Ruby基础培训

技术管理部 张哲

目标

- 掌握ruby基础知识
- 掌握rails基础知识
- 可以独立开发简单rails应用
- 培养良好编码习惯

课程大纲

- Ruby基础和环境安装
- Ruby基础
- Rails基础
- 习题地址： http://grandsoft.github.io

第一讲

- Ruby简介
- Ruby环境搭建
- Ruby基础

Ruby介绍

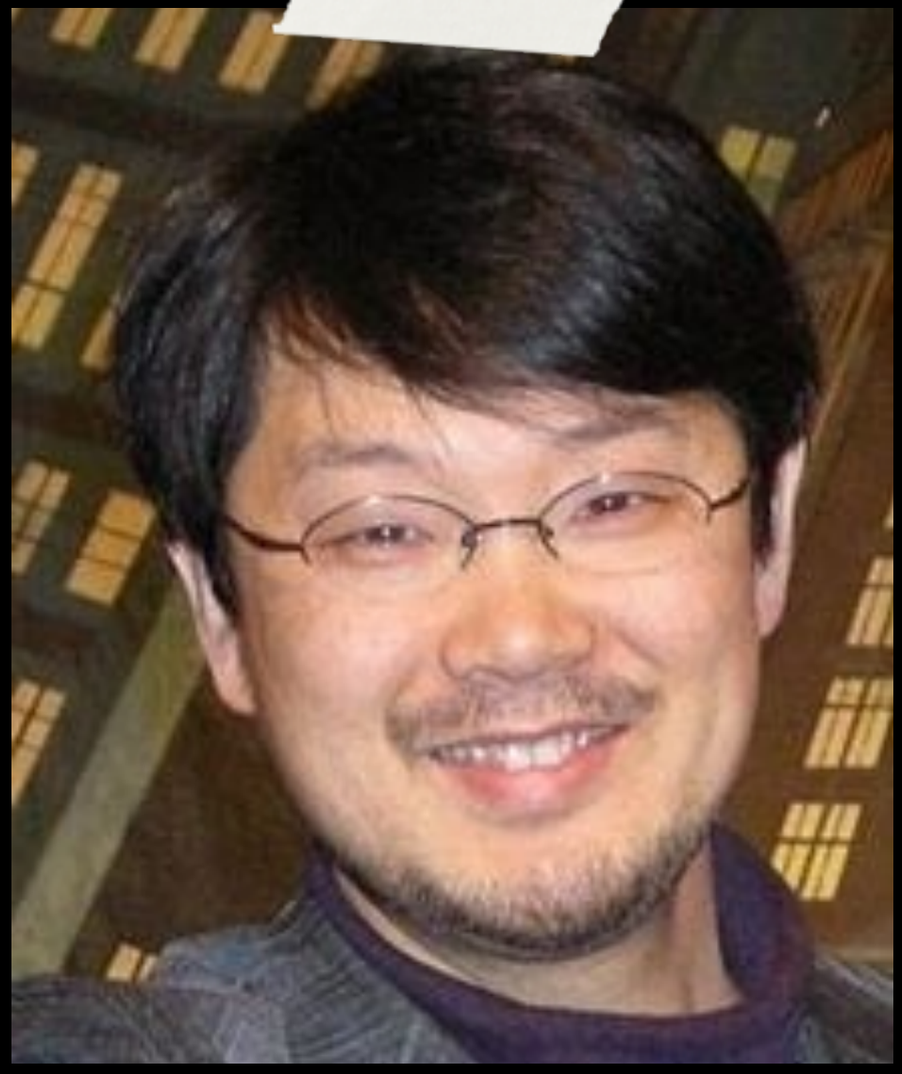
- Ruby作者
- Ruby历程
- Ruby特点
- 协议与拓展
- 应用领域

Ruby作者

- 松本行弘 Matz
- 传教士
- 语言宅男
- 强调编程语言应该不单给程序员带来工资,也要给他们带来快乐。
- Heroku首席架构师

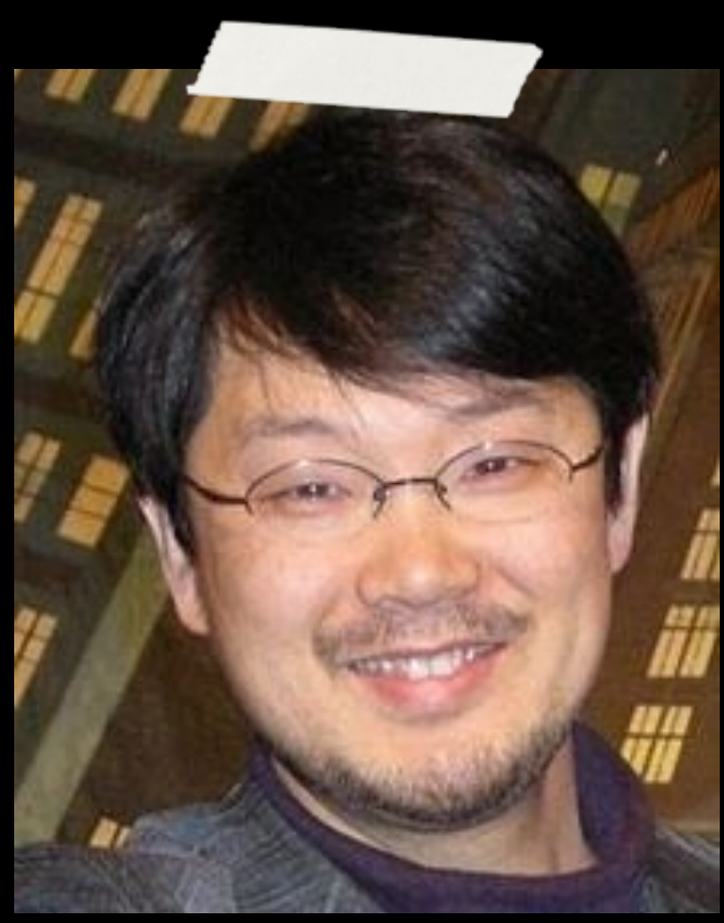
Ruby作者

- 松本行弘 Matz
- 传教士
- 语言宅男
- 强调编程语言应该不单给程序员带来工资,也要给他们带来快乐。
- Heroku首席架构师



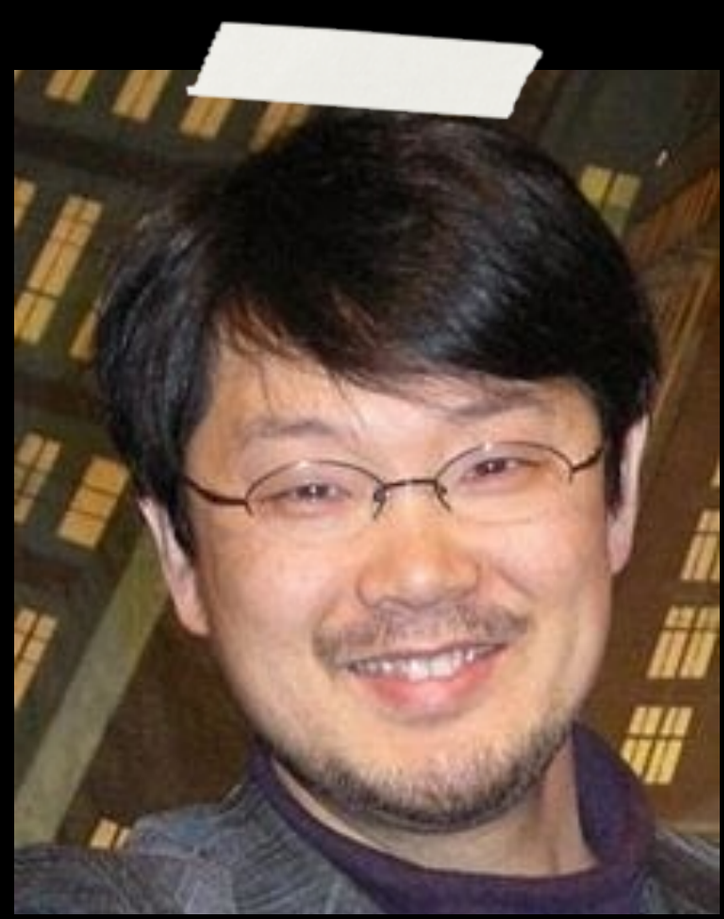
Ruby作者

- 松本行弘 Matz
- 传教士
- 语言宅男
- 强调编程语言应该不单给程序员带来工资,也要给他们带来快乐。
- Heroku首席架构师



Ruby作者

- 松本行弘认为以往人们在开发编程语言时过于看重"计算机"而忽视"人"。
- 他提出机器不是主人，真正的主人应该是"人"。
- 于是他开发出一种更人性化的编程语言，Ruby。



Rails作者

- 赛车手
- 煽动者
- 两本书
- 幽默 twitter \ group on

Ruby On Rails

历程

- Ruby于1993年诞生，在2004年Rails发布之后开始走红，虽然有twitter（后迁移到Java平台），groupon，hulu，github等硅谷新贵公司大量使用Ruby，但在企业级市场，Ruby从未得到过一家重量级厂商的支持。Salesforce的支持终于让Ruby得以登堂入室，成为云计算时代的标准语言之一。
- Ruby on Rails的出现，在Web开发领域掀起了一阵敏捷开发的风暴。在人们惊讶于Ruby on Rails简洁高效的同时，Ruby也迅速被大家所认识，一举成为了最受欢迎的十大程序设计语言之一。

协议与扩展

- Ruby遵守GPL(GNU通用公共许可证)协议和Ruby License。
- Ruby的灵感与特性来自于 Perl、Smalltalk、Eiffel、Ada 以及 Lisp 语言。
- 由 Ruby 语言本身还发展出了JRuby（Java 平台）、IronRuby（.NET 平台）、Mac Ruby（Mac OS X）、RubyMotion（IOS）等其他平台的 Ruby 语言替代品。

应用领域

- 网络开发
- 测试脚本
- 系统管理
- 创建领域特定语言

环境搭建

- 安装Ruby (1.9.3)
  - http://www.ruby-lang.org/
  - rvm
- 安装文档
  - http://railsapi.com/
- 安装开发工具
  - textmate(mac)
  - gedit(mac linux window)
  - vim(mac linux window)
  - emacs(mac linux window)
  - Sublime Text(mac linux window)

Ruby基础

- 关键字
- 命名
- 注释
- 变量、常量和基本类型
- 运算符
- 条件控制
- 异常

一些特征

- 解释型语言
- 面向对象
  - 操作的东西都是对象，操作的结果也是对象
  - (-1).abs VS Math.abs(-1)
- 面向行

```
def hello
  p "hello world!"
end

def hello; p "hello world!"; end
```

关键字

- 模块定义：module
- 类定义：　class
- 方法定义：def，undef
- 检查类型：defined?
- 条件语句：if，then，else，elsif，case，when，unless
- 循环语句：for，in，while，until，next，break，do，redo，retry，yield
- 逻辑判断：not，and，or
- 逻辑值：　true，false
- 空值：　　nil
- 异常处理：rescue，ensure
- 对象引用：super，self
- 块的起始：begin，end
- 嵌入模块：BEGIN，END
- 文件相关：__FILE__，__LINE__
- 方法返回：return
- 别名：　　alias

命名惯例

- Camel命名法：每个单词的首字母都大写，单词间没有间隔，如：BackColor。
- snake命名法：在小写单词的中间用下划线作为分割，如：to_s。
- SCREAMING_SNAKE_CASE命名法：在大写单词的中间用下划线作为分割，如：USER_AGE。

命名惯例

- 方法名使用snake_case命名法，如to_s。
- 常量名使用SCREAMING_SNAKE_CASE命名法，如USER_NAME。
- 变量使用snake_case命名法，如 user_name。
- 实例变量使用@+snake_case命名法，如@user。
- 类变量使用@@+snake_case命名法，如@@user。
- 全局变量使用$+snake_case命名法，如$user（尽量不要使用全局变量）。
- 类和模块名使用Camel命名法，如User。

注释

- 单行注释： #
- 多行注释： =begin ... =end(须顶格写)

```
=begin
def hello
  p "hello world!"
end
=end

# def hello; p "hello world!"; end
```

常量、变量和类型

- 变量没有类型，其引用的对象有类型
- 最简单的数据类型是数值和字符串(String)

```
a = 3
b = 10
puts "#{a} times #{b} equals #{a*b}"
puts '#{a} times #{b} equals #{a*b}'
```

- 反引号字符串

```
p `whoami`
p `ls -l`
%x(say #{`whoami`})
```

常量、变量和类型

- 正则表达式(Regexp) `/^([^_^0-9]*)[\_0-9]/`
- 数组(Array)

```
[1, 2, 3]
[1, 2, "strong"]
a = [1, 2, ["strong", "dog"]]
["dog", "cat", "car"]
%w(dog cat car)
a = [1, 2, ["strong", "dog"]]
p a[2]
```

- 哈希(Hash)

```
{1 => 1, 2 => 2, 3 => 3}
h = {"dog" => "dogs", "cat" => "cats"}
{[1, 2] => "ss", 10 => h}
p h["dog"]
```

运算符

- ::
- []
- **
- +-!
- */%
- &|
- =
- == !=
- ...
- 多功能： +

示例

```
p "Please enter a temperature and scale(C or F)"
str = gets
exit if str.nil? or str.empty?
str.chomp!
temp, scale = str.split(" ")

abort "#{temp} is not a valid number." if temp !~ /-?\d+/

temp = temp.to_f
case scale
when "C", "c"
  f = 1.8*temp + 32
when "F", "f"
  c = (5.0/9.0)*(temp-32)
else
  abort "Must specify C or F."
end

if f.nil?
  p "#{c} degrees C."
else
  p "#{f} degrees F."
end
```

条件语句

```
# if 形式
a = 1 if y == 3
x = if a > 0 then 2 else 3 end

if x < 5 then
  a = 1
else
 a = 2
end
```

```
# unless 形式
a = 1 unless y != 3
x = unless a <= 0 then 2 else 3 end

unless x < 5 then
  a =2
else
  a =1
end
```

循环语句

```ruby
#while循环
i = 0
while i < list.size do
  print "#{list[i]}"
  i += 1
end
```

```ruby
# loop循环
i = 0
n = list.size-1
loop do
  print "#{list[i]}"
  i += 1
  break id i > n
end
```

```ruby
#until循环
i = 0
until i == list.size do
  print "#{list[i]}"
  i += 1
end
```

```ruby
# upto循环
n = list.size - 1
0.upto(n) do |i|
  print "#{list[i]}"
end
```

```ruby
# times循环
n = list.size
n.times do |i|
  print "#{list[i]}"
end
```

```ruby
# for循环
for x in list do
  print "#{x}"
end
```

```ruby
# each循环
list.each do |x|
  print "#{x}"
end
```

循环语句

- 关键字
  - break
  - redo
  - retry
  - next
- 修饰符形式

```
perform_task until finished
perform_task while not finished
```

case语句

- 语法

```
case expression
  when value
    some_action
end
```

- 关系运算符 ===

```
4.times do |i|
  a = "s"*i
  case a
  when "sss"
    p "a is sss"
  when /^ss/
    p "a is ss"
  when "a" .. "z"
    p "a is in (a .. z)"
  when String
    p "a is a String"
  end
end
```

```
p String === "sss"
# >> true
p "sss" === String
# >> false
p /^ss/ === "ss"
# >> true
p "ss" === /^ss/
# >> false
p ("a".."z") === "s"
# >> true
p "s" === ("a".."z")
# >> false
```

异常

- 引发异常

```
raise
raise "Some error"
raise ArgumentError
raise ArgumentError, "Bad data"
raise ArgumentError, new("Bad data")
```

- 处理异常

```
begin
  p aaa
rescue => err
  p "in rescue, error is: #{err}"
  #retry
ensure
  p "must be ensure!"
end
```

作业题

- http://code.bitaec.com

推荐书籍

- Programming Ruby
- Getting real
- Rework
- 广联达Ruby编程规范

第二讲 Ruby中的面向对象

对象

- 数字、字符串、数组、哈希、正则表达式等等 ... 都是对象

```
p 10.class
# >> Fixnum
p "sss".class
# >> String
p ["a", "b", "c"].class
# >> Array
p ("a" .. "c").class
# >> Range
a = {1 => 2, 3 => 4}
p a.class
# >> Hash
p /\d+/.class
# >> Regexp
```

内置类

- 30多个内置类： Array, File, Fixnum, Hash, IO, String, Struct, Thread ...
- 创建对象：
  - new

```
s = String.new("sss")
```

  - 字面量

```
s = "sss"
```

- 变量本身没有类型，只是对对象的引用(小型对象直接复制)

```
x = "abc"
y = x
x.gsub!(/a/, "x")
p y
# >> "xbc"
```

```ruby
class Greeter
  NAME = "GREETER"
  @@count = 0
  def initialize(name="nobody", from="world")
    @name = name; @from = from
  end

  def say_hi
    p "Hi, #{@name} from #{@from}"
    @@count += 1
  end

  def say_bye
    p "Bye, #{@name} from #{@from}"
    @@count += 1
  end

  def self.count; @@count; end
end

greeter = Greeter.new("Gary")
greeter.say_hi # >> "Hi, Gary from world"
greeter.say_bye # >> "Bye, Gary from world"
p greeter.count # >> 2
p Greeter::NAME # >> "GREETER"
```

类

- 类名本身是全局变量
- 类定义可以包括：
  - 类常量、类变量（类实例变量）、类方法
  - 实例变量、实例方法
- 类数据可供类的所有对象使用，实例数据只供一个对象使用

attr_accessor

```
class Customer
  def initialize
    @name = "nobody"
    @gender = "male"
  end

  def name
    @name
  end

  def name=(n)
    @name = n
  end

  def gender
    @gender
  end

  def gender=(g)
    @gender = g
  end
end
```

attr_accessor

```ruby
class Customer
  def initialize
    @name = "nobody"
    @gender = "male"
  end

  def name
    @name
  end

  def name=(n)
    @name = n
  end

  def gender
    @gender
  end

  def gender=(g)
    @gender = g
  end
end
```

```ruby
class Customer
  attr_accessor :name, :gender
  def initialize
    @name = "nobody"
    @gender = "male"
  end
end
```

访问控制

- private(private to this instance): 只能在当前类和子类的上下文中调用，不能指定方法接收者（即：接收者只能是self，且self必须省略。）
- protect: 只能在当前类和子类的上下文中调用
- public: 没有限制

访问控制

```
class A
  def test
    protected_mth
    private_mth
    #self.private_mth     #wrong
    self.protected_mth
    obj = B.new
    #obj.private_mth      #wrong
    obj.protected_mth
  end

  protected
  def protected_mth
    puts "#{self.class}-protected"
  end

  private
  def private_mth
    puts "#{self.class}-private"
  end
end
```

```
class B < A
  def test
    protected_mth
    private_mth

    #self.private_mth     #wrong
    self.protected_mth

    obj = B.new
    #obj.private_mth      #wrong
    obj.protected_mth
  end
end
```

```
class C
  def test
    a = A.new
    #a.private_mth          #wrong
    #a.protected_mth        #wrong
  end
end
```

方法定义

- 使用 def 关键字
- 一般参数： param `def test_find(data, string); end`
- 默认参数： param=value `def test (name, age=20); end`
- 长度可变的参数：*param
- Proc化后接收的参数： &param

方法定义

- 实例方法 – 针对实例
- 类方法 – 针对类
  - mbp
  - 具备名称和价格属性
  - 降价
  - 可以统一降价
  - 找出最贵的一款

实例练习

- 员工管理
  - 可记录员工总数
  - 所在部门统一为：技术管理部（TMD）
  - 员工个人信息包括：姓名、年龄、性别
  - 可新建员工
  - 可修改员工年龄
  - 可找出年龄最小的员工
  - 可找出所有女员工
  - 可返回员工所在部门
  - 可计算员工总数

方法定义

- 实例方法 – 针对实例
- 类方法 – 针对类
  - mbp
  - 具备名称和价格属性
  - 降价
  - 可以统一降价
  - 找出最贵的一款

实例练习

- 员工管理
  - 可记录员工总数
  - 所在部门统一为：技术管理部（TMD）
  - 员工个人信息包括：姓名、年龄、性别
  - 可新建员工
  - 可修改员工年龄
  - 可找出年龄最小的员工
  - 可找出所有女员工
  - 可返回员工所在部门
  - 可计算员工总数

类继承

- 普通继承 – Ruby只支持单继承

```
class Child < Father
end
```

  - Object是所有类的始祖(ancestors, superclass)
  - super 调用其父类中同名的方法。
  - 向一个对象发送调用方法时，ruby的本质过程如下：
    - ruby将检查该对象所属的类，如果该类实现了该方法，就运行该方法
    - 如果该类没有实现该方法，ruby将找到其直接父类中的方法，然后是其祖父类，一直追溯整个祖先链。
- 例子:
  - Human(name, age, job#work, speak, run)
  - Dog(name, age#bark run)

文艺继承 — 多重继承

模块 Module

- Mixin - include
- 当类包含模块后，模块的所有的实例方法和常量在该类中就可以使用了。
  - 类继承自模块。模块不能被实例化，不能被继承。
  - 如果模块和类不在同一个文件中，要使用include，则先使用require，把文件引入。（require VS load）
  - 当一个类包含一个模块时，Ruby创建了一个指向该模块的匿名代理类，并将这个匿名代理插入到当前类作为其直接父类。
  - 如果有多个include，将依次生成代理类，最后一个include的将是该类的直接父类。
- 例子： Programmer(name, age, job, salary, department#raise_salary, get_fired)
- 如果在类中使用extend，模块的方法会变为类方法。

内省

```
p obj.class
p obj.class.instance_methods false
p obj.instance_variables
```

模块

- 命名空间

```
module Grandsoft
  module Tech
    module Cmd
      NAME = "Technical Management Department"
      def test
        p "Test in #{NAME}"
      end
    end
  end
end

p Grandsoft::Tech::Cmd::NAME
# >> "Technical Management Department"
include Grandsoft::Tech::Cmd
test
# >> "Test in Technical Management Department"
```

第三讲 Ruby基本类型

数字

- Ruby中数字分为整数（Integer）、浮点数（Float）和复数（Complex）这3种。
- 整数类又分为定整数（Fixnum）和大整数（Bignum）。

数字常用方法

- +,-,*,/是最基本的操作符
- 两个整数相除，得到的结果是一个整数（即相除结果的整数部分）。如果想得到的结果是一个浮点数，需要做除法的两个数中至少有一个是浮点数。

数字常用方法

- %：取模，即余数。
- to_f：返回整数的浮点值，其实就是加一个".0"。
- to_s：返回整数的字符串。
- downto(integer)：接收一个block，进行从大到小的循环执行block。
- upto(integer)：类似downto。
- next：返回下一个数。
- step(end, step)：从当前数循环到end，步长为step。
- times：循环当前数字次。
- ceil：返回比当前数大的最小的整数。
- floor：返回比当前数小的最大整数。
- to_i：返回当前数截掉小数点后面数字的整数。

字符串定义

- 字符串都是String的对象： String.new(“ssss”)
- 使用单引号或双引号定义
  - str = “1234567#{1+1}”
  - str = ‘abcdefg#{1+1}’
- %q或%Q字符串表示法。
  - str = %q[abcdefg#{1+1}]
  - str = %Q[1234567#{1+1}]

字符串定义

- here document

```
string = <<END
如果有多个include，将依次生成代理类，最后一个include的将是该类的直接父类。
例子：Programmer(name, age, job, salary, department#raise_salary, get_fired)

p Dog.new("fangcai", 10)
p Human.new("BOB", 20, "programmer")
END
```

字符串常用方法

- str.length(str.size)：返回字符串的长度。
- str +(str <<)：追加字符串。
- str.capitalize：将字符串首字符大写，其余字符小写。
- str.delete(string)：返回删除string后的字符串。
- str.delete!(string)：将delete的返回结果直接作用于当前字符串。
- str.strip：返回去除字符串收尾的非可见字符的字符串。
- str.strip!：将strip的返回结果直接作用于当前字符串。

字符串常用方法

- str.downcase(str.upcase)：字符串转换为小(大)写。
- str.include?(string)：如果字符串包含string则返回true，否则返回false。
- str.index(string)：返回string在字符串中首次出现的位置。rindex方法则返回从后开始检索，字符串string首次出现的位置。
- str.reverse：返回字符串的顺序反转字符串。
- str.split(patten)：基于分隔符patten将字符串str分割成若干子字符串，并以数组形式返回。
- str.scan(pattern)：搜索出str中所有匹配pattern的子字符串，并以数组形式返回。pattern可以是字符串，也可以是正则表达式。

字符串实例

- 地区分类

数组和哈希

- 简介
- 数组
- 哈希

简介

- Ruby的数组和哈希是索引集合。
- 两者都是保存对象集合并能通过键来读取。
- 数组的键是数字，哈希则支持对象作为键。
- 数组和哈希都能保存不同类型的对象。
- 数组有序，哈希无序。
- 在访问元素方面，数组效率比较高，但哈希表更加的灵活。
- 数组（哈希）不需要声明，也不需要告诉解释器里面将容纳什么类型的对象或是数组需要容纳多少元素，Ruby中数组（哈希）的大小是随时可以改变的。

数组定义

- Array.new
- [“one”, “two”, “three”, “four”]
- %w(one two three four)

数组常用方法

- arr.insert(pos, elem) 插入
- arr.delete(elem) arr.delete_at(pos) 删除
- arr[pos]= new_elem 修改
- arr.push 压入
- arr.pop 弹出
- arr.shift 入队
- arr.unshift 出队
- arr1 << elem 相加
- arr1 &(| -) arr2
- arr.compact 去除所有nil
- arr.uniq 压缩所有重复元素
- arr.sort 排序
- arr.reverse 反序
- arr.flatten
- arr.transpose
- arr.clear
- arr.empty?

例子

- 实现：
  - 1,2,3,4,5,6,7 => Mon-Sun
  - 1,2,3,6,7 => Mon-Wed, Sat, Sun
  - 1,3,4,5,6 => Mon, Wed-Sat
  - 2,3,4,6,7 => Tue-Thu, Sat, Sun
  - 1,3,4,6,7 => Mon, Wed, Thu, Sat, Sun
  - 7 => Sun
  - 1,7 => Mon, Sun
  - 1,8 => ArgumentError

正则表达式

- 简介
- 定义
- 例子

简介

- 正则表达式主要用于描述字符串规则。
- 例如获取网页代码中的所有超级链接地址或邮件地址，如果仅使用字符串的检索功能，实现起来会非常繁琐，而如果使用正则表达式进行匹配查找就会非常简单了。
- 正则表达式由一些普通字符和一些含有特殊含义的标记符号组成。普通字符包括大小写的字母、数字和一些普通符号，而标记符号则用于描述特殊的字符规则。
- 例如：\d{4}-\d{2}-\d{2} 就可以匹配类似于“2008-04-01”这样的日期字符串。

定义

- 定义正则表达式的3种方法：
  - 使用Regexp类： r1 = Regexp.new(‘[Rr]uby’)
  - 使用/pattern/： r2 = /[Rr]uby/
  - 使用%r{pattern}： r3 = %r{[Rr]uby}

规则 - 字符类缩写

Sequence	As [. . .]	Meaning (Unicode)
\d	[0-9]	Decimal digit character (*Decimal_Number*)
\D	[^0-9]	Any character except a digit
\h	[0-9a-fA-F]	Hexadecimal digit character
\H	[^0-9a-fA-F]	Any character except a hex digit
\s	[␣\t\r\n\f]	Whitespace character (+ *Line_Separator*)
\S	[^␣\t\r\n\f]	Any character except whitespace
\w	[A-Za-z0-9_]	Word character (+ *Connector_Punctuation*, *Letter*, *Mark*, and *Number*)
\W	[^A-Za-z0-9_]	Any character except a word character

规则 - 锚

- ^　行首
- $　行尾
- \A　字符串的开始
- \z　字符串的结尾
- \Z　字符串的结尾(不包括最后的换行符)
- \b　单词边界

规则 - 重复

- ?代表0或1个字符。/Mrs?\.?/匹配"Mr"， "Mrs"， "Mr."， "Mrs."。
- *代表0或多个字符。/Hello*/匹配"Hello"，"HelloJack"。
- +代表1或多个字符。/a+c/匹配： "abc"，"abbdrec"。
- /d{3}/匹配3个数字。
- /d{1,10}/匹配1-10个数字。
- /d{3,}/匹配3个数字以上。
- /([A-Z]\d){5}/匹配首位是大写字母，后面4个是数字的字符串。

说明

- $`
- $&
- $’
- $n
- sub(gsub)
- grep

例子

- http://www.rubular.com/
- 电话号码
- email(gary.zzhang@gmail.com)
- url(http://www.163.com)
- 省份格式化

block 块

- 借鉴函数式编程思想
- 由多行代码组成的一个代码块，可以认为是一个匿名方法。
- 可以定义在大括号 {} 中，也可以放在 do ... end 关键字中。
- 结合yield调用，把yield看做方法调用。

例子

```
{ puts "In a block" }
```

例子

```
{ puts "In a block" }
```

```
def call_block
  puts "Start of method"
  yield
  puts "End of method"
end

call_block { puts "In a block" }
```

参数

```
def who_says_what
  yield("Dave", "hello")
  yield("Andy", "goodbye")
end

who_says_what {|person, phrase| puts "#{person} says #{phrase}"}
```

询问

- block_given?

```
def a_method
  return yield if block_given?
  p "no block"
end
```

Block实例

- ['cat', 'dog', 'horse'].each {|name| p name, " " }

- 5.times { p "*" }
- 3.upto(6) {|i| p i }
- ('a'..'e').each {|char| p char }

练习题

- 5.times { p "*" }

- 实现using关键字

```
RemoteConnection conn = new RemoteConnection("remote_server")
using (conn)
{
  doSomeStuff();
}
```

```
using(resource) do

end
```