Ruby 日期 & 时间（Date & Time） Ruby 迭代器

# Ruby 范围（Range）

范围（Range）无处不在：a 到 z、 0 到 9、等等。Ruby 支持范围，并允许我们以不同的方式使用范围：

- 作为序列的范围
- 作为条件的范围
- 作为间隔的范围

## 作为序列的范围

范围的第一个也是最常见的用途是表达序列。序列有一个起点、一个终点和一个在序列产生连续值的方式。

Ruby 使用 ".." 和 "..." 范围运算符创建这些序列。两点形式创建一个包含指定的最高值的范围，三点形式创建一个不包含指定的最高值的范围。

```
(1..5) #==> 1, 2, 3, 4, 5
(1...5) #==> 1, 2, 3, 4
('a'..'d') #==> 'a', 'b', 'c', 'd'
```

序列 1..100 是一个 *Range* 对象，包含了两个 *Fixnum* 对象的引用。如果需要，您可以使用 *to_a* 方法把范围转换为列表。尝试下面的实例：

**实例**

```
#!/usr/bin/ruby
$, =", " # Array 值分隔符
range1 = (1..10).to_a
range2 = ('bar'..'bat').to_a
puts "#{range1}"
puts "#{range2}"
```

尝试一下 »

以上实例运行输出结果为：

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
["bar", "bas", "bat"]
```

范围实现了让您可以遍历它们的方法，您可以通过多种方式检查它们的内容：

**实例**

```
#!/usr/bin/ruby
# -*- coding: UTF-8 -*-
# 指定范围
digits = 0..9
puts digits.include?(5)
```

```
ret = digits.min
puts "最小值为 #{ret}"
ret = digits.max
puts "最大值为 #{ret}"
ret = digits.reject {|i| i < 5 }
puts "不符合条件的有 #{ret}"
digits.each do |digit|
puts "在循环中 #{digit}"
end
```

尝试一下 »

以上实例运行输出结果为：

```
true
最小值为 0
最大值为 9
不符合条件的有 [5, 6, 7, 8, 9]
在循环中 0
在循环中 1
在循环中 2
在循环中 3
在循环中 4
在循环中 5
在循环中 6
在循环中 7
在循环中 8
在循环中 9
```

## 作为条件的范围

范围也可以用作条件表达式。例如，下面的代码片段从标准输入打印行，其中每个集合的第一行包含单词 *start*，最后一行包含单词 *end.*：

```
while gets
print if /start/../end/
end
```

范围可以用在 case 语句中：

**实例**

```
#!/usr/bin/ruby
# -*- coding: UTF-8 -*-
score = 70
result = case score
when 0..40
"糟糕的分数"
when 41..60
"快要及格"
```

```
when 61..70
"及格分数"
when 71..100
"良好分数"
else
"错误的分数"
end
puts result
```

尝试一下 »

以上实例运行输出结果为：

```
及格分数
```

## 作为间隔的范围

范围的最后一个用途是间隔检测：检查指定值是否在指定的范围内。需要使用 === 相等运算符来完成计算。

**实例**

```
#!/usr/bin/ruby
# -*- coding: UTF-8 -*-
if ((1..10) === 5)
puts "5 在 (1..10)"
end
if (('a'..'j') === 'c')
puts "c 在 ('a'..'j')"
end
if (('a'..'j') === 'z')
puts "z 在 ('a'..'j')"
end
```

尝试一下 »

以上实例运行输出结果为：

```
5 在 (1..10)
c 在 ('a'..'j')
```

← Ruby 日期 & 时间（Date & Time）

Ruby 迭代器 →

点我分享笔记