Ruby 变量 Ruby 注释

Ruby 运算符

Ruby 支持一套丰富的运算符。大多数运算符实际上是方法调用。例如，a + b 被解释为 a.+(b)，其中指向变量 a 的 + 方法被调用，b 作为方法调用的参数。

对于每个运算符（ + - * / % ** & | ^ << >> && || ），都有一个相对应的缩写赋值运算符（ += -= 等等 ）。

Ruby 算术运算符

假设变量 a 的值为 10，变量 b 的值为 20，那么：

运算符	描述	实例
+	加法 - 把运算符两边的操作数相加	a + b 将得到 30
-	减法 - 把左操作数减去右操作数	a - b 将得到 -10
*	乘法 - 把运算符两边的操作数相乘	a * b 将得到 200
/	除法 - 把左操作数除以右操作数	b / a 将得到 2
%	求模 - 把左操作数除以右操作数，返回余数	b % a 将得到 0
**	指数 - 执行指数计算	a**b 将得到 10 的 20 次方

Ruby 比较运算符

假设变量 a 的值为 10，变量 b 的值为 20，那么：

运算符	描述	实例
==	检查两个操作数的值是否相等，如果相等则条件为真。	(a == b) 不为真。
!=	检查两个操作数的值是否相等，如果不相等则条件为真。	(a != b) 为真。
>	检查左操作数的值是否大于右操作数的值，如果是则条件为真。	(a > b) 不为真。
<	检查左操作数的值是否小于右操作数的值，如果是则条件为真。	(a < b) 为真。
>=	检查左操作数的值是否大于或等于右操作数的值，如果是则条件为真。	(a >= b) 不为真。

<=	检查左操作数的值是否小于或等于右操作数的值，如果是则条件为真。	(a <= b) 为真。
<=>	联合比较运算符。如果第一个操作数等于第二个操作数则返回 0，如果第一个操作数大于第二个操作数则返回 1，如果第一个操作数小于第二个操作数则返回 -1。	(a <=> b) 返回 -1。
===	用于测试 *case* 语句的 when 子句内的相等。	(1...10) === 5 返回 true。
.eql?	如果接收器和参数具有相同的类型和相等的值，则返回 true。	1 == 1.0 返回 true，但是 1**.eql?**(1.0) 返回 false。
equal?	如果接收器和参数具有相同的对象 id，则返回 true。	如果 aObj 是 bObj 的副本，那么 aObj == bObj 返回 true，a**.equal?**bObj 返回 false，但是 a**.equal?**aObj 返回 true。

Ruby 赋值运算符

假设变量 a 的值为 10，变量 b 的值为 20，那么：

运算符	描述	实例
=	简单的赋值运算符，把右操作数的值赋给左操作数	c = a + b 将把 a + b 的值赋给 c
+=	加且赋值运算符，把右操作数加上左操作数的结果赋值给左操作数	c += a 相当于 c = c + a
-=	减且赋值运算符，把左操作数减去右操作数的结果赋值给左操作数	c -= a 相当于 c = c - a
*=	乘且赋值运算符，把右操作数乘以左操作数的结果赋值给左操作数	c *= a 相当于 c = c * a
/=	除且赋值运算符，把左操作数除以右操作数的结果赋值给左操作数	c /= a 相当于 c = c / a
%=	求模且赋值运算符，求两个操作数的模赋值给左操作数	c %= a 相当于 c = c % a
**=	指数且赋值运算符，执行指数计算，并赋值给左操作数	c **= a 相当于 c = c ** a

Ruby 并行赋值

Ruby 也支持变量的并行赋值。这使得多个变量可以通过一行的 Ruby 代码进行初始化。例如：

```
a = 10
b = 20
c = 30
```

使用并行赋值可以更快地声明：

```
a, b, c = 10, 20, 30
```

并行赋值在交换两个变量的值时也很有用：

```
a, b = b, c
```

Ruby 位运算符

位运算符作用于位，并逐位执行操作。

假设如果 a = 60，且 b = 13，现在以二进制格式，它们如下所示：

a = 0011 1100

b = 0000 1101

---

a&b = 0000 1100

a|b = 0011 1101

a^b = 0011 0001

~a = 1100 0011

下表列出了 Ruby 支持的位运算符。

运算符	描述	实例
&	如果同时存在于两个操作数中，二进制 AND 运算符复制一位到结果中。	(a & b) 将得到 12，即为 0000 1100
|	如果存在于任一操作数中，二进制 OR 运算符复制一位到结果中。	(a | b) 将得到 61，即为 0011 1101
^	如果存在于其中一个操作数中但不同时存在于两个操作数中，二进制异或运算符复制一位到结果中。	(a ^ b) 将得到 49，即为 0011 0001
~	二进制补码运算符是一元运算符，具有"翻转"位效果，即0变成1，1变成0。	(~a) 将得到 -61，即为 1100 0011，一个有符号二进制数的补码形式。
<<	二进制左移运算符。左操作数的值向左移动右操作数指定的位数。	a << 2 将得到 240，即为 1111 0000
>>	二进制右移运算符。左操作数的值向右移动右操作数指定的位数。	a >> 2 将得到 15，即为 0000 1111

Ruby 逻辑运算符

下表列出了 Ruby 支持的逻辑运算符。

假设变量 a 的值为 10，变量 b 的值为 20，那么：

运算符	描述	实例
and	称为逻辑与运算符。如果两个操作数都为真，则条件为真。	(a and b) 为真。
or	称为逻辑或运算符。如果两个操作数中有任意一个非零，则条件为真。	(a or b) 为真。
&&	称为逻辑与运算符。如果两个操作数都非零，则条件为真。	(a && b) 为真。
||	称为逻辑或运算符。如果两个操作数中有任意一个非零，则条件为真。	(a || b) 为真。
!	称为逻辑非运算符。用来逆转操作数的逻辑状态。如果条件为真则逻辑非运算符将使其为假。	!(a && b) 为假。
not	称为逻辑非运算符。用来逆转操作数的逻辑状态。如果条件为真则逻辑非运算符将使其为假。	not(a && b) 为假。

Ruby 三元运算符

有一个以上的操作称为三元运算符。第一个计算表达式的真假值，然后根据这个结果决定执行后边两个语句中的一个。条件运算符的语法如下：

运算符	描述	实例
? :	条件表达式	如果条件为真 ? 则值为 X : 否则值为 Y

Ruby 范围运算符

在 Ruby 中，序列范围用于创建一系列连续的值 - 包含起始值、结束值（视情况而定）和它们之间的值。

在 Ruby 中，这些序列是使用 ".." 和 "..." 范围运算符来创建的。两点形式创建的范围包含起始值和结束值，三点形式创建的范围只包含起始值不包含结束值。

运算符	描述	实例
..	创建一个从开始点到结束点的范围（包含结束点）	1..10 创建从 1 到 10 的范围

...	创建一个从开始点到结束点的范围（不包含结束点）	1...10 创建从 1 到 9 的范围

Ruby defined? 运算符

defined? 是一个特殊的运算符，以方法调用的形式来判断传递的表达式是否已定义。它返回表达式的描述字符串，如果表达式未定义则返回 *nil*。

下面是 defined? 运算符的各种用法：

用法 1

```
defined? variable # 如果 variable 已经初始化，则为 True
```

例如：

```
foo = 42
defined? foo # => "local-variable"
defined? $_ # => "global-variable"
defined? bar # => nil（未定义）
```

用法 2

```
defined? method_call # 如果方法已经定义，则为 True
```

例如：

```
defined? puts # => "method"
defined? puts(bar) # => nil（在这里 bar 未定义）
defined? unpack # => nil（在这里未定义）
```

用法 3

```
# 如果存在可被 super 用户调用的方法，则为 True
defined? super
```

例如：

```
defined? super # => "super"（如果可被调用）
defined? super # => nil（如果不可被调用）
```

用法 4

```
defined? yield # 如果已传递代码块，则为 True
```

例如：

```
defined? yield # => "yield"（如果已传递块）
defined? yield # => nil（如果未传递块）
```

Ruby 点运算符 "." 和双冒号运算符 "::"

你可以通过在方法名称前加上类或模块名称和 `.` 来调用类或模块中的方法。你可以使用类或模块名称和两个冒号 `::` 来引用类或模块中的常量。

`::` 是一元运算符，允许在类或模块内定义常量、实例方法和类方法，可以从类或模块外的任何地方进行访问。

请记住：在 Ruby 中，类和方法也可以被当作常量。

你只需要在表达式的常量名前加上 :: 前缀，即可返回适当的类或模块对象。

如果 `::` 前的表达式为类或模块名称，则返回该类或模块内对应的常量值；如果 `::` 前未没有前缀表达式，则返回主Object类中对应的常量值。 。

下面是两个实例：

```
MR_COUNT = 0 # 定义在主 Object 类上的常量
module Foo
MR_COUNT = 0
::MR_COUNT = 1 # 设置全局计数为 1
MR_COUNT = 2 # 设置局部计数为 2
end
puts MR_COUNT # 这是全局常量
puts Foo::MR_COUNT # 这是 "Foo" 的局部常量
```

第二个实例：

```
CONST = ' out there'
class Inside_one
CONST = proc {' in there'}
def where_is_my_CONST
::CONST + ' inside one'
end
end
class Inside_two
CONST = ' inside two'
def where_is_my_CONST
CONST
end
end
puts Inside_one.new.where_is_my_CONST
puts Inside_two.new.where_is_my_CONST
puts Object::CONST + Inside_two::CONST
puts Inside_two::CONST + CONST
puts Inside_one::CONST
puts Inside_one::CONST.call + Inside_two::CONST
```

Ruby 运算符的优先级

下表按照运算符的优先级从高到低列出了所有的运算符。

方法	运算符	描述
是	::	常量解析运算符

是	[ ] [ ]=	元素引用、元素集合
是	**	指数
是	! ~ + -	非、补、一元加、一元减（最后两个的方法名为 +@ 和 -@ ）
是	* / %	乘法、除法、求模
是	+ -	加法和减法
是	>> <<	位右移、位左移
是	&	位与
是	^ |	位异或、位或
是	<= < > >=	比较运算符
是	<=> == === != =~ !~	相等和模式匹配运算符（!= 和 !~ 不能被定义为方法 ）
	&&	逻辑与
	||	逻辑或
	.. ...	范围（包含、不包含 ）
	? :	三元 if-then-else
	= %= { /= -= += |= &= >>= <<= *= &&= ||= **=	赋值
	defined?	检查指定符号是否已定义
	not	逻辑否定
	or and	逻辑组成

注意：在方法列标识为 *是* 的运算符实际上是方法，因此可以被重载。

Ruby 变量 Ruby 注释