Ruby 条件判断 Ruby 方法

Ruby 循环

Ruby 中的循环用于执行相同的代码块若干次。本章节将详细介绍 Ruby 支持的所有循环语句。

Ruby *while* 语句

语法

```
while conditional [do]
code
end
```

或者

语法

```
while conditional [:]
code
end
```

当 *conditional* 为真时，执行 *code*。

语法中 do 或 : 可以省略不写。但若要在一行内写出 while 式，则必须以 do 或 : 隔开条件式或程式区块。

实例

```
#!/usr/bin/ruby
# -*- coding: UTF-8 -*-
$i = 0
$num = 5
while $i < $num do
puts("在循环语句中 i = #$i" )
$i +=1
end
```

尝试一下 »

以上实例输出结果为：

```
在循环语句中 i = 0
在循环语句中 i = 1
在循环语句中 i = 2
在循环语句中 i = 3
在循环语句中 i = 4
```

Ruby *while* 修饰符

语法

```
code while condition
或者
begin
code
end while conditional
```

当 *conditional* 为真时，执行 *code*。

如果 *while* 修饰符跟在一个没有 *rescue* 或 ensure 子句的 *begin* 语句后面，*code* 会在 *conditional* 判断之前执行一次。

实例

```
#!/usr/bin/ruby
# -*- coding: UTF-8 -*-
$i = 0
$num = 5
begin
puts("在循环语句中 i = #$i" )
$i +=1
end while $i < $num
```

尝试一下 »

以上实例输出结果为：

```
在循环语句中 i = 0
在循环语句中 i = 1
在循环语句中 i = 2
在循环语句中 i = 3
在循环语句中 i = 4
```

Ruby *until* 语句

语法

```
until conditional [do]
code
end
```

当 *conditional* 为假时，执行 *code*。

语法中 do 可以省略不写。但若要在一行内写出 until 式，则必须以 do 隔开条件式或程式区块。

实例

```
#!/usr/bin/ruby
# -*- coding: UTF-8 -*-
$i = 0
$num = 5
until $i > $num do
puts("在循环语句中 i = #$i" )
$i +=1;
end
```

尝试一下 »

以上实例输出结果为：

```
在循环语句中 i = 0
在循环语句中 i = 1
在循环语句中 i = 2
在循环语句中 i = 3
在循环语句中 i = 4
在循环语句中 i = 5
```

Ruby *until* 修饰符

语法

```
code until conditional
或者
begin
code
end until conditional
```

当 *conditional* 为 false 时，执行 *code*。

如果 *until* 修饰符跟在一个没有 *rescue* 或 ensure 子句的 *begin* 语句后面，*code* 会在 *conditional* 判断之前执行一次。

实例

```
#!/usr/bin/ruby
# -*- coding: UTF-8 -*-
$i = 0
$num = 5
begin
puts("在循环语句中 i = #$i" )
$i +=1;
end until $i > $num
```

尝试一下 »

以上实例输出结果为：

```
在循环语句中 i = 0
在循环语句中 i = 1
在循环语句中 i = 2
在循环语句中 i = 3
在循环语句中 i = 4
在循环语句中 i = 5
```

Ruby *for* 语句

语法

```
for variable [, variable ...] in expression [do]
code
end
```

先计算表达式得到一个对象，然后针对 *expression* 中的每个元素分别执行一次 *code*。

实例

```
#!/usr/bin/ruby
# -*- coding: UTF-8 -*-
for i in 0..5
puts "局部变量的值为 #{i}"
end
```

尝试一下 »

在这里，我们已经定义了范围 0..5。语句 for i in 0..5 允许 i 的值从 0 到 5（包含 5）。

以上实例输出结果为：

```
局部变量的值为 0
局部变量的值为 1
局部变量的值为 2
局部变量的值为 3
局部变量的值为 4
局部变量的值为 5
```

for...in 循环几乎是完全等价于：

```
(expression).each do |variable[, variable...]| code end
```

但是，for 循环不会为局部变量创建一个新的作用域。

语法中 do 可以省略不写。但若要在一行内写出 for 式，则必须以 do 隔开条件式或程式区块。

实例

```
#!/usr/bin/ruby
# -*- coding: UTF-8 -*-
(0..5).each do |i|
puts "局部变量的值为 #{i}"
end
```

尝试一下 »

以上实例输出结果为：

```
局部变量的值为 0
局部变量的值为 1
局部变量的值为 2
局部变量的值为 3
局部变量的值为 4
局部变量的值为 5
```

Ruby *break* 语句

语法

```
break
```

终止最内部的循环。如果在块内调用，则终止相关块的方法（方法返回 nil）。

实例

```
#!/usr/bin/ruby
# -*- coding: UTF-8 -*-
for i in 0..5
if i > 2 then
break
end
puts "局部变量的值为 #{i}"
end
```

尝试一下 »

以上实例输出结果为：

```
局部变量的值为 0
局部变量的值为 1
局部变量的值为 2
```

Ruby *next* 语句

语法

```
next
```

跳到循环的下一个迭代。如果在块内调用，则终止块的执行（*yield* 表达式返回 nil）。

实例

```
#!/usr/bin/ruby
# -*- coding: UTF-8 -*-
for i in 0..5
if i < 2 then
next
end
```

```
puts "局部变量的值为 #{i}"
end
```

尝试一下 »

以上实例输出结果为：

```
局部变量的值为 2
局部变量的值为 3
局部变量的值为 4
局部变量的值为 5
```

Ruby *redo* 语句

语法

```
redo
```

重新开始最内部循环的该次迭代，不检查循环条件。如果在块内调用，则重新开始 *yield* 或 *call*。

实例

```
#!/usr/bin/ruby
# -*- coding: UTF-8 -*-
for i in 0..5
if i < 2 then
puts "局部变量的值为 #{i}"
redo
end
end
```

这将产生以下结果，并会进入一个无限循环：

```
局部变量的值为 0
局部变量的值为 0
.............................
```

Ruby *retry* 语句

注意：1.9以及之后的版本不支持在循环中使用retry。

语法

```
retry
```

如果 *retry* 出现在 begin 表达式的 rescue 子句中，则从 begin 主体的开头重新开始。

```
begin
do_something # 抛出的异常
rescue
# 处理错误
retry # 重新从 begin 开始
end
```

如果 retry 出现在迭代内、块内或者 for 表达式的主体内，则重新开始迭代调用。迭代的参数会重新评估。

```
for i in 1..5
retry if some_condition # 重新从 i == 1 开始
end
```

实例

```
#!/usr/bin/ruby
# -*- coding: UTF-8 -*-
for i in 1..5
retry if i > 2
puts "局部变量的值为 #{i}"
end
```

这将产生以下结果，并会进入一个无限循环：

```
局部变量的值为 1
局部变量的值为 2
局部变量的值为 1
局部变量的值为 2
局部变量的值为 1
局部变量的值为 2
.............................
```

← Ruby 条件判断 Ruby 方法 →

点我分享笔记