Ruby 循环 Ruby 块

Ruby 方法

Ruby 方法与其他编程语言中的函数类似。Ruby 方法用于捆绑一个或多个重复的语句到一个单元中。

方法名应以小写字母开头。如果您以大写字母作为方法名的开头，Ruby 可能会把它当作常量，从而导致不正确地解析调用。

方法应在调用之前定义，否则 Ruby 会产生未定义的方法调用异常。

语法

```
def method_name [( [arg [= default]]...[, * arg [, &expr ]])]
expr..
end
```

所以，您可以定义一个简单的方法，如下所示：

```
def method_name
expr..
end
```

您可以定义一个接受参数的方法，如下所示：

```
def method_name (var1, var2)
expr..
end
```

您可以为参数设置默认值，如果方法调用时未传递必需的参数则使用默认值：

```
def method_name (var1=value1, var2=value2)
expr..
end
```

当您要调用方法时，只需要使用方法名即可，如下所示：

```
method_name
```

但是，当您调用带参数的方法时，您在写方法名时还要带上参数，例如：

```
method_name 25, 30
```

使用带参数方法最大的缺点是调用方法时需要记住参数个数。例如，如果您向一个接受三个参数的方法只传递了两个参数，Ruby 会显示错误。

实例

```
#!/usr/bin/ruby
# -*- coding: UTF-8 -*-
def test(a1="Ruby", a2="Perl")
puts "编程语言为 #{a1}"
puts "编程语言为 #{a2}"
```

```
end
test "C", "C++"
test
```

尝试一下 »

以上实例运行输出结果为：

```
编程语言为 C
编程语言为 C++
编程语言为 Ruby
编程语言为 Perl
```

从方法返回值

Ruby 中的每个方法默认都会返回一个值。这个返回的值是最后一个语句的值。例如：

实例

```
def test
i = 100
j = 10
k = 0
end
```

在调用这个方法时，将返回最后一个声明的变量 k。

Ruby *return* 语句

Ruby 中的 *return* 语句用于从 Ruby 方法中返回一个或多个值。

语法

```
return [expr[`,' expr...]]
```

如果给出超过两个的表达式，包含这些值的数组将是返回值。如果未给出表达式，nil 将是返回值。

实例

```
return
或
return 12
或
return 1,2,3
```

看看下面的实例：

实例

```
#!/usr/bin/ruby
# -*- coding: UTF-8 -*-
def test
i = 100
```

```
j = 200
k = 300
return i, j, k
end
var = test
puts var
```

尝试一下 »

以上实例运行输出结果为：

```
100
200
300
```

可变数量的参数

假设您声明了一个带有两个参数的方法，当您调用该方法时，您同时还需要传递两个参数。

但是，Ruby 允许您声明参数数量可变的方法。让我们看看下面的实例：

实例

```
#!/usr/bin/ruby
# -*- coding: UTF-8 -*-
def sample (*test)
puts "参数个数为 #{test.length}"
for i in 0...test.length
puts "参数值为 #{test[i]}"
end
end
sample "Zara", "6", "F"
sample "Mac", "36", "M", "MCA"
```

尝试一下 »

在这段代码中，您已经声明了一个方法 sample，接受一个参数 test。但是，这个参数是一个变量参数。这意味着参数可以带有不同数量的变量。以上实例运行输出结果为：

```
参数个数为 3
参数值为 Zara
参数值为 6
参数值为 F
参数个数为 4
参数值为 Mac
参数值为 36
参数值为 M
参数值为 MCA
```

类方法

当方法定义在类的外部，方法默认标记为 *private*。另一方面，如果方法定义在类中的，则默认标记为 public。

方法默认的可见性和 *private* 标记可通过模块（Module）的 *public* 或 *private* 改变。

当你想要访问类的方法时，您首先需要实例化类。然后，使用对象，您可以访问类的任何成员。

Ruby 提供了一种不用实例化即可访问方法的方式。让我们看看如何声明并访问类方法：

```
class Accounts
def reading_charge
end
def Accounts.return_date
end
end
```

我们已经知道方法 return_date 是如何声明的。它是通过在类名后跟着一个点号，点号后跟着方法名来声明的。您可以直接访问类方法，如下所示：

```
Accounts.return_date
```

如需访问该方法，您不需要创建类 Accounts 的对象。

Ruby *alias* 语句

这个语句用于为方法或全局变量起别名。别名不能在方法主体内定义。即使方法被重写，方法的别名也保持方法的当前定义。

为编号的全局变量（$1, $2,...）起别名是被禁止的。重写内置的全局变量可能会导致严重的问题。

语法

```
alias 方法名 方法名
alias 全局变量 全局变量
```

实例

```
alias foo bar
alias $MATCH $&
```

在这里，我们已经为 bar 定义了别名为 foo，为 $& 定义了别名为 $MATCH。

Ruby *undef* 语句

这个语句用于取消方法定义。*undef* 不能出现在方法主体内。

通过使用 *undef* 和 *alias*，类的接口可以从父类独立修改，但请注意，在自身内部方法调用时，它可能会破坏程序。

```
undef 方法名
```

实例

下面的实例取消名为 *bar*的方法定义：

```
undef bar
```

Ruby 循环 Ruby 块

点我分享笔记