5 天搞定 Ruby on Rails

企业内训纲要

第二天：Ruby 面向对象编程

版本： V1

口号：快速迭代，不断完善

ruby 技术交流--南方群 95824005

ruby 技术交流--北方群 101388340

ruby 技术交流--东部群 236263776

ruby 技术交流--西部群 230015785

ruby 技术交流--中部群 104131248

文档，实例地址：

https://github.com/nienwoo/ruby-learn

1. 第二天目标

- 掌握 Ruby 语言的类和对象。

2. 面向对象基础

2.1.1. 什么是类，什么是对象

（1）基本概念

对比女娲造人的例子，类是造人的模型，对象是依据模型造出来的人。类是抽象的， 对象是具体的。

类有两大成员构成：数据成员、方法成员。

举个例子：

车辆类的数据成员：轮子，马力，燃油或燃气罐容量。

车辆类的方法成员：如停止，驾驶，超速驾驶。

车辆类可以被定义为：

```
Class Vehicle
{
    Number no_of_wheels
    Number horsepower
    Characters type_of_tank
    Number Capacity
    Function speeding
    {
    }
    Function driving
    {
    }
    Function halting
    {
    }
}
```

通过这些数据成员分配不同的值，可以形成类车辆的几个实例。例如，飞机的有三个轮

子，1,000 马力，不同的燃料罐及容量为 100 升。同样的方式，一辆汽车有四个轮子，200 马力，气体作为不同的罐及容量 25 升。

类的定义

Ruby 中一个类总是以关键字 class 类的名称开头。名称应始终以首字母大写。如以是 Customer 类可以显示为：

```
class Customer
end
```

类定义结束通过使用关键字 end 结束。在类的所有数据成员是类之间的定义，并以 end 关键字作为结束符。

2. 1. 2. Ruby 类中的变量:

Ruby 提供了四种类型的变量：

（1）局部变量:

局部变量是在一个方法中定义的变量。局部变量是不可用的方法外。更多细节在随后的章节中的方法中会介绍。局部变量一般以小写字母或_开头。

实例变量

实例变量是可跨越任何特定实例或对象的方法。这意味着，从对象到对象的实例变量改变。实例变量前面加上 at 符号（@），跟着变量名。

类变量

类变量是可在各种不同的对象。 一个类变量属于类，是类的一个特点。他们前面的符号@@跟着的变量名。

全局变量

类变量是不能跨类。如果想要一个单一的变量可以跨类，需要定义一个全局变量。全局变量的前面总是用美元符号（$）。

例子:

使用类变量@@no_of_customers，能确定创建的对象的数量。这使得导出的客户数量。

```
class Customer
    @@no_of_customers=0
End
```

2.1.3. 对象

（1）创建对象

Ruby 中使用 new 方法创建对象：

对象是类的实例。现在，将学习如何在 Ruby 中创建对象一个类对象。Ruby 中通过使用 new 方法创建对象。

new 方法是一种独特的方法，这是预定义在 Ruby 库。new 方法属于类的方法。

下面的例子是创建两个对象类客户 cust1 和 cust2：

```
cust1 = Customer. new
cust2 = Customer. new
```

在这里，cust1 和 cust2 是两个对象的名字。在等于号（=）之后，类名称将按照对象名称。然后，点运算符和关键字 new 在后面。

自定义方法来创建 Ruby 对象 ：

可以通过 new 方法的参数，这些参数可以用来初始化类变量。

当打算声明的 new 方法具有参数，需要声明的方法在创建类的时候初始化。

initialize 方法是一种特殊类型的方法，该方法时将执行 new 方法的类被称为参数。

下面的例子是创建 initialize 方法：

```
class Customer
    @@no_of_customers=0
    def initialize(id, name, addr)
        @cust_id=id
        @cust_name=name
        @cust_addr=addr
    end
end
```

在这个例子中，可以声明局部变量的初始化方法 id, name 和 addr。这里 def 结束被用来定义一个 Ruby 的方法初始化。这些将在有关后续章节中了解更多。

在 initialize 方法中，对这些局部变量的值传递到实例变量@cust_id,@cust_name 和 @cust_addr。这里的局部变量持有的值由 new 方法一同传递。

现在可以创建对象，如下所示：

```
cust1=Customer.new("1", "John", "Wisdom Apartments, Ludhiya")
cust2=Customer.new("2", "Poul", "New Empire road, Khandala")
```

2.1.4. 方法成员:

方法成员，也叫函数成员或者成员函数。在一个类中的每个方法的方法名用关键字 def 开始。

方法名总是以小写字母最好。你最终的方法 Ruby 中通过使用关键字 end 表示结束。

下面的例子是定义一个 Ruby 的 方法：

```
class Sample
    def function
        statement 1
        statement 2
    end
end
```

这里 statement1 和 statement2 为函数体的一部分。这些 statments 可以是任何有效的 Ruby 语句。例如，我们可以在方法中打印 Hello Ruby 如下：

```
class Sample
    def hello
        puts "Hello Ruby!"
    end
end
```

```
# Now using above class to create objects
object = Sample. new
Object.hello
```

2. 1. 5. 访问器(accessor) & 设置器(setter)方法

为了在类的外部使用变量，我们必须在访问器方法内部定义这些变量，accessor 其实相当于 getter。下面的实例演示了访问器方法的用法：

```
#!/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
box = Box.new(10, 20)

# 使用访问器方法
x = box.printWidth()
y = box.printHeight()

puts "Width of the box is : #{x}"
puts "Height of the box is : #{y}"
```

当上面的代码执行时，它会产生以下结果：

```
Width of the box is : 10
Height of the box is : 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设置变量值的方式，也就是所谓的设置器方法，定义如下：

```
#!/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 "Width of the box is : #{x}"
puts "Height of the box is : #{y}"
```

当上面的代码执行时，它会产生以下结果：

```
Width of the box is : 30

Height of the box is : 50
```

2.1.6. 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：

```
#!/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}"
```

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)
```

2.1.7. 练习

(1) 用默认构建器创建一个类（没有自变量），用它打印一条消息。创建属于这个类的一个对象。

(2) 在练习 1 的基础上增加一个过载的构建器，令其采用一个 String 自变量，并随同自己的消息打印出来。

(3) 以练习 2 创建的类为基础上，创建属于它的对象句柄的一个数组，但不要实际创建对象并分配到数组里。运行程序时，注意是否打印出来自构建器调用的初始化消息。

(4) 创建同句柄数组联系起来的对象，最终完成练习 3。

(5) 用自变量“before”，“after”和“none”运行程序，试验 Garbage.java。重复这个操作，观察是否

从输出中看出了一些固定的模式。改变代码，使 System.runFinalization()在 System.gc()之前调用，再观

察结果。

3. 访问控制

3.1.1. 访问控制的三个级别

Ruby 提供了三个级别的保护实例方法的级别：public, private 和 protected。 Ruby 没有应用实例和类变量的任何访问控制权。

- Public Methods: 任何人都可以被称为 public 方法。方法默认为公用初始化，这始终是 private 除外。 .
- Private Methods: private 方法不能被访问，或者甚至从类的外部浏览。只有类方法可以访问私有成员。
- Protected Methods: 受保护的方法可以被调用，只能通过定义类及其子类的对象。访问保存在类内部。

3.1.2. 示例

以下是一个简单的例子来说明三个访问修饰符的语法：

```
#!/usr/bin/ruby -w

# define a class
```

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

    # instance method by default it is public
    def getArea
        getWidth() * getHeight
    end

    # define private accessor methods
    def getWidth
        @width
    end

    def getHeight
        @height
    end
    # make them private
    private :getWidth, :getHeight

    # instance method to print area
    def printArea
        @area = getWidth() * getHeight
        puts "Big box area is : #@area"
    end
    # make it protected
    protected :printArea
end

# create an object
box = Box.new(10, 20)
```

```
# call instance methods
a = box.getArea()
puts "Area of the box is : #{a}"

# try to call protected or methods
box.printArea()
```

当上面的代码被执行时，产生下面的结果。在这里，第一种方法被调用成功，但第二种方法给一个提示。

```
Area of the box is : 200

test.rb:42: protected method `printArea' called for #

<Box:0xb7f11280 @height=20, @width=10> (NoMethodError)
```

3.1.3. 练习

5.6 练习

(1) 用 public、private、protected 以及“友好的”数据成员及方法成员创建一个类。创建属于这个类的一个对象，并观察在试图访问所有类成员时会获得哪种类型的编译器错误提示。注意同一个目录内的类属于“默认”包的一部分。

(2) 用 protected 数据创建一个类。在相同的文件里创建第二个类，用一个方法操纵第一个类里的 protected 数据。

(3) 新建一个目录，并编辑自己的 CLASSPATH，以便包括那个新目录。将 P.class 文

件复制到自己的新目录，然后改变文件名、P 类以及方法名（亦可考虑添加额外的输出，观察它的运行过程）。在一个不同的目录里创建另一个程序，令其使用自己的新类。

(4) 在 c05 目录（假定在自己的 CLASSPATH 里）创建下述文件：

```
//: PackagedClass.java
package c05;
class PackagedClass {
   public PackagedClass() {
      System.out.println(
         "Creating a packaged class");
   }
} ///:~
```

然后在 c05 之外的另一个目录里创建下述文件：

```
//: Foreign.java
package c05.foreign;
import c05.*;
public class Foreign {
    public static void main (String[] args) {
        PackagedClass pc = new PackagedClass();
    }


} ///:~
```

解释编译器为什么会产生一个错误。将 Foreign（外部）类作为 c05 包的一部分改变了什么东西吗？

4. 类的继承

在面向对象的编程中最重要的概念之一是继承。继承允许我们定义一个类在另一个类的项目，这使得它更容易创建和维护应用程序。

继承也提供了一个机会，重用代码的功能和快速的实现时间，但不幸的是 Ruby 不支持多级的继承，但 Ruby 支持混入。一个 mixin 继承多重继承，只有接口部分像一个专门的实现。

当创建一个类，而不是写入新的数据成员和成员函数，程序员可以指定新的类继承现有类的成员。这种现有的类称为基类或父类和新类称为派生类或子类。

Ruby 也支持继承。继承和下面的例子解释了这个概念。扩展类的语法很简单。只需添加一个<字符的超类声明的名称。

4. 1. 1. 继承案例

例如，定义 Box 类的子类 classBigBox：

```
#!/usr/bin/ruby -w

# define a class
class Box
    # constructor method
    def initialize(w,h)
        @width, @height = w, h
```

```
    end
    # instance method
    def getArea
        @width * @height
    end
end

# define a subclass
class BigBox < Box

    # add a new instance method
    def printArea
        @area = @width * @height
        puts "Big box area is : #@area"
    end
end

# create an object
box = BigBox.new(10, 20)

# print the area
box.printArea()
```

当上面的代码执行时，它会产生以下结果：

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

4.1.2. 方法重载：

虽然可以在派生类中添加新的函数，但有时想改变的行为已经在父类中定义的方法。只

需通过保持相同的方法名和重写该方法的功能，如下图所示，在这个例子可以这样做：

```
#!/usr/bin/ruby -w

# define a class
class Box
   # constructor method
   def initialize(w,h)
      @width, @height = w, h
   end
   # instance method
   def getArea
      @width * @height
   end
end

# define a subclass
class BigBox < Box

   # change existing getArea method as follows
   def getArea
      @area = @width * @height
      puts "Big box area is : #@area"
   end
end

# create an object
box = BigBox.new(10, 20)

# print the area using overriden method.
box.getArea()
```

运算符重载：

我们想“+”运算符使用+，*操作由一个标量乘以一箱的宽度和高度，这里是一个版 Box 类的定义及数学运算符：

```
class Box
   def initialize(w,h) # Initialize the width and height
      @width,@height = w, h
   end

   def +(other)              # Define + to do vector addition
      Box.new(@width + other.width, @height + other.height)
   end

   def -@                    # Define unary minus to negate width and height
      Box.new(-@width, -@height)
   end

   def *(scalar)             # To perform scalar multiplication
      Box.new(@width*scalar, @height*scalar)
   end
end
```

4.1.3. 冻结对象：

有时候，我们要防止被改变的对象。冻结对象的方法可以让我们做到这一点，有效地把一个对象到一个恒定。任何对象都可以被冻结通过调用 Object.freeze。不得修改冻结对象：不能改变它的实例变量。

可以使用 Object.frozen？语句检查一个给定的对象是否已经被冻结，被冻结的情况下的

对象语句方法返回 true，否则返回 false 值。下面的示例 freeze 的概念：

```
#!/usr/bin/ruby -w

# define a class
class Box
    # constructor method
    def initialize(w,h)
        @width, @height = w, h
    end

    # accessor methods
    def getWidth
        @width
    end
    def getHeight
        @height
    end

    # setter methods
    def setWidth=(value)
        @width = value
    end
    def setHeight=(value)
        @height = value
    end
end

# create an object
box = Box.new(10, 20)

# let us freez this object
box.freeze
if( box.frozen? )
```

```
    puts "Box object is frozen object"
else
    puts "Box object is normal object"
end

# now try using setter methods
box.setWidth = 30
box.setHeight = 50

# use accessor methods
x = box.getWidth()
y = box.getHeight()

puts "Width of the box is : #{x}"
puts "Height of the box is : #{y}"
```

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
```

4.1.4. 类常量：

可以在类里定义分配一个直接的数字或字符串值，而不使用其定义一个变量为@@ 或@。按照规范，我们保持常量名大写。

一个常量一旦被定义就不能改变它的值，但可以在类里像常量一样直接访问，但如果要访问一个类之外的常量，那么要使用类名::常量，所示在下面的例子。

```
#!/usr/bin/ruby -w

# define a class
class Box
   BOX_COMPANY = "TATA Inc"
   BOXWEIGHT = 10
   # constructor method
   def initialize(w,h)
      @width, @height = w, h
   end
   # instance method
   def getArea
      @width * @height
   end
end

# create an object
box = Box.new(10, 20)

# call instance methods
a = box.getArea()
puts "Area of the box is : #{a}"
puts Box::BOX_COMPANY
puts "Box weight is: #{Box::BOXWEIGHT}"
```

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
```

类常量继承和实例方法一样，可以覆盖。

4.1.5. 创建对象使用分配：

当创建一个对象，而不调用它的构造函数初始化，即可能有一个情况：采用 new 方法，在这种情况下可以调用分配，这将创造一个未初始化的对象，看下面的例子：

```
#!/usr/bin/ruby -w

# define a class
class Box
    attr_accessor :width, :height

    # constructor method
    def initialize(w,h)
        @width, @height = w, h
    end

    # instance method
    def getArea
        @width * @height
    end
end

# create an object using new
box1 = Box.new(10, 20)

# create another object using allocate
box2 = Box.allocate

# call instance method using box1
a = box1.getArea()
puts "Area of the box is : #{a}"
```

```
# call instance method using box2
a = box2.getArea()
puts "Area of the box is : #{a}"
```

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
```

4.1.6. 类信息：

如果类定义的可执行代码，这意味着他们在执行的上下文中一些对象：自身都必须引用的东西。让我们来看看它是什么。