Ruby 块 Ruby 字符串（String）

Ruby 模块（Module）

模块（Module）是一种把方法、类和常量组合在一起的方式。模块（Module）为您提供了两大好处。

- 模块提供了一个*命名空间*和避免名字冲突。
- 模块实现了 *mixin* 装置。

模块（Module）定义了一个命名空间，相当于一个沙盒，在里边您的方法和常量不会与其他地方的方法常量冲突。

模块类似与类，但有一下不同：

- 模块不能实例化
- 模块没有子类
- 模块只能被另一个模块定义

语法

```
module Identifier
statement1
statement2
...........
end
```

模块常量命名与类常量命名类似，以大写字母开头。方法定义看起来也相似：模块方法定义与类方法定义类似。

通过类方法，您可以在类方法名称前面放置模块名称和一个点号来调用模块方法，您可以使用模块名称和两个冒号来引用一个常量。

实例

```
#!/usr/bin/ruby
# 定义在 trig.rb 文件中的模块
module Trig
PI = 3.141592654
def Trig.sin(x)
# ..
end
def Trig.cos(x)
# ..
end
end
```

我们可以定义多个函数名称相同但是功能不同的模块：

实例

```
#!/usr/bin/ruby
# 定义在 moral.rb 文件中的模块
module Moral
```

```
VERY_BAD = 0
BAD = 1
def Moral.sin(badness)
# ...
end
end
```

就像类方法，当您在模块中定义一个方法时，您可以指定在模块名称后跟着一个点号，点号后跟着方法名。

Ruby *require* 语句

require 语句类似于 C 和 C++ 中的 include 语句以及 Java 中的 import 语句。如果一个第三方的程序想要使用任何已定义的模块，则可以简单地使用 Ruby *require* 语句来加载模块文件：

语法

语法

```
require filename
```

在这里，文件扩展名 **.rb** 不是必需的。

实例

```
$LOAD_PATH << '.'
require 'trig.rb'
require 'moral'
y = Trig.sin(Trig::PI/4)
wrongdoing = Moral.sin(Moral::VERY_BAD)
```

在这里，我们使用 **$LOAD_PATH << '.'** 让 Ruby 知道必须在当前目录中搜索被引用的文件。如果您不想使用 $LOAD_PATH，那么您可以使用 **require_relative** 来从一个相对目录引用文件。

注意：在这里，文件包含相同的函数名称。所以，这会在引用调用程序时导致代码模糊，但是模块避免了这种代码模糊，而且我们可以使用模块的名称调用适当的函数。

Ruby *include* 语句

您可以在类中嵌入模块。为了在类中嵌入模块，您可以在类中使用 *include* 语句：

语法

```
include modulename
```

如果模块是定义在一个单独的文件中，那么在嵌入模块之前就需要使用 *require* 语句引用该文件。

实例

假设下面的模块写在 *support.rb* 文件中。

```
module Week
FIRST_DAY = "Sunday"
def Week.weeks_in_month
puts "You have four weeks in a month"
end
```

```
def Week.weeks_in_year
puts "You have 52 weeks in a year"
end
end
```

现在，您可以在类中引用该模块，如下所示：

实例

```
#!/usr/bin/ruby
$LOAD_PATH << '.'
require "support"
class Decade
include Week
no_of_yrs=10
def no_of_months
puts Week::FIRST_DAY
number=10*12
puts number
end
end
d1=Decade.new
puts Week::FIRST_DAY
Week.weeks_in_month
Week.weeks_in_year
d1.no_of_months
```

这将产生以下结果：

```
Sunday
You have four weeks in a month
You have 52 weeks in a year
Sunday
120
```

Ruby 中的 Mixins

在阅读本节之前，您需要初步了解面向对象的概念。

当一个类可以从多个父类继承类的特性时，该类显示为多重继承。

Ruby 不直接支持多重继承，但是 Ruby 的模块（Module）有另一个神奇的功能。它几乎消除了多重继承的需要，提供了一种名为 *mixin* 的装置。

Ruby 没有真正实现多重继承机制，而是采用成为mixin技术作为替代品。将模块include到类定义中，模块中的方法就mix进了类中。

让我们看看下面的示例代码，深入了解 mixin：

实例

```
module A
def a1
end
```

```
def a2
end
end
module B
def b1
end
def b2
end
end
class Sample
include A
include B
def s1
end
end
samp=Sample.new
samp.a1
samp.a2
samp.b1
samp.b2
samp.s1
```

- 模块 A 由方法 a1 和 a2 组成。
- 模块 B 由方法 b1 和 b2 组成。
- 类 Sample 包含了模块 A 和 B。
- 类 Sample 可以访问所有四个方法，即 a1、a2、b1 和 b2。

因此，您可以看到类 Sample 继承了两个模块，您可以说类 Sample 使用了多重继承或 *mixin* 。

Ruby 块 Ruby 字符串（ String ）