash字典中

push(@testList, \%test2Hash); # 将test2Hash字典引用放入testList列表中

print Dumper(\@testList);

print Dumper(\%testHash);

print @{$testHash{"MappingList"}}[0]->{"A"}; # 获取列表引用，使用反引用获取字典引用，再用字典反引用获取 “A” 所对应值” B”

🡪

Perl Net::Telnet 模块

模块中的print()和cmd()方法自己尾随\n符号。Put () 方法不会。

$t = new Net::Telnet (Timeout => 10,Prompt => '/bash\$ $/') 可以在构建telnet实例的时候告诉telnet实例，命令prompt的符号是什么。那么login()和cmd()将使用这个prompt作为login()或者cmd()结束的等待符。

在构建Telnet实例的时候有2种指定参数的方法：

Net::Telnet->new(Timeout => 20)

以及

Net::Telnet->new(-timeout => 20)

如果不指定参数，那么返回的实例将会以默认的参数构建。例如默认的timeout是10秒，如果不指定timeout参数，那么返回的实例中timeout便是10秒。

$telnetobj->cmd(…..) 将返回$command返回的字符以及matching prompt.

如果是在scalar环境下，那么将返回返回的字数个数。

如果是在list context环境下，那么将返回list,每一个元素是返回的一行。如果没有命令没有返回，那么将会返回一个空的list.

Cmd命令将默认添加output\_record\_separator 默认情况下是\n.

errmode参数。Errmode总共有4种模式：

--die, 默认的参数，当telnet遇到error的时候，将错误信息输出，然后程序中断。

--return, 遇到telnet错误的时候，将错误信息放置到telnet实例中，然后返回undefined value (scalar contenxt) 或者是空列表 (list context)

--coderef， 遇到telnet错误的时候， coderef所代表的方法将会调用，并且错误信息作为第一个参数。这个目的在于当遇到telnet错误的时候， 用户可以自行去做某些error handle的操作。

--arrayref, 指定一个数组，数组的第一个元素是coderef, 剩下的元素都是coderef的参数。当遇到telnet错误的时候，将调用coderef将参数传入。其实就是coderef的加强版本。

🡪

Perl 调用外部命令的三种方式

system("command");  
使用该命令将开启一个子进程执行引号中的命令，父进程将等待子进程结束，等待被调用的命令结束，并继续执行下面的代码。  
  
exec("command");  
效果同system命令类似，区别是不会开启子进程，而是取代父进程，因此执行完引号中的命令后进程即结束。一般和fork配合使用。  
  
`command`;  
使用反引号调用外部命令能够捕获其标准输出，并按行返回且每行结束处附带一个回车。反引号中的变量在编译时会被内插为其值。同样的，程序会等待被调用的命令结束，然后继续执行下面代码。

🡪

Telnet timeout的情况

情况1: 如果一个perl脚本去telnet另外一个perl脚本，将telnet是10秒。那么要求被call的脚本必须要在10秒内完成，即便被call的脚本每1秒都print出一些内容。这些print出的内容也不会让链接保持活性。

情况2: 如果一个perl脚本去telnet另外一个python脚本，该telnet是10秒。那么要求被call的脚本必须要在10秒内完成，即便被call的脚本每1秒都print出一些内容。这些print出的内容也不会让链接保持活性。

情况1与情况2的状态是， 当telnet去执行cmd命令的时候，它所等待的是timeout时间内收到command prompt的时间，所以期间所输出的一些内容，只要不满足command prompt的时候，都无效。而如果我们在被call的脚本中，硬性的去输出telnet所期待的command prompt那么telnet不会timeout,但是这样telnet会认为该命令已经结束了，直接变结束。从而实际上被call的命令并没有完全执行完，telnet便结束了。并且需要注意的是，telnet一结束，那么在运程服务器上被调用的脚本命令，也会被结束。而不是空挂着在远程服务器上执行完。

P.s 当telnet进程结束，远程服务器中被该telnet进程call的进程也会终止。

情况3： 如果一个perlA脚本去telnet执行脚本perlB，然后perlB脚本去执行perlC脚本。 如果perlC脚本需要10秒，perlB的timeout是20秒，perlA脚本是5秒。那么无法执行成功，因为5秒以后perlA会断，从而perlB也断，那么perlC也随着断。所以要顺利执行完，将perlA的脚本定位10秒timeout.

🡪

Perl 单行注释使用#， 多行注释使用=pod开头以及=cut结尾

🡪

Perl的Data::Dumper模块可以将哈希表， 列表， 甚至嵌套的哈希表中的内容打印出来，而不是默认的print 哈希表只打印出哈希表的引用地址。

🡪

Perl的字符串替代使用正则表达式的s///进行。

🡪

Perl判断条件时：

%test = ("B" =>"");

if ($test{"A"}) {# 进行直接判断是，如果键值没有被定义或者有键定义但是值为空字符串时，都为非。而使用defined进行判断时， 只要有定义键，那么就算键值为空也为正。

print "AAA";

}

else {

print "BBB";

}

🡪

Perl一些特殊变量

$\_ 循环时的每一个元素

$$ 运行当前Perl脚本程序的进程号

$1, $2… 正则表达式的捕获分组

🡪

<=>运算符用于返回两个数的比较结果。如果左边数小于右边的数返回-1，等于右边的数返回0，大于右边的数返回1.

🡪

Perl排序sort的三种形式：

sort list;

第一种形式默认返回按字母升序排序的列表

sort block list;

第二种形式默认返回按数值升序排序的列表。

在block中有特殊变量$a,以及$b.

sort subroutine\_name list

第三种形式，用于复杂排序。将自定义一个排序函数并且传入用于比较。

🡪