Ruby 环境变量 Ruby 类和对象

Ruby 语法

让我们编写一个简单的 Ruby 程序。所有的 Ruby 文件扩展名都是 **.rb**。所以，把下面的源代码放在 test.rb 文件中。

实例

```
#!/usr/bin/ruby -w
puts "Hello, Ruby!";
```

运行实例 »

在这里，假设您的 /usr/bin 目录下已经有可用的 Ruby 解释器。现在，尝试运行这个程序，如下所示：

```
$ ruby test.rb
```

这将会产生下面的结果：

```
Hello, Ruby!
```

您已经看到了一个简单的 Ruby 程序，现在让我们看看一些 Ruby 语法相关的基本概念：

Ruby 程序中的空白

在 Ruby 代码中的空白字符，如空格和制表符一般会被忽略，除非当它们出现在字符串中时才不会被忽略。然而，有时候它们用于解释模棱两可的语句。当启用 -w 选项时，这种解释会产生警告。

实例：

```
a + b 被解释为 a+b （这是一个局部变量）
a  +b 被解释为 a(+b) （这是一个方法调用）
```

Ruby 程序中的行尾

Ruby 把分号和换行符解释为语句的结尾。但是，如果 Ruby 在行尾遇到运算符，比如 +、- 或反斜杠，它们表示一个语句的延续。

Ruby 标识符

标识符是变量、常量和方法的名称。Ruby 标识符是大小写敏感的。这意味着 Ram 和 RAM 在 Ruby 中是两个不同的标识符。Ruby 标识符的名称可以包含字母、数字和下划线字符（ _ ）。

保留字

下表列出了 Ruby 中的保留字。这些保留字不能作为常量或变量的名称。但是，它们可以作为方法名。

BEGIN	do	next	then
END	else	nil	true
alias	elsif	not	undef
and	end	or	unless
begin	ensure	redo	until
break	false	rescue	when
case	for	retry	while
class	if	return	while
def	in	self	__FILE__
defined?	module	super	__LINE__

Ruby 中的 Here Document

"Here Document" 是指建立多行字符串。在 << 之后，您可以指定一个字符串或标识符来终止字符串，且当前行之后直到终止符为止的所有行是字符串的值。

如果终止符用引号括起，引号的类型决定了面向行的字符串类型。请注意<< 和终止符之间必须没有空格。

下面是不同的实例：

实例

```
#!/usr/bin/ruby -w
# -*- coding : utf-8 -*-
print <<EOF
这是第一种方式创建here document 。
多行字符串。
EOF
print <<"EOF"; # 与上面相同
这是第二种方式创建here document 。
多行字符串。
EOF
print <<`EOC` # 执行命令
echo hi there
echo lo there
EOC
print <<"foo", <<"bar" # 您可以把它们进行堆叠
I said foo.
foo
I said bar.
bar
```

运行实例 »

这将产生以下结果：

```
这是第一种方式创建here document 。
多行字符串。
这是第二种方式创建here document 。
多行字符串。
hi there
lo there
I said foo.
I said bar.
```

Ruby *BEGIN* 语句

语法

```
BEGIN {
   code
}
```

声明 *code* 会在程序运行之前被调用。

实例

```
#!/usr/bin/ruby
puts "这是主 Ruby 程序"
BEGIN {
puts "初始化 Ruby 程序"
}
```

这将产生以下结果：

```
初始化 Ruby 程序
这是主 Ruby 程序
```

Ruby *END* 语句

语法

```
END {
   code
}
```

声明 *code* 会在程序的结尾被调用。

实例

```
#!/usr/bin/ruby
puts "这是主 Ruby 程序"
END {
puts "停止 Ruby 程序"
}
BEGIN {
puts "初始化 Ruby 程序"
}
```

这将产生以下结果：

```
初始化 Ruby 程序
这是主 Ruby 程序
停止 Ruby 程序
```

Ruby 注释

注释会对 Ruby 解释器隐藏一行，或者一行的一部分，或者若干行。您可以在行首使用字符（ # ）：

```
# 我是注释，请忽略我。
```

或者，注释可以跟着语句或表达式的同一行的后面：

```
name = "Madisetti" # 这也是注释
```

您可以注释多行，如下所示：

```
# 这是注释。
# 这也是注释。
# 这也是注释。
# 这还是注释。
```

下面是另一种形式。这种块注释会对解释器隐藏 =begin/=end 之间的行：

```
=begin
这是注释。
这也是注释。
这也是注释。
这还是注释。
=end
```

← Ruby 环境变量 Ruby 类和对象 →

here document 介绍

构建一个 **here document** 最通用的语法是 `<<` 紧跟一个标识符，从下一行开始是想要引用的文字，然后再在单独的一行用相同的标识符关闭。

```
puts <<EOF
这是第一行
这是第二行
EOF
```

执行输出结果为：

```
这是第一行
这是第二行
```

更多用法可查看以下实例：

```
#!/usr/bin/ruby -w
# -*- coding : utf-8 -*-

# (<<) here doucument 感觉本来单行是一个整体变成是一种多行作为一个整体进行导入的方式和 sh 语法相似

print <<EOF
    这是第一种方式创建 here document
    多行字符串。
EOF

print <<"EOF";                # 与上面相同
    这是第二种方式创建here document
    多行字符串。
EOF

print <<`EOC`                 # 执行命令
    ls -al;
    ps -a
EOC

print <<"foo", <<"bar"          # 您可以把它们进行堆叠
    I said foo.
foo
    I said bar.
bar
text =  <<`foo`             # 您可以把它们进行转存
    cat /etc/passwd
foo

puts text

File.open("/home/abc","w") do |io|
```

```
    io.write(text)
end
puts "-------------------------"
exec "ls -al /home/ && cat /home/abc"
```

日语初学者 9个月前 (06-13)