第十三章 Perl的面向对象编程

by flamephoenix

本章介绍如何使用Perl的面向对象编程(OOP)特性及如何构建对象，还包括继承、方法重载和数据封装等内容。

一、模块简介

模块(module)就是Perl包(pachage)。Perl中的对象基于对包中数据项的引用。（引用见第x章引用）。详见http://www.metronet.com的perlmod和perlobj。

在用其它语言进行面向对象编程时，先声明一个类然后创建该类的对象（实例），特定类所有对象的行为方式是相同的，由类方法确定，可以通过定义新类或从现存类继承来创建类。已熟悉面向对象编程的人可以在此遇到许多熟悉的术语。Perl一直是一个面向对象的语言，在Perl5中，语法略有变动，更规范化了对象的使用。

下面三个定义对理解对象、类和方法在Perl中如何工作至关重要。

.类是一个Perl包，其中含提供对象方法的类。

.方法是一个Perl子程序，类名是其第一个参数。

.对象是对类中数据项的引用。

二、Perl中的类

再强调一下，一个Perl类是仅是一个包而已。当你看到Perl文档中提到“类”时，把它看作“包”就行了。Perl5的语法可以创建类，如果你已熟悉C++，那么大部分语法你已经掌握了。与Perl4不同的概念是用双冒号(::)来标识基本类和继承类（子类）。

面向对象的一个重要特性是继承。Perl中的继承特性与其它面向对象语言不完全一样，它只继承方法，你必须用自己的机制来实现数据的继承。

因为每个类是一个包，所以它有自己的名字空间及自己的符号名关联数组（详见第x章关联数组），每个类因而可以使用自己的独立符号名集。与包的引用结合，可以用单引号(')操作符来定位类中的变量，类中成员的定位形式如：$class'$member。在Perl5中，可用双冒号替代单引号来获得引用，如：$class'$member与$class::$member相同。

三、创建类。

本节介绍创建一个新类的必要步骤。下面使用的例子是创建一个称为Cocoa的简单的类，其功能是输出一个简单的Java应用的源码的必要部分。放心，这个例子不需要你有Java的知识，但也不会使你成为Java专家，其目的是讲述创建类的概念。

首先，创建一个名为Cocoa.pm的包文件(扩展名pm是包的缺省扩展名，意为Perl Module)。一个模块就是一个包，一个包就是一个类。在做其它事之前，先加入“1;”这样一行，当你增加其它行时，记住保留“1;”为最后一行。这是Perl包的必需条件，否则该包就不会被Perl处理。下面是该文件的基本结构。

```
package Cocoa;

#
```

```
# Put "require" statements in for all required,imported packages
#

#
# Just add code here
#

1; # terminate the package with the required 1;
```

接下来，我们往包里添加方法使之成为一个类。第一个需添加的方法是new()，它是创建对象时必须被调用的，new()方法是对象的构造函数。

四、构造函数

构造函数是类的子程序，它返回与类名相关的一个引用。将类名与引用相结合称为“祝福”一个对象，因为建立该结合的函数名为bless()，其语法为：

bless YeReference [,classname]

YeReference是对被“祝福”的对象的引用，classname是可选项，指定对象获取方法的包名，其缺省值为当前包名。

创建一个构建函数的方法为返回已与该类结合的内部结构的引用，如：

```
sub new {
  my $this = {}; # Create an anonymous hash, and #self points to it.
  bless $this; # Connect the hash to the package Cocoa.
  return $this; # Return the reference to the hash.
}

1;
```

{}创建一个对不含键/值对的哈希表（即关联数组）的引用，返回值被赋给局域变量$this。函数bless()取出该引用，告诉对象它引用的是Cocoa，最后返回该引用。函数的返回值现在指向这个匿名哈希表。

从new()函数返回后，$this引用被销毁，但调用函数保存了对该哈希表的引用，因此该哈希表的引用数不会为零，从而使Perl在内存中保存该哈希表。创建对象可如下调用：

$cup = new Cocoa;

下面语句为使用该包创建对象的例子：

```
1 #!/usr/bin/perl
2 push (@INC,'pwd');
3 use Cocoa;
4 $cup = new Cocoa;
```

第一行指出Perl解释器的位置，第二行中，将当前目录加到路径寻找列表@INC中供寻找包时使用。你也可以在不同的目录中创建你的模块并指出该绝对路径。例如，如果在/home/test/scripts/创建包，第二行就应该如下：

push (@INC , "/home/test/scripts");

在第三行中，包含上包Cocoa.pm以获取脚本中所需功能。use语句告诉Perl在@INC路径寻找文件Cocoa.pm并包含到解析的源文件拷贝中。use语句是使用类必须的。第四行调用new函数创建对象，这是Perl的妙处，也是其易混淆之处，也是其强大之处。创建对象的方法有多种，可以这样写：

$cup = cocoa->new();

如果你是C程序员，可以用双冒号强制使用Cocoa包中的new()函数，如：

$cup = Cocoa::new();

可以在构造函数中加入更多的代码，如在Cocoa.pm中，可以在每个对象创建时输出一个简单声明，还可以用构造函数初始化变量或设置数组或指针。

注意：

1、一定要在构造函数中初始化变量；

2、一定要用my函数在方法中创建变量；

3、一定不要在方法中使用local，除非真的想把变量传递给其它子程序；
4、一定不要在类模块中使用全局变量。

加上声明的Cocoa构造函数如下：

```
sub new {
  my $this = {};
  print "\n /* \n ** Created by Cocoa.pm \n ** Use at own risk";
  print "\n ** Did this code even get pass the javac compiler? ";
  print "\n **/ \n";
  bless $this;
  return $this;
}
```

也可以简单地调用包内或包外的其它函数来做更多的初始化工作，如：

```
sub new {
  my $this = {}
  bless $this;
  $this->doInitialization();
  return $this;
}
```

创建类时，应该允许它可被继承，应该可以把类名作为第一个参数来调用new函数，那么new函数就象下面的语句：

```
sub new {
  my $class = shift; # Get the request class name
  my $this = {};
  bless $this, $class # Use class name to bless() reference
  $this->doInitialization(); return $this;
}
```

此方法使用户可以下列三种方式之一来进行调用：

- Cocoa::new()
- Cocoa->new()
- new Cocoa

可以多次bless一个引用对象，然而，新的将被bless的类必然把对象已被bless的引用去掉，对C和Pascal程序员来说，这就象把一个指针赋给分配的一块内存，再把同一指针赋给另一块内存而不释放掉前一块内存。总之，一个Perl对象每一时刻只能属于一个类。
对象和引用的真正区别是什么呢？Perl对象被bless以属于某类，引用则不然，如果引用被bless，它将属于一个类，也便成了对象。对象知道自己属于哪个类，引用则不属于任何类。

- 实例变量

作为构造函数的new()函数的参数叫做实例变量。实例变量在创建对象的每个实例时用于初始化，例如可以用new()函数为对象的每个实例起个名字。
可以用匿名哈希表或匿名数组来保存实例变量。
用哈希表的代码如下：

```
sub new {
        my $type = shift;
    my %parm = @_;
```

```
        my $this = {};
        $this->{'Name'} = $parm{'Name'};
        $this->{'x'} = $parm{'x'};
        $this->{'y'} = $parm{'y'};
        bless $this, $type;

    }
```

用数组保存的代码如下：

```
                sub new {

                    my $type = shift;
        my %parm = @_;
        my $this = [];
        $this->[0] = $parm{'Name'};
        $this->[1] = $parm{'x'};
        $this->[2] = $parm{'y'};
        bless $this, $type;

    }
```

构造对象时，可以如下传递参数：

```
$mug = Cocoa::new( 'Name' => 'top','x' => 10,'y' => 20 );
```

操作符=>与逗号操作服功能相同，但=>可读性好。访问方法如下：

```
print "Name=$mug->{'Name'}\n";
print "x=$mug->{'x'}\n";
print "y=$mug->{'y'}\n";
```

五、方法

Perl类的方法只不过是一个Perl子程序而已，也即通常所说的成员函数。Perl的方法定义不提供任何特殊语法，但规定方法的第一个参数为对象或其被引用的包。Perl有两种方法：静态方法和虚方法。

静态方法第一个参数为类名，虚方法第一个参数为对象的引用。方法处理第一个参数的方式决定了它是静态的还是虚的。静态方法一般忽略掉第一个参数，因为它们已经知道自己在哪个类了，构造函数即静态方法。虚方法通常首先把第一个参数shift到变量self或this中，然后将该值作普通的引用使用。如：

```
                1. sub nameLister {
    2.     my $this = shift;
    3.     my ($keys ,$value );
    4.     while (($key, $value) = each (%$this)) {
    5.         print "\t$key is $value.\n";
    6.     }
    7. }
```

六、方法的输出

如果你现在想引用Cocoa.pm包，将会得到编译错误说未找到方法，这是因为Cocoa.pm的方法还没有输出。输出方法需要Exporter模块，在包的开始部分加上下列两行：

```
require Exporter;
@ISA = qw (Exporter);
```

这两行包含上Exporter.pm模块，并把Exporter类名加入@ISA数组以供查找。接下来把你自己的类方法列在@EXPORT数组中就可以了。例如想输出方法closeMain和declareMain，语句如下：

```
@EXPORT = qw (declareMain , closeMain);
```

Perl类的继承是通过@ISA数组实现的。@ISA数组不需要在任何包中定义，然而，一旦它被定义，Perl就把它看作目录名的特殊数组。它与@INC数组类似，@INC是包含文件的寻找路径。@ISA数组含有类(包)名，当一个方法在当前包中未找到时就到@ISA中的包去寻找。@ISA中还含有当前类继承的基类名。

类中调用的所有方法必须属于同一个类或@ISA数组定义的基类。如果一个方法在@ISA数组中未找到，Perl就到AUTOLOAD()子程序中寻找，这个可选的子程序在当前包中用sub定义。若使用AUTOLOAD子程

序，必须用use Autoload;语句调用autoload.pm包。AUTOLOAD子程序尝试从已安装的Perl库中装载调用的方法。如果AUTOLOAD也失败了，Perl再到UNIVERSAL类做最后一次尝试，如果仍失败，Perl就生成关于该无法解析函数的错误。

七、方法的调用

调用一个对象的方法有两种方法，一是通过该对象的引用（虚方法），一是直接使用类名（静态方法）。当然该方法必须已被输出。现在给Cocoa类增加一些方法，代码如下：

```
                package Cocoa;
require Exporter;
@ISA = qw(Exporter);
@EXPORT = qw(setImports, declareMain, closeMain);
#
# This routine creates the references for imports in Java functions
#
sub setImports{
  my $class = shift @_;
  my @names = @_;
  foreach (@names) {
    print "import " . $_ . ";\n";
  }
}
#
# This routine declares the main function in a Java script
#
sub declareMain{
  my $class = shift @_;
  my ( $name, $extends, $implements) = @_;
  print "\n public class $name";
  if ($extends) {
    print " extends " . $extends;
  }
  if ($implements) {
    print " implements " . $implements;
  }
  print " { \n";
}
#
# This routine declares the main function in a Java script
#
sub closeMain{
  print "} \n";
}
#
# This subroutine creates the header for the file.
#
sub new {
  my $this = {};
  print "\n /* \n ** Created by Cocoa.pm \n ** Use at own risk \n */ \n";
  bless $this;
  return $this;
}

1;
```

现在，我们写一个简单的Perl脚本来使用该类的方法，下面是创建一个Java applet源代码骨架的脚本代码：

```
#!/usr/bin/perl
use Cocoa;
$cup = new Cocoa;
$cup->setImports( 'java.io.InputStream', 'java.net.*');
$cup->declareMain( "Msg" , "java.applet.Applet", "Runnable");
$cup->closeMain();
```

这段脚本创建了一个叫做Msg的Java applet，它扩展(extend)了java.applet.Applet小应用程序并使之可运行(runnable)，其中最后三行也可以写成如下：

```
Cocoa::setImports($cup, 'java.io.InputStream', 'java.net.*');
Cocoa::declareMain($cup, "Msg" , "java.applet.Applet", "Runnable");
Cocoa::closeMain($cup);
```

其运行结果如下：

```
/*
** Created by Cocoa.pm
** Use at own risk
*/
import java.io.InputStream;
import java.net.*;

public class Msg extends java.applet.Applet implements Runnable {
}
```

注意：如果用->操作符调用方法（也叫间接调用），参数必须用括号括起来，如：$cup->setImports('java.io.InputStream', 'java.net.*');而双冒号调用如：Cocoa::setImports($cup, 'java.io.InputStream', 'java.net.*');也可去掉括号写成：Cocoa::setImports $cup, 'java.io.InputStream', 'java.net.*' ;

八、重载

有时需要指定使用哪个类的方法，如两个不同的类有同名方法的时候。假设类Espresso和Qava都定义了方法grind，可以用::操作符指定使用Qava的方法：

```
$mess = Qava::grind("whole","lotta","bags");
Qava::grind($mess, "whole","lotta","bags");
```

可以根据程序的运行情况来选择使用哪个类的方法，这可以通过使用符号引用去调用来实现：

```
$method = $local ? "Qava::" : "Espresso::";
$cup->{$method}grind(@args);
```

九、析构函数

Perl跟踪对象的链接数目，当某对象的最后一个应用释放到内存池时，该对象就自动销毁。对象的析构发生在代码停止后，脚本将要结束时。对于全局变量而言，析构发生在最后一行代码运行之后。

如果你想在对象被释放之前获取控制权，可以定义DESTROY()方法。DESTROY()在对象将释放前被调用，使你可以做一些清理工作。DESTROY()函数不自动调用其它DESTROY()函数，Perl不做内置的析构工作。如果构造函数从基类多次bless，DESTROY()可能需要调用其它类的DESTROY()函数。当一个对象被释放时，其内含的所有对象引用自动释放、销毁。

一般来说，不需要定义DESTROY()函数，如果需要，其形式如下：

```
sub DESTROY {
#
# Add code here.
#
}
```

因为多种目的，Perl使用了简单的、基于引用的垃圾回收系统。任何对象的引用数目必须大于零，否则该对象的内存就被释放。当程序退出时，Perl的一个彻底的查找并销毁函数进行垃圾回收，进程中的一切被简单地删除。在UNIX类的系统中，这像是多余的，但在内嵌式系统或多线程环境中这确实很必要。

十、继承

类方法通过@ISA数组继承，变量的继承必须明确设定。下例创建两个类Bean.pm和Coffee.pm，其中Coffee.pm继承Bean.pm的一些功能。此例演示如何从基类（或称超类）继承实例变量，其方法为调用基类的构造函数并把自己的实例变量加到新对象中。

Bean.pm代码如下：

```
package Bean;
require Exporter;
@ISA = qw(Exporter);
@EXPORT = qw(setBeanType);

sub new {
  my $type = shift;
  my $this = {};
  $this->{'Bean'} = 'Colombian';
  bless $this, $type;
  return $this;
}

#
# This subroutine sets the class name
sub setBeanType{
  my ($class, $name) = @_;
  $class->{'Bean'} = $name;
  print "Set bean to $name \n";
}
1;
```

此类中，用$this变量设置一个匿名哈希表，将'Bean'类型设为'Colombian'。方法setBeanType()用于改变'Bean'类型，它使用$class引用获得对对象哈希表的访问。

Coffee.pm代码如下：

```
1  #
2  # The Coffee.pm file to illustrate inheritance.
3  #
4  package Coffee;
5  require Exporter;
6  require Bean;
7  @ISA = qw(Exporter, Bean);
8  @EXPORT = qw(setImports, declareMain, closeMain);
9  #
10 # set item
11 #
12 sub setCoffeeType{
13   my ($class,$name) = @_;
14   $class->{'Coffee'} = $name;
15   print "Set coffee type to $name \n";
16   }
17 #
18 # constructor
19 #
20 sub new {
```

```
21  my $type = shift;
22  my $this = Bean->new(); ##### <- LOOK HERE!!! ####
23  $this->{'Coffee'} = 'Instant'; # unless told otherwise
24  bless $this, $type;
25  return $this;
26  }
27 1;
```

第6行的require Bean;语句包含了Bean.pm文件和所有相关函数，方法setCoffeeType()用于设置局域变量$class->{'Coffee'}的值。在构造函数new()中，$this指向Bean.pm返回的匿名哈希表的指针，而不是在本地创建一个，下面两个语句分别为创建不同的哈希表从而与Bean.pm构造函数创建的哈希表无关的情况和继承的情况：

```
my $this = {}; #非继承
my $this = $theSuperClass->new(); #继承
```

下面代码演示如何调用继承的方法：

```
1  #!/usr/bin/perl
2  push (@INC,'pwd');
3  use Coffee;
4  $cup = new Coffee;
5  print "\n ------------------- Initial values ------------ \n";
6  print "Coffee: $cup->{'Coffee'} \n";
7  print "Bean: $cup->{'Bean'} \n";
8  print "\n ------------------- Change Bean Type ---------- \n";
9  $cup->setBeanType('Mixed');
10 print "Bean Type is now $cup->{'Bean'} \n";
11 print "\n ----------------- Change Coffee Type ---------- \n";
12 $cup->setCoffeeType('Instant');
13 print "Type of coffee: $cup->{'Coffee'} \n";
```

该代码的结果输出如下：

```
------------------- Initial values ------------
Coffee: Instant
Bean: Colombian
------------------- Change Bean Type ----------
Set bean to Mixed
Bean Type is now Mixed
----------------- Change Coffee Type ----------
Set coffee type to Instant
Type of coffee: Instant
```

上述代码中，先输出对象创建时哈希表中索引为'Bean'和'Coffee'的值，然后调用各成员函数改变值后再输出。

方法可以有多个参数，现在向Coffee.pm模块增加函数makeCup()，代码如下：

```
sub makeCup {
  my ($class, $cream, $sugar, $dope) = @_;
  print "\n================================== \n";
  print "Making a cup \n";
  print "Add cream \n" if ($cream);
  print "Add $sugar sugar cubes\n" if ($sugar);
  print "Making some really addictive coffee ;-) \n" if ($dope);
  print "================================== \n";
 }
```

此函数可有三个参数，不同数目、值的参数产生不同的结果，例如：

```
#!/usr/bin/perl
push (@INC,'pwd');
use Coffee;
$cup = new Coffee;
#
# With no parameters
#
print "\n Calling with no parameters: \n";
$cup->makeCup;
#
# With one parameter
#
print "\n Calling with one parameter: \n";
$cup->makeCup('1');
#
# With two parameters
#
print "\n Calling with two parameters: \n";
$cup->makeCup(1,'2');
#
# With all three parameters
#
print "\n Calling with three parameters: \n";
$cup->makeCup('1',3,'1');
```

其结果输出如下：

```
Calling with no parameters:
==================================
Making a cup
==================================
Calling with one parameter:
==================================
Making a cup
Add cream
==================================
Calling with two parameters:
==================================
Making a cup
Add cream
Add 2 sugar cubes
==================================
Calling with three parameters:
==================================
Making a cup
Add cream
Add 3 sugar cubes
Making some really addictive coffee ;-)
==================================
```

在此例中，函数makeCup()的参数既可为字符串也可为整数，处理结果相同，你也可以把这两种类型的数据处理区分开。在对参数的处理中，可以设置缺省的值，也可以根据实际输入参数值的个数给予不同处理。

十一、子类方法的重载

继承的好处在于可以获得基类输出的方法的功能，而有时需要对基类的方法重载以获得更具体或不同的功能。下面在Bean.pm类中加入方法printType()，代码如下：

```
sub printType {
  my $class = shift @_;
  print "The type of Bean is $class->{'Bean'} \n";
}
```

然后更新其@EXPORT数组来输出：

```
@EXPORT = qw ( setBeanType , printType );
```

现在来调用函数printType()，有三种调用方法：

```
$cup->Coffee::printType();
$cup->printType();
$cup->Bean::printType();
```

输出分别如下：

```
The type of Bean is Mixed
The type of Bean is Mixed
The type of Bean is Mixed
```

为什么都一样呢？因为在子类中没有定义函数printType()，所以实际均调用了基类中的方法。如果想使子类有其自己的printType()函数，必须在Coffee.pm类中加以定义：

```
#
# This routine prints the type of $class->{'Coffee'}
#
sub printType {
  my $class = shift @_;
  print "The type of Coffee is $class->{'Coffee'} \n";
}
```

然后更新其@EXPORT数组：

```
@EXPORT = qw(setImports, declareMain, closeMain, printType);
```

现在输出结果变成了：

```
The type of Coffee is Instant
The type of Coffee is Instant
The type of Bean is Mixed
```

现在只有当给定了Bean::时才调用基类的方法，否则直接调用子类的方法。

那么如果不知道基类名该如何调用基类方法呢？方法是使用伪类保留字SUPER::。在类方法内使用语法如：$this->SUPER::function(...argument list...);，它将从@ISA列表中寻找。刚才的语句用SUPER::替换Bean::可以写为$cup->SUPER::printType();，其结果输出相同，为：

```
The type of Bean is Mixed
```

十二、Perl类和对象的一些注释

OOP的最大好处就是代码重用。OOP用数据封装来隐藏一些复杂的代码，Perl的包和模块通过my函数提供数据封装功能，但是Perl并不保证子类一定不会直接访问基类的变量，这确实减少了数据封装的好处，虽然这种动作是可以做到的，但却是个很坏的编程风格。

注意：

1、一定要通过方法来访问类变量。

2、一定不要从模块外部直接访问类变量。

当编写包时，应该保证方法所需的条件已具备或通过参数传递给它。在包内部，应保证对全局变量的访问只用通过方法传递的引用来访问。对于方法要使用的静态或全局数据，应该在基类中用local()来定义，子类通过调用基类来获取。有时，子类可能需要改变这种数据，这时，基类可能就不知道怎样去寻找新的数据，因此，这时最好定义对该数据的引用，子类和基类都通过引用来改变该数据。

最后，你将看到如下方式来使用对象和类：

```
use coffee::Bean;
```

这句语句的含义是“在@INC数组所有目录的Coffee子目录来寻找Bean.pm”。如果把Bean.pm移到./Coffee目录，上面的例子将用这一use语句来工作。这样的好处是有条理地组织类的代码。再如，下面的语句：

```
use Another::Sub::Menu;
```

意味着如下子目录树：

```
./Another/Sub/Menu.pm
```

上一章 下一章 目录