◆ Ruby 方法

Ruby 模块 ( Module ) →

# Ruby 块

您已经知道 Ruby 如何定义方法以及您如何调用方法。类似地, Ruby 有一个块的概念。

- 块由大量的代码组成。
- 您需要给块取个名称。
- 块中的代码总是包含在大括号 {} 内。
- 块总是从与其具有相同名称的函数调用。这意味着如果您的块名称为 test, 那么您要使用函数 test 来调用这个块。
- 您可以使用 yield 语句来调用块。

## 语法

```
block_name{
statement1
statement2
......
}
```

在这里,您将学到如何使用一个简单的 yield 语句来调用块。您也将学到如何使用带有参数的 yield 语句来调用块。在实例中,您将看到这两种类型的 yield 语句。

# yield 语句

让我们看一个 yield 语句的实例:

### 实例

```
#!/usr/bin/ruby
# -*- coding: UTF-8 -*-
def test
puts "在 test 方法内"
yield
puts "你又回到了 test 方法内"
yield
end
test {puts "你在块内"}
```

以上实例运行结果为:

尝试一下»

```
在 test 方法内
你在块内
你又回到了 test 方法内
你在块内
```

您也可以传递带有参数的 yield 语句。下面是一个实例:

```
实例
```

```
#!/usr/bin/ruby
# -*- coding: UTF-8 -*-
def test
yield 5
puts "在 test 方法内"
yield 100
end
test {|i| puts "你在块 #{i} 内"}
```

## 尝试一下»

以上实例运行结果为:

```
你在块 5 内
在 test 方法内
你在块 100 内
```

在这里, yield 语句后跟着参数。您甚至可以传递多个参数。在块中,您可以在两个竖线之间放置一个变量来接受参数。因此,在上面的代码中, yield 5 语句向 test 块传递值 5 作为参数。

现在,看下面的语句:

```
test {|i| puts "你在块 #{i} 内"}
```

在这里, 值5会在变量i中收到。现在,观察下面的puts语句:

```
puts "你在块 #{i} 内"
```

这个 puts 语句的输出是:

你在块5 内

如果您想要传递多个参数,那么 yield 语句如下所示:

```
yield a, b
```

此时,块如下所示:

```
test {|a, b| statement}
```

参数使用逗号分隔。

## 块和方法

您已经看到块和方法之间是如何相互关联的。您通常使用 yield 语句从与其具有相同名称的方法调用块。因此,代码如下所示:

#### 实例

```
#!/usr/bin/ruby
def test
yield
end
test{ puts "Hello world"}
```

本实例是实现块的最简单的方式。您使用 yield 语句调用 test 块。

但是如果方法的最后一个参数前带有 & , 那么您可以向该方法传递一个块 , 且这个块可被赋给最后一个参数。如果 \* 和 & 同时出现在参数列表中 , & 应放在后面。

## 实例

```
#!/usr/bin/ruby
def test(&block)
block.call
end
test { puts "Hello World!"}
```

以上实例运行结果为:

Hello World!

# BEGIN 和 END 块

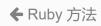
每个 Ruby 源文件可以声明当文件被加载时要运行的代码块(BEGIN 块),以及程序完成执行后要运行的代码块(END 块)。

## 实例

```
#!/usr/bin/ruby
BEGIN {
# BEGIN 代码块
puts "BEGIN 代码块"
}
END {
# END 代码块
puts "END 代码块
puts "END 代码块"
}
# MAIN 代码块
puts "MAIN 代码块
```

一个程序可以包含多个 BEGIN 和 END 块。BEGIN 块按照它们出现的顺序执行。END 块按照它们出现的相反顺序执行。当执行时,上面的程序输出以下结果:

```
BEGIN 代码块
MAIN 代码块
END 代码块
```



Ruby 模块 ( Module ) →

# ② 点我分享笔记