Ruby 异常 Ruby 正则表达式

Ruby 面向对象

Ruby 是纯面向对象的语言，Ruby 中的一切都是以对象的形式出现。Ruby 中的每个值都是一个对象，即使是最原始的东西：字符串、数字，甚至连 true 和 false 都是对象。类本身也是一个*对象*，是 *Class* 类的一个实例。本章将向您讲解所有与 Ruby 面向对象相关的主要功能。

类用于指定对象的形式，它结合了数据表示法和方法，把数据整理成一个整齐的包。类中的数据和方法被称为类的成员。

Ruby 类定义

当您定义一个类时，您实际是定义了一个数据类型的蓝图。这实际上并没有定义任何的数据，而是定义了类的名称意味着什么，也就是说，定义了类的对象将由什么组成，以及在该对象上能执行什么操作。

类定义以关键字 **class** 开始，后跟**类名称**，最后以一个 **end** 进行分隔表示终止该类定义。例如，我们使用关键字 class 来定义 Box 类，如下所示：

```
class Box
code
end
```

按照惯例，名称必须以大写字母开头，如果包含多个单词，每个单词首字母大写，但此间没有分隔符（例如：CamelCase）。

定义 Ruby 对象

类提供了对象的蓝图，所以基本上，对象是根据类进行创建的。我们使用 **new** 关键字声明类的对象。下面的语句声明了类 Box 的两个对象：

```
box1 = Box.new
box2 = Box.new
```

initialize 方法

initialize 方法是一个标准的 Ruby 类方法，是类的构造函数，与其他面向对象编程语言中的 **constructor** 工作原理类似。当您想要在创建对象的同时初始化一些类变量，initialize 方法就派上用场了。该方法带有一系列参数，与其他 Ruby 方法一样，使用该方法时，必须在前面放置 **def** 关键字，如下所示：

```
class Box
def initialize(w,h)
@width, @height = w, h
end
end
```

实例变量

实例变量是类属性，它们在使用类创建对象时就变成对象的属性。每个对象的属性是单独赋值的，和其他对象之间不共享值。

在类的内部，是使用 @ 运算符访问这些属性，在类的外部，则是使用称为**访问器方法**的**公共**方法进行访问。下面我们以上面定义的类 **Box** 为实例，把 @width 和 @height 作为类 Box 的实例变量。

```
class Box
def initialize(w,h)
# 给实例变量赋值
@width, @height = w, h
end
end
```

访问器(getter) & 设置器(setter)方法

为了在类的外部读取类中已定义的变量，我们可以通过定义访问器(getter)方法来访问。下面的实例演示了访问器方法的用法：

实例

```
#!/usr/bin/ruby -w
# 定义类
class Box
# 构造函数
def initialize(w,h)
@width, @height = w, h
end
# 访问器方法
def printWidth
@width
end
def printHeight
@height
end
end
# 创建对象，初始化盒子的高度与宽度
box = Box.new(10, 20)
# 使用访问器方法
x = box.printWidth()
y = box.printHeight()
puts "盒子宽度 : #{x}"
puts "盒子高度 : #{y}"
```

尝试一下 »

当上面的代码执行时，它会产生以下结果：

```
盒子宽度 : 10
盒子高度 : 20
```

与用于访问变量值的访问器方法类似，Ruby 提供了一种在类的外部将参数传入类中已定义的变量，也就是所谓的**设置器方法**，定义如下：

实例

```
#!/usr/bin/ruby -w
# 定义类
class Box
# 构造器方法
def initialize(w,h)
@width, @height = w, h
end
# 访问器方法
def getWidth
@width
end
def getHeight
@height
end
# 设置器方法
def setWidth=(value)
@width = value
end
def setHeight=(value)
@height = value
end
end
# 创建对象
box = Box.new(10, 20)
# 使用设置器方法
box.setWidth = 30
box.setHeight = 50
# 使用访问器方法
x = box.getWidth()
y = box.getHeight()
puts "盒子宽度 : #{x}"
puts "盒子高度 : #{y}"
```

尝试一下 »

当上面的代码执行时，它会产生以下结果：

```
盒子宽度 : 30
盒子高度 : 50
```

由于两种方法非常常用，Ruby 定义了 ***attr_accessor :variable_name、attr_reader :variable_name、attr_writer :variable_name*** *三种属性声明方法。其中：* ***accessor=reader+writer。***

同时注意：变量名前一定要带 `:` *，变量名之间要用* `,` *分割。*

实例方法

实例方法的定义与其他方法的定义一样，都是使用 **def** 关键字，但它们只能通过类实例来使用，如下面实例所示。它们的功能不限于访问实例变量，也能按照您的需求做更多其他的任务。

实例

```
#!/usr/bin/ruby -w
# 定义类
class Box
# 构造方法
def initialize(w,h)
@width, @height = w, h
end
# 实例方法
def getArea
@width * @height
end
end
# 创建对象
box = Box.new(10, 20)
# 调用实例方法
a = box.getArea()
puts "Area of the box is : #{a}"
```

尝试一下 »

当上面的代码执行时，它会产生以下结果：

```
Area of the box is : 200
```

类方法 & 类变量

类变量是在类的所有实例中共享的变量。换句话说，类变量的实例可以被所有的对象实例访问。类变量以两个 @ 字符（@@）作为前缀，类变量必须在类定义中被初始化，如下面实例所示。

类方法使用 **def self.methodname()** 定义，类方法以 end 分隔符结尾。类方法可使用带有类名称的 **classname.methodname** 形式调用，如下面实例所示：

实例

```
#!/usr/bin/ruby -w
class Box
# 初始化类变量
@@count = 0
def initialize(w,h)
# 给实例变量赋值
@width, @height = w, h
@@count += 1
end
def self.printCount()
puts "Box count is : #@@count"
end
end
```

```ruby
# 创建两个对象
box1 = Box.new(10, 20)
box2 = Box.new(30, 100)
# 调用类方法来输出盒子计数
Box.printCount()
```

尝试一下 »

当上面的代码执行时，它会产生以下结果：

```
Box count is : 2
```

to_s 方法

您定义的任何类都有一个 **to_s** 实例方法来返回对象的字符串表示形式。下面是一个简单的实例，根据 width 和 height 表示 Box 对象：

实例

```ruby
#!/usr/bin/ruby -w
class Box
# 构造器方法
def initialize(w,h)
@width, @height = w, h
end
# 定义 to_s 方法
def to_s
"(w:#@width,h:#@height)" # 对象的字符串格式
end
end
# 创建对象
box = Box.new(10, 20)
# 自动调用 to_s 方法
puts "String representation of box is : #{box}"
```

尝试一下 »

当上面的代码执行时，它会产生以下结果：

```
String representation of box is : (w:10,h:20)
```

访问控制

Ruby 为您提供了三个级别的实例方法保护，分别是 **public、private 或 protected**。Ruby 不在实例和类变量上应用任何访问控制。

- **Public 方法：** Public 方法可被任意对象调用。默认情况下，方法都是 public 的，除了 initialize 方法总是 private 的。
- **Private 方法：** Private 方法不能从类外部访问或查看。只有类方法可以访问私有成员。

- **Protected 方法：** Protected 方法只能被类及其子类的对象调用。访问也只能在类及其子类内部进行。

下面是一个简单的实例，演示了这三种修饰符的语法：

实例

```
#!/usr/bin/ruby -w
# 定义类
class Box
# 构造器方法
def initialize(w,h)
@width, @height = w, h
end
# 实例方法默认是 public 的
def getArea
getWidth() * getHeight
end
# 定义 private 的访问器方法
def getWidth
@width
end
def getHeight
@height
end
# make them private
private :getWidth, :getHeight
# 用于输出面积的实例方法
def printArea
@area = getWidth() * getHeight
puts "Big box area is : #@area"
end
# 让实例方法是 protected 的
protected :printArea
end
# 创建对象
box = Box.new(10, 20)
# 调用实例方法
a = box.getArea()
puts "Area of the box is : #{a}"
# 尝试调用 protected 的实例方法
box.printArea()
```

尝试一下 »

当上面的代码执行时，它会产生以下结果。在这里，第一种方法调用成功，但是第二方法会产生一个问题。

```
Area of the box is : 200
test.rb:42: protected method `printArea' called for #
<Box:0xb7f11280 @height=20, @width=10> (NoMethodError)
```

类的继承

继承，是面向对象编程中最重要的概念之一。继承允许我们根据另一个类定义一个类，这样使得创建和维护应用程序变得更加容易。

继承有助于重用代码和快速执行，不幸的是，Ruby 不支持多继承，但是 Ruby 支持 **mixins**。mixin 就像是多继承的一个特定实现，在多继承中，只有接口部分是可继承的。

当创建类时，程序员可以直接指定新类继承自某个已有类的成员，这样就不用从头编写新的数据成员和成员函数。这个已有类被称为**基类或父类**，新类被称为**派生类或子类**。

Ruby 也提供了子类化的概念，子类化即继承，下面的实例解释了这个概念。扩展一个类的语法非常简单。只要添加一个 < 字符和父类的名称到类语句中即可。例如，下面定义了类 *BigBox* 是 *Box* 的子类：

实例

```
#!/usr/bin/ruby -w
# 定义类
class Box
# 构造器方法
def initialize(w,h)
@width, @height = w, h
end
# 实例方法
def getArea
@width * @height
end
end
# 定义子类
class BigBox < Box
# 添加一个新的实例方法
def printArea
@area = @width * @height
puts "Big box area is : #@area"
end
end
# 创建对象
box = BigBox.new(10, 20)
# 输出面积
box.printArea()
```

尝试一下 »

当上面的代码执行时，它会产生以下结果：

```
Big box area is : 200
```

方法重载

虽然您可以在派生类中添加新的功能，但有时您可能想要改变已经在父类中定义的方法的行为。这时您可以保持方法名称不变，重载方法的功能即可，如下面实例所示：

实例

```ruby
#!/usr/bin/ruby -w
# 定义类
class Box
# 构造器方法
def initialize(w,h)
@width, @height = w, h
end
# 实例方法
def getArea
@width * @height
end
end
# 定义子类
class BigBox < Box
# 改变已有的 getArea 方法
def getArea
@area = @width * @height
puts "Big box area is : #@area"
end
end
# 创建对象
box = BigBox.new(10, 20)
# 使用重载的方法输出面积
box.getArea()
```

尝试一下 »

以上实例运行输出结果为：

```
Big box area is : 200
```

运算符重载

我们希望使用 + 运算符执行两个 Box 对象的向量加法，使用 * 运算符来把 Box 的 width 和 height 相乘，使用一元运算符 - 对 Box 的 width 和 height 求反。下面是一个带有数学运算符定义的 Box 类版本：

```ruby
class Box
def initialize(w,h) # 初始化 width 和 height
@width,@height = w, h
end
def +(other) # 定义 + 来执行向量加法
Box.new(@width + other.width, @height + other.height)
end
def -@ # 定义一元运算符 - 来对 width 和 height 求反
Box.new(-@width, -@height)
end
def *(scalar) # 执行标量乘法
Box.new(@width*scalar, @height*scalar)
end
end
```

冻结对象

有时候，我们想要防止对象被改变。在 Object 中，freeze 方法可实现这点，它能有效地把一个对象变成一个常量。任何对象都可以通过调用 **Object.freeze** 进行冻结。冻结对象不能被修改，也就是说，您不能改变它的实例变量。

您可以使用 **Object.frozen?** 方法检查一个给定的对象是否已经被冻结。如果对象已被冻结，该方法将返回 true，否则返回一个 false 值。下面的实例解释了这个概念：

实例

```
#!/usr/bin/ruby -w
# 定义类
class Box
# 构造器方法
def initialize(w,h)
@width, @height = w, h
end
# 访问器方法
def getWidth
@width
end
def getHeight
@height
end
# 设置器方法
def setWidth=(value)
@width = value
end
def setHeight=(value)
@height = value
end
end
# 创建对象
box = Box.new(10, 20)
# 让我们冻结该对象
box.freeze
if( box.frozen? )
puts "Box object is frozen object"
else
puts "Box object is normal object"
end
# 现在尝试使用设置器方法
box.setWidth = 30
box.setHeight = 50
# 使用访问器方法
x = box.getWidth()
y = box.getHeight()
puts "Width of the box is : #{x}"
puts "Height of the box is : #{y}"
```

尝试一下 »

当上面的代码执行时，它会产生以下结果：

```
Box object is frozen object
test.rb:20:in `setWidth=': can't modify frozen object (TypeError)
        from test.rb:39
```

类常量

您可以在类的内部定义一个常量，通过把一个直接的数值或字符串值赋给一个变量来定义的，常量的定义不需要使用 @ 或 @@。按照惯例，常量的名称使用大写。

一旦常量被定义，您就不能改变它的值，您可以在类的内部直接访问常量，就像是访问变量一样，但是如果您想要在类的外部访问常量，那么您必须使用 **classname::constant**，如下面实例所示。

实例

```ruby
#!/usr/bin/ruby -w
# 定义类
class Box
BOX_COMPANY = "TATA Inc"
BOXWEIGHT = 10
# 构造器方法
def initialize(w,h)
@width, @height = w, h
end
# 实例方法
def getArea
@width * @height
end
end
# 创建对象
box = Box.new(10, 20)
# 调用实例方法
a = box.getArea()
puts "Area of the box is : #{a}"
puts Box::BOX_COMPANY
puts "Box weight is: #{Box::BOXWEIGHT}"
```

尝试一下 »

当上面的代码执行时，它会产生以下结果：

```
Area of the box is : 200
TATA Inc
Box weight is: 10
```

类常量可被继承，也可像实例方法一样被重载。

使用 allocate 创建对象

可能有一种情况，您想要在不调用对象构造器 **initialize** 的情况下创建对象，即，使用 new 方法创建对象，在这种情况下，您可以调用 allocate 来创建一个未初始化的对象，如下面实例所示：

实例

```
#!/usr/bin/ruby -w
# 定义类
class Box
attr_accessor :width, :height
# 构造器方法
def initialize(w,h)
@width, @height = w, h
end
# 实例方法
def getArea
@width * @height
end
end
# 使用 new 创建对象
box1 = Box.new(10, 20)
# 使用 allocate 创建两一个对象
box2 = Box.allocate
# 使用 box1 调用实例方法
a = box1.getArea()
puts "Area of the box is : #{a}"
# 使用 box2 调用实例方法
a = box2.getArea()
puts "Area of the box is : #{a}"
```

尝试一下 »

当上面的代码执行时，它会产生以下结果：

```
Area of the box is : 200
test.rb:14: warning: instance variable @width not initialized
test.rb:14: warning: instance variable @height not initialized
test.rb:14:in `getArea': undefined method `*'
   for nil:NilClass (NoMethodError) from test.rb:29
```

类信息

Ruby的 self 和 Java 的 this 有相似之处，但又大不相同。Java的方法都是在实例方法中引用，所以this一般都是指向当前对象的。而Ruby的代码逐行执行，所以在不同的上下文(context)self就有了不同的含义。让我们来看看下面的实例：.

实例

```
#!/usr/bin/ruby -w
class Box
# 输出类信息
puts "Class of self = #{self.class}"
```

```
puts "Name of self = #{self.name}"
end
```

尝试一下 »

当上面的代码执行时，它会产生以下结果：

```
Class of self = Class
Name of self = Box
```

这意味着类定义可通过把该类作为当前对象来执行，同时也意味着元类和父类中的该方法在方法定义执行期间是可用的。

← Ruby 异常 Ruby 正则表达式 →