Perl Socket 编程 Perl 数据库连接

Perl 面向对象

Perl 中有两种不同地面向对象编程的实现：

- 一是基于匿名哈希表的方式，每个对象实例的实质就是一个指向匿名哈希表的引用。在这个匿名哈希表中，存储来所有的实例属性。
- 二是基于数组的方式，在定义一个类的时候，我们将为每一个实例属性创建一个数组，而每一个对象实例的实质就是一个指向这些数组中某一行索引的引用。在这些数组中，存储着所有的实例属性。

面向对象基础概念

面向对象有很多基础概念，这里我们接收三个：对象、类和方法。

- **对象**：对象是对类中数据项的引用。.
- **类**：类是个Perl包，其中含提供对象方法的类。
- **方法**：方法是个Perl子程序，类名是其第一个参数。

Perl 提供了 bless() 函数，bless 是用来构造对象的， 通过 bless 把一个引用和这个类名相关联，返回这个引用就构造出一个对象。

类的定义

一个类只是一个简单的包。

可以把一个包当作一个类用，并且把包里的函数当作类的方法来用。

Perl 的包提供了独立的命名空间，所以不同包的方法与变量名不会冲突。

Perl 类的文件后缀为 .pm。

接下来我们创建一个 Person 类：

```
package Person;
```

类的代码范围到脚本文件的最后一行，或者到下一个 package 关键字前。

创建和使用对象

创建一个类的实例 (对象) 我们需要定义一个构造函数，大多数程序使用类名作为构造函数，Perl 中可以使用任何名字。

你可以使用多种 Perl 的变量作为 Perl 的对象。大多数情况下我们会使用引用数组或哈希。

接下来我们为 Person 类创建一个构造函数，使用了 Perl 的哈希引用。

在创建对象时，你需要提供一个构造函数，它是一个子程序，返回对象的引用。

实例如下：

实例

```
package Person;
sub new
{
my $class = shift;
my $self = {
_firstName => shift,
_lastName => shift,
_ssn => shift,
};
# 输出用户信息
print "名字: $self->{_firstName}\n";
print "姓氏: $self->{_lastName}\n";
print "编号: $self->{_ssn}\n";
bless $self, $class;
return $self;
}
```

接下来我们创建一个对象：

```
$object = new Person( "小明", "王", 23234345);
```

定义方法

Perl类的方法只但是是个Perl子程序而已，也即通常所说的成员函数。

Perl面向对象中Perl的方法定义不提供任何特别语法，但规定方法的第一个参数为对象或其被引用的包。

Perl 没有提供私有变量，但我们可以通过辅助的方式来管理对象数据。

接下来我们定义一个获取名字的方法：

```
sub getFirstName {
    return $self->{_firstName};
}
```

同样也可以这么写：

```
sub setFirstName {
    my ( $self, $firstName ) = @_;
    $self->{_firstName} = $firstName if defined($firstName);
    return $self->{_firstName};
}
```

接下来我们修改 Person.pm 文件的代码，如下所示：

实例

```
#!/usr/bin/perl
package Person;
sub new
{
my $class = shift;
my $self = {
_firstName => shift,
_lastName => shift,
_ssn => shift,
};
# 输出用户信息
print "名字: $self->{_firstName}\n";
print "姓氏: $self->{_lastName}\n";
print "编号: $self->{_ssn}\n";
bless $self, $class;
return $self;
}
sub setFirstName {
my ( $self, $firstName ) = @_;
$self->{_firstName} = $firstName if defined($firstName);
return $self->{_firstName};
}
sub getFirstName {
my( $self ) = @_;
return $self->{_firstName};
}
1;
```

employee.pl 脚本代码如下：

实例

```
#!/usr/bin/perl
use Person;
$object = new Person( "小明", "王", 23234345);
# 获取姓名
$firstName = $object->getFirstName();
print "设置前姓名为 : $firstName\n";
# 使用辅助函数设置姓名
$object->setFirstName( "小强" );
# 通过辅助函数获取姓名
$firstName = $object->getFirstName();
print "设置后姓名为 : $firstName\n";
```

执行以上程序后，输出结果为：

```
$ perl employee.pl
名字: 小明
姓氏: 王
编号: 23234345
设置前姓名为 : 小明
设置后姓名为 : 小强
```

继承

Perl 里 类方法通过@ISA数组继承，这个数组里面包含其他包（类）的名字，变量的继承必须明确设定。

多继承就是这个@ISA数组包含多个类（包）名字。

通过@ISA只能继承方法，不能继承数据。

接下来我们创建一个 Employee 类继承 Person 类。

Employee.pm 文件代码如下所示：

实例

```
#!/usr/bin/perl
package Employee;
use Person;
use strict;
our @ISA = qw(Person); # 从 Person 继承
```

现在 Employee 类包含了 Person 类的所有方法和属性，我们在 main.pl 文件中输入以下代码，并执行:

实例

```
#!/usr/bin/perl
use Employee;
$object = new Employee( "小明", "王", 23234345);
# 获取姓名
$firstName = $object->getFirstName();
print "设置前姓名为 : $firstName\n";
# 使用辅助函数设置姓名
$object->setFirstName( "小强" );
# 通过辅助函数获取姓名
$firstName = $object->getFirstName();
print "设置后姓名为 : $firstName\n";
```

执行以上程序后，输出结果为：

```
$ perl main.pl
名字: 小明
姓氏: 王
编号: 23234345
设置前姓名为 : 小明
设置后姓名为 : 小强
```

方法重写

上面实例中，Employee 类继承了 Person 类，但如果 Person 类的方法无法满足需求，就需要对其方法进行重写。

接下来我们在 Employee 类中添加一些新方法，并重写了 Person 类的方法：

实例

```
#!/usr/bin/perl
package Employee;
```

```
use Person;
use strict;
our @ISA = qw(Person); # 从 Person 继承
# 重写构造函数
sub new {
my ($class) = @_;
# 调用父类的构造函数
my $self = $class->SUPER::new( $_[1], $_[2], $_[3] );
# 添加更多属性
$self->{_id} = undef;
$self->{_title} = undef;
bless $self, $class;
return $self;
}
# 重写方法
sub getFirstName {
my( $self ) = @_;
# 这是子类函数
print "这是子类函数\n";
return $self->{_firstName};
}
# 添加方法
sub setLastName{
my ( $self, $lastName ) = @_;
$self->{_lastName} = $lastName if defined($lastName);
return $self->{_lastName};
}
sub getLastName {
my( $self ) = @_;
return $self->{_lastName};
}
1;
```

我们在 main.pl 文件中输入以下代码，并执行:

实例

```
#!/usr/bin/perl
use Employee;
$object = new Employee( "小明", "王", 23234345);
# 获取姓名，使用修改后的构造函数
$firstName = $object->getFirstName();
print "设置前姓名为 : $firstName\n";
# 使用辅助函数设置姓名
$object->setFirstName( "小强" );
# 通过辅助函数获取姓名
$firstName = $object->getFirstName();
print "设置后姓名为 : $firstName\n";
```

执行以上程序后，输出结果为：

```
$ perl main.pl
名字: 小明
```

```
姓氏：王
编号：23234345
这是子类函数
设置前姓名为 ： 小明
这是子类函数
设置后姓名为 ： 小强
```

默认载入

如果在当前类、当前类所有的基类、还有 UNIVERSAL 类中都找不到请求的方法， 这时会再次查找名为 AUTOLOAD() 的一个方法。如果找到了 AUTOLOAD，那么就会 调用，同时设定全局变量 $AUTOLOAD 的值为缺失的方法的全限定名称。

如果还不行，那么 Perl 就宣告失败并出错。

如果你不想继承基类的 AUTOLOAD，很简单，只需要一句：

```
sub AUTOLOAD;
```

析构函数及垃圾回收

当对象的最后一个引用释放时，对象会自动析构。

如果你想在析构的时候做些什么，那么你可以在类中定义一个名为"DESTROY"的方法。它将在适合的时机自动调用，并且按照你的意思执行额外的清理动作。

```
package MyClass;
...
sub DESTROY
{
    print "MyClass::DESTROY called\n";
}
```

Perl 会把对象的引用作为 唯一的参数传递给 DESTROY。注意这个引用是只读的，也就是说你不能通过访问$_[0] 来修改它。（译者注：参见 perlsub）但是对象自身（比如 "${$_[0]" 或者 "@{$_[0]}" 还有 "%{$_[0]}" 等等）还是可写的。

如果你在析构器返回之前重新 bless 了对象引用，那么 Perl 会在析构器返回之后接着调用你重新 bless 的那个对象的 DESTROY 方法。这可以让你有机会调用基类或者你指定的其它类的析构器。需要说明的是，DESTROY 也可以手工调用，但是通常没有必要这么做。

在当前对象释放后，包含在当前对象中的其它对象会自动释放。

Perl 面向对象实例

我们可以通过以下实例来进一步理解Perl面向对象的应用：

实例

```
#!/usr/bin/perl
# 下面是简单的类实现
```

```
package MyClass;
sub new
{
print "MyClass::new called\n";
my $type = shift; # 包名
my $self = {}; # 引用空哈希
return bless $self, $type;
}
sub DESTROY
{
print "MyClass::DESTROY called\n";
}
sub MyMethod
{
print "MyClass::MyMethod called!\n";
}
# 继承实现
package MySubClass;
@ISA = qw( MyClass );
sub new
{
print "MySubClass::new called\n";
my $type = shift; # 包名
my $self = MyClass->new; # 引用空哈希
return bless $self, $type;
}
sub DESTROY
{
print "MySubClass::DESTROY called\n";
}
sub MyMethod
{
my $self = shift;
$self->SUPER::MyMethod();
print " MySubClass::MyMethod called!\n";
}
# 调用以上类的主程序
package main;
print "调用 MyClass 方法\n";
$myObject = MyClass->new();
$myObject->MyMethod();
print "调用 MySubClass 方法\n";
$myObject2 = MySubClass->new();
$myObject2->MyMethod();
print "创建一个作用域对象\n";
{
my $myObject2 = MyClass->new();
}
# 自动调用析构函数
print "创建对象\n";
$myObject3 = MyClass->new();
undef $myObject3;
print "脚本执行结束...\n";
# 自动执行析构函数
```

执行以上程序，输出结果为：

```
调用 MyClass 方法
MyClass::new called
MyClass::MyMethod called!
调用 MySubClass 方法
MySubClass::new called
MyClass::new called
MyClass::MyMethod called!
   MySubClass::MyMethod called!
创建一个作用域对象
MyClass::new called
MyClass::DESTROY called
创建对象
MyClass::new called
MyClass::DESTROY called
脚本执行结束...
MyClass::DESTROY called
MySubClass::DESTROY called
```

← Perl Socket 编程

Perl 数据库连接 →

点我分享笔记