第十四章 Perl5的包和模块

by flamephoenix

一、require函数

用require函数可以把程序分割成多个文件并创建函数库。例如，在myfile.pl中有定义好的Perl函数，可用语句require ("myfile.pl"); 在程序中包含进来。当Perl解释器看到这一语句，就在内置数组变量@INC指定的目录中寻找文件myfile.pl。如果找到了，该文件中的语句就被执行，否则程序终止并输出错误信息：

Can't find myfile.pl in @INC

作为子程序调用参数，文件中最后一个表达式的值成为返回值，require函数查看其是否为零，若为零则终止。例如myfile.pl最后的语句是：

```
print ("hello, world!\n");
$var = 0;
```

因为最后的语句值为零，Perl解释器输出下列错误信息并推出：

myfile.pl did not reture true value

可以用简单变量或数组元素等向require传递参数，如：

```
@reqlist = ("file1.pl", "file2.pl", "file3.pl");
require ($reqlist[$0]);
require ($reqlist[$1]);
require ($reqlist[$2]);
```

还可以不指定文件名，即：

```
require;
```

这时，变量$_的值即作为文件名传递给require。

注：如果@INC中有多个目录中含有同一个文件，则只有第一个被包含。

1、require函数和子程序库

用require函数可以创建可用于所有Perl程序的子程序库，步骤如下：

a、确定存贮子程序库的目录
b、将子程序抽取放到单独的文件中，将文件放到子程序库目录
c、每个文件末尾加一句非零值的语句，最简单的办法是语句 1;
d、在主程序中用require包含一个或多个所需的文件。
e、运行主程序时，用 -I 选项指定子程序库目录，或者在调用require前将该目录添加到@INC数组中。

例如：假设目录/u/perldir中存有你的Perl子程序库，子程序mysub存贮在文件mysub.pl中。现在来包含上该文件：

```
unshift (@INC, "/u/perldir");
require ("mysub.pl");
```

对unshift的调用把目录/u/perldir添加到@INC数组，对require的调用将mysub.pl文件的内容包含进来作为程序的一部分。

注意：

1、应该使用unshift来向@INC中添加目录，而不是push。因为push增加到@INC的末尾，则该目录将被最后搜寻。

2、如果你的库文件名与/usr/local/lib/perl中的某文件同名，则不会被包含进来，因为require只包含同名文件中的第一个。

2、用require指定Perl版本

Perl 5中，可以用require语句来指定程序运行所需的Perl版本。当Perl解释器看到require后跟着数字时，则只有其版本高于或等于该数字时才运行该程序。例如，下面语句表明只有Perl解释器为5.001版或更高时才运行该程序：

```
require 5.001;
```

二、包

Perl程序把变量和子程序的名称存贮到符号表中，perl的符号表中名字的集合就称为包(package)。

1、包的定义

在一个程序中可以定义多个包，每个包有一个单独的符号表，定义语法为：

```
package mypack;
```

此语句定义一个名为mypack的包，从此以后定义的所有变量和子程序的名字都存贮在该包关联的符号表中，直到遇到另一个package语句为止。

每个符号表有其自己的一组变量、子程序名，各组名字是不相关的，因此可以在不同的包中使用相同的变量名，而代表的是不同的变量。如：

```
$var = 14;
package mypack;
$var = 6;
```

第一个语句创建变量$var并存贮在main符号表中，第三个语句创建另一个同名变量$var并存贮在mypack包的符号表中。

2、在包间切换

在程序里可以随时在包间来回切换，如：

```
1: #!/usr/local/bin/perl
2:
3: package pack1;
4: $var = 26;
5: package pack2;
6: $var = 34;
7: package pack1;
8: print ("$var\n");
```

运行结果如下：

```
$ program
26
$
```

第三行定义了包pack1，第四行创建变量$var，存贮在包pack1的符号表中，第五行定义新包pack2，第六行创建另一个变量$var，存贮在包pack2的符号表中。这样就有两个独立的$var，分别存贮在不同的包中。第七行又指定pack1为当前包，因为包pack1已经定义，这样，所有变量和子程序的定义和调用都为该包的符号表中存贮的名字。因此第八行对$var的调用为pack1包中的$var，其值为26。

3、main包

存贮变量和子程序的名字的缺省符号表是与名为main的包相关联的。如果在程序里定义了其它的包，当你想切换回去使用缺省的符号表，可以重新指定main包：

```
package main;
```

这样，接下来的程序就好象从没定义过包一样，变量和子程序的名字象通常那样存贮。

4、包的引用

在一个包中可以引用其它包中的变量或子程序，方法是在变量名前面加上包名和一个单引号，如：

```
                package mypack;
    $var = 26;
    package main;
    print ("$mypack'var\n");
```

这里，$mypack'var为mypack包中的变量$var。

注意：在Perl 5中，包名和变量名用双冒号隔开，即$mypack::var。单引号引用的方式仍然支持，但将来的版本中未必支持。

5、指定无当前包

在Perl 5中，可以用如下语句指定无当前包：

```
 package;
```

这时，所有的变量必须明确指出所属包名，否则就无效--错误。

```
 $mypack::var = 21; #ok
 $var = 21;  #error - no current package
```

这种情况直到用package语句指定当前包为止。

6、包和子程序

包的定义影响到程序中的所有语句，包括子程序，如：

```
                package mypack;
    subroutine mysub {
      local ($myvar);
      # stuff goes here
    }
```

这里，mysub和myvar都是包mypack的一部分。在包mypack外调用子程序mysub，则要指定包：$mypack'mysub。

可以在子程序中切换包：

```
                package pack1;
    subroutine mysub {
      $var1 = 1;
      package pack2;
      $var1 = 2;
    }
```

这段代码创建了两个变量$var1，一个在包pack1中，一个在包pack2中，包中的局域变量只能在其定义的子程序等语句块中使用，像普通的局域变量一样。

7、用包定义私有数据

包最通常的用途是用在含有子程序和子程序所使用的全局变量的文件中，为子程序定义这样的包，可以保证子程序使用的全局变量不可在其它地方使用，这样的数据即为私有数据。更进一步，可以保证包名不可在其它地方使用。私有数据例：

```
                1 : package privpack;
    2 : $valtoprint = 46;
    3 :
    4 : package main;
    5 : # This function is the link to the outside world.
    6 : sub printval {
```

```
7 :   &privpack'printval();
8 : }
9 :
10: package privpack;
11: sub printval {
12:   print ("$valtoprint\n");
13: }
14:
15: package main;
16: 1; # return value for require
```

此子程序只有在调用printval后才能产生输出。

该文件分为两个部分：与外界联系的部分和私有部分。前者为缺省的main包，后者为包privpack。第6~8行定义的子程序printval可被其它程序或子程序调用。printval输出变量$valtoprint的值，此变量仅在包privpack中定义和使用。第15、16行确保其被其它程序用require语句包含后工作正常，15行将当前包设置回缺省包main，16行返回非零值使require不报错。

8、包和系统变量

下列变量即使从其它包中调用，也在main包中起作用：

- 文件变量STDIN, STDOUT, STDERR 和 ARGV
- 变量%ENV, %INC, @INC, $ARGV 和 @ARGV
- 其它含有特殊字符的系统变量

9、访问符号表

在程序中查找符号表可用数组%_package，此处package为想访问的符号表所属的包名。例如%_main含有缺省的符号表。

通常不需要亲自查找符号表。

三、模块

多数大型程序都分割成多个部件，每一部件通常含有一个或多个子程序及相关的变量，执行特定的一个或多个任务。集合了变量和子程序的部件称为程序模块。

1、创建模块

Perl 5中用包来创建模块，方法是创建包并将之存在同名的文件中。例如，名为Mymodult的包存贮在文件Mymodult.pm中（扩展名.pm表示Perl Module）。下例的模块Mymodult含有子程序myfunc1和myfunc2及变量$myvar1和$myvar2。

```
1 : #!/usr/local/bin/perl
2 :
3 : package Mymodule;
4 : require Exporter;
5 : @ISA = qw(Exporter);
6 : @EXPORT = qw(myfunc1 myfunc2);
7 : @EXPORT_OK = qw($myvar1 $myvar2);
8 :
9 : sub myfunc1 {
10:   $myvar1 += 1;
11: }
12:
13: sub myfunc2 {
14:   $myvar2 += 2;
15: }
```

第3~7行是标准的Perl模块定义方式。第3行定义包，第4行包含内置Perl模块Exporter，6、7行进行子程序和变量的输出以与外界联系。第6行创建名为@EXPORT的特殊数组，该数组中的子程序可以被其它程序调用，这里，myfunc1和myfunc2可以被访问。其它任何在模块中定义但没有赋给数组

@EXPORT的子程序都是私有的，只能在模块内部调用。第7行创建另一个名为@EXPORT_OK的特殊数组，其中含有可被外部程序访问的变量，这里含有$myvar1和$myvar2。

2、导入模块

将模块导入你的Perl程序中使用use语句，如下句导入了Mymodule模块：

```
use Mymodule;
```

这样，模块Mymodule中的子程序和变量就可以使用了。

取消导入模块使用no语句，如下句取消了Mymodule模块的导入：

```
no Mymodule;
```

下面看一个导入模块和取消导入的例子，使用integer模块要求所有数字运算基于整数，浮点数在运算前均被转化为整数。

```
1: #!/usr/local/bin/perl
2:
3: use integer;
4: $result = 2.4 + 2.4;
5: print ("$result\n");
6:
7: no integer;
8: $result = 2.4 + 2.4;
9: print ("$result\n");
```

程序输出如下：

```
$ program
4
4.8
$
```

如果use或no语句出现在语句块中，则只在该块的有效范围内起作用，如：

```
use integer;
$result1 = 2.4 + 2.4;
if ($result1 == 4) {
no integer;
$result2 = 3.4 + 3.4;
}
$result3 = 4.4 + 4.4;
```

结果输出如下：

```
4
6.8
8
```

这里，no语句只在if语句中有效，出了if语句仍使用integer模块，因此4.4在做加法前被转化成了4。

3、预定义模块

Perl 5提供了许多有用的预定义模块，可以用use导入和no语句取消。下面是库中最有用的一些模块：

integer	使用整数运算
Diagnostics	输出较多的诊断信息（警告）
English	允许英文名用作系统变量的别名
Env	导入环境变量的Perl模块
POSIX	POSIX标准（IEEE 1003.1）的Perl接口
Socket	装载C语言的套接字处理机制

Perl文档中有完整的预定义模块列表。

注：世界各地的Perl 5用户写了许多有用的模块，CPAN(Comprehensive Perl Archive Network)的Perl文档有其完整的列表。关于CPAN的更多信息见其网址：http://www.perl.com/perl/CPAN/README.html。

上一章 下一章 目录