Ruby 基础

全栈体系课：Rails 入门

版权声明

联系方式

如果你想要购买本课程
请微信联系 xiedaimala02 或 xiedaimala03

如果你发现有人盗用本课程
请微信联系 xiedaimala02 或 xiedaimala03

我默认
你已经有一点点编程基础

JS / Java / PHP / Python / C 无论什么基础都行

目录

- Ruby 的基本概念
- Ruby 的控制语句
- Ruby 的循环
- Ruby 的数组与散列

运行 ruby 的方式

- irb
  - 可交互命令行
  - 你每次输入一个式子，它都会用 => 给出一个值
- ruby
  - ruby 加文件路径

变量

- 没有关键字
  - 没有 var、let、const，直接 a = 1 就行
- 局部变量
  - 小写字母开头
  - _ 开头
- 全局变量
  - $ 开头
- 类变量
  - @@ 开头
- 实例变量
  - @ 开头

ruby 默认没有闭包

- 代码

```
a = 1
def f1
  p a
end
f1() // 括号可以省略
// 报错：a 不存在
```

- 默认没有闭包
  - ✓ 闭包是指函数可以使用外部的自由变量
  - ✓ 没有闭包是指函数不可以使用外部的自由变量
  - ✓ 用Ruby其他语法可以实现闭包，先不讲

全局变量

- 代码

```
$a = 1
def f1
  p $a
end
f1()
// 1
```

- 加上 $ 就是全局变量

常量

全大写：RUBY_VERSION / ARGV

常量

- 代码

```
A = 1
def f1
  p A
end
f1
// 1
```

- 大写就是常量

多重赋值

- 简化
  - a, b, c = 1, 2, 3
  - a, b, *c=1, 2, 3, 4, 5
  - a, *b, c=1, 2, 3, 4, 5
- 交换
  - a, b = b, a
- 数组
  - arr = [1, 2]
  - a, b = arr

字符串

- 单引号
  - puts '12\n34'
  - 不会转义
- 双引号
  - puts "12\n34"
  - 会把 \n 转义成换行
- 多行字符串
  - puts <<xxx
  - 12
  - 34
  - xxx
  - 如果有 \n，会转义成换行

log

- 打印数据
  - print('a', 'b') - 默认不加换行
  - puts('a', 'b') - 默认每个加换行
  - p 'a','b' - 加换行，不转义，且以人类理解的形式加标记
  - 只要没有歧义，括号()都可以不写
- 使用场景
  - 程序员一般使用 p
  - 日志里一般使用 puts 和 print

注释

- 单行注释

```
# 这是单行注释
# 这是第二行注释
```

- 多行注释

```
=begin
多行注释
=end
```

控制语句

if / case

ruby 的 if 语句

```
if a>3 then p '大' end

if a>3 then p '大' else p '小' end

p(if a>3 then '大' else '小' end)

p(if a>4
  '大'
elsif a>2
  '中'
else
  '小'
end)
```

ruby 的 if 语句2

```
b = if a>3 then 'big' else 'small' end
b = a>3 ? 'big' : 'small'

return if error
return unless success
```

case 语句

```
z = case x
    when 1
       '1'
    when 2,3
       '2 or 3'
    else
       'hi'
    end
```

循环

- .times 方法
- for 关键字
- while 关键字
- until 关键字
- .each方法
- loop 关键字

7.times

- 一夜七次

```
7.times do
  p '一次'
end

7.times { p '一次'}
```

- 相当于 JS 的

```
[0,1,2,3,4,5,6].map((i)=>{
   p '一次'
})
```

- 块
  - .times 是个函数
  - do ... end 是代码块
  - { ... } 也是代码块
  - 一般多行用 do end 单行用 {...}
  - 代码块作为 .times 函数的参数
  - Ruby 的函数调用可以省略 ()

获取 i

- |i|

```
7.times do |i|
  p "#{i+1}次"
end

7.times { |i|
  p "#{i+1}次"
}
```

- 注意
  - ✓ 右边两个代码都可以写成一行

.each

- 代码

```
names = ['Frank','Jack']
names.each do |name|
  p name
end
names.each {  |name| p name   }

(1..7).each do |i|
  p i
end
(1..7).each {|i| p i}
```

for

- 代码

```
# 1 到 5
for i in 1..5 #do可省略
  p i
end
# 遍历数组
names = ['Frank','Jack']
for name in names
  p name
end
```

while / until

- 代码

```
i = 1
while i<3
  p i
  i+=1
end

j = 1
until j>=3
  p j
  j+=1
end
```

loop

- 代码（不要运行）

```
loop do
  p 'ruby'
end
# 上面代码会一直无限循环，除非在里面写 break
```

跳出循环

break 表示退出所有循环

next 表示退出当前循环，进入下一次循环

对应 JS 里的 break 和 continue

Ruby 中的数据类型

- 只有对象，可用 .class 查看类
  - ✓ 整数 - Integer 对象 / Numeric 对象
  - ✓ 浮点数 - Float 对象
  - ✓ 字符串 - String 对象
  - ✓ 数组 - Array 对象
  - ✓ 正则表达式 - Regexp 对象
  - ✓ 时间 - Time 对象
  - ✓ 文件 - File 对象
  - ✓ 符号 - Symbol 对象
  - ✓ 异常 - Exception 对象
  - ✓ 散列 - Hash 对象
- 标识
  - ✓ 每个对象都有一个 object_id 属性，表示其唯一性

数组

由 Array 类构造出来的对象

跟 JS 的数组差不多

- Ruby 数组

- mixed = [1, '中', 2, '国', 3]
- mixed.size
- => 5
- mixed.methods
- => 打印出 mixed 的所有方法，如 pop / shift / push / append / find / find_all / each / each_with_index ... 注意这些方法前面都有一个冒号，这是 symbol，后面讲

散列

由 Hash 构造出来的对象

Ruby 散列

- 创建一个散列
  - person = {name: 'frank', age: 18}
- symbol
  - 上面的 name 和 age 不是字符串，而是 symbol
  - 等价于 person = { :name => 'frank', :age => 18 }
  - person.keys
  - => [:name, :age]
  - 如果你想要字符串作为 key，应该写成
  - person = { 'name' => 'frank', 'age' => 18}
- 注意
  - 你可以认为 symbol 是轻量的字符串，功能更少
  - :name.to_s 得到字符串，'name'.to_sym 得到符号

遍历散列

- .each 方法

```
person.each do |key, value|
  p "key: #{key}, value: #{value}"
end
# 输出时，symbol 会自动变为 string
```

与 JS 的区别1

person = {name:'frank', age:18}
如果你想要获取 'frank

- 不能用 person.name
  - ✓ 必须用 person[:name]
- 不能用 person['name']
  - ✓ 必须用 person[:name]
- 'name' 和 :name 是不同的
  - ✓ person['name'] = 'jack'
  - ✓ person.keys # [:name, :age, 'name']

与 JS 的区别2

```
def say_hi
  p 'hi'
end
person = {name:'frank', age:18}
```

如果你想要给 person 加一个方法/函数

- 不能用 person[:say_hi] = say_hi
  - 因为 say_hi 等价于 say_hi()
  - say_hi 的返回值为 p 'hi' 的返回值，也就是 'hi'
  - 所以相当于 person[:say_hi] = 'hi'
- 可以使用 lambda 表达式做到
  - person[:say_hi] = lambda { p 'hi'}
  - person[:say_hi].call() #括号可以省略

基础知识就学到这里

下节课做些小 demo 来理解一下这些知识

推荐一些学习资源

- 菜鸟教程
  - 过于简单，适合新手
- API 手册（字典）
  - devdocs.io
  - apidock.com/ruby
- 书籍（目前没必要买）
  - 《Ruby 基础教程》很基础，看起来很慢，这节课程包含了它 1/5 的内容
  - 《Ruby 元编程》高级教程，等你熟悉 Rails 后可以考虑购买