# 常用类介绍

## 数组

1. 建立一个数组 [] Array.new %W(a b c)
2. 访问数组元素
3. 增加、删除数组元素
4. 数组运算

## 字符串

1. 生成一个字符串
   1. str1 =“str1”
   2. str2 = ‘str2’
   3. str3 = %q/this is str3/
   4. str4 = %Q/this is str4/
   5. str5=<<EOV

I am s string too.

EOV

%q 用来生成单引号字符串；%Q用来生成双引号字符串。%q或者%Q后面跟着的是分隔符，可以是配对的! !；/ /；<>； ( )； [ ] ；{ }；等等。

1. 字符串操作
2. 字符串转义

双引号括起来的字符串会有转义，例如：“\n”表示换行。还有一些其它的转义符号，比如制表符之类。

单引号括起来的字符串并不会对字符串作任何解释，你看到的是什么便是什么，有一个例外：单引号字符串里的单引号需要转义。

1. 字符串内嵌表达式

在双引号扩起来的字符串中，不仅可以使用各种转义符，而且可以放置任意的。

Ruby 表达式在 #{ } 之中，这些表达式在使用这个字符串的时候被计算出值，然后放入字符串。

字符串内嵌表达式，使得你能够更加灵活地组织代码，表现出更强、更多的动态特性。

## 正则表达式

正则表达式(regular expression)描述了一种字符串匹配的模式，可以用来检查一个串是否含有某种子串；将匹配的子串做替换；或者从某个串中取出符合某个条件的子串；等等。

Ruby 中，可以使用构造器显式地创建一个正则表达式，也可以使用字面值形式 /正则模式/ 来创建一个正则表达式。

=~

Regexp

特别字符

()

标记一个子表达式的开始和结束位置。要匹配(，请使用\(。

子表达式可以获取供以后使用。$1, $2…..

[]

范围描述符 (比如,[a - z] 表示在 a 到 z 范围内的一个字母)。要匹配[，请使用\[

\

将下一个字符标记为或特殊字符、或原义字符、或向后引用、或八进制转义字符。

例如， 'n' 匹配字符 'n'。'\n' 匹配换行符。序列 '\\' 匹配 "\"，而 '\(' 则匹配“(“。

|

指明两项之间的一个选择。要匹配 |，请使用 \|。

.

匹配除换行符 \n之外的任何单字符。要匹配 .，请使用 \.

## 迭代器、代码块、闭包

(1..9).each {|i| print i if i<7}

迭代器 each 是数组类的一个方法；大括号{ }里的代码是代码块，简称块。

你可以用大括号{ }将代码组织成块，也可以用 do…end 将代码组织成块。大括号{ }的优先级高于 do…end。

defone\_block

yield

yield

yield

end

one\_block{ puts "This is a block. " }

可以看到调用一个块要用关键字 yield。每一次 yield，块就被调用一次。yield 还可以带参数调用块。

defone\_block

fornum in 1..3

yield(num)

end

end

one\_block do |i|

puts "This is block #{i}. "

end

一个块可以接收 yield 传来的参数，还可以将结果返回给调用它的方法。到目前为止，实在看不出使用代码块的优势，可以把块里的代码直接写在方法中。如果我们还没有决定块里写什么代码，又或者块里的代码会随着不同的情形而变化，那么就看出代码块的灵活性了。

defdo\_something

yield

end

do\_something do

(1..9).each {|i| print i if i<5}

puts

end

do\_something do

3.times { print "Hi!" }

puts

end

代码块是一段代码，相当于一个匿名方法，被调用它的方法所调用。如果我们不仅仅想调用代码块，还想把代码块作为参数传递给其它方法，就要使用闭包了。闭包也是一段代码，一个代码块，而且能够共享其它方法的局部变量。

闭包既然是一段代码，也就有自己的状态，属性，作用范围，也就是一个可以通过变量引用的对象，我们称之为过程对象。一个过程对象用proc创建，用 call方法来调用。

先看一个闭包作为参数传递给其它方法的例子；

def method(pr)

putspr.call(7)

end

oneProc = proc{|k| k \*=3 }

method(oneProc)

## 哈希

1. Hash的定义
   1. h = Hash.[](:a, 'a', :b, 'b')
   2. h = Hash.[](:a => 'a', :b => 'b')
   3. h = Hash[:a, 'a', :b, 'b']
   4. h = Hash[:a => 'a', :b => 'b']
   5. h = { :a => 'a', :b => 'b' }
   6. h = Hash.new
   7. {a: "1", b: "2"}
2. Hash的用法

## 文件操作

1. 文件的创建

file = File.new("testfile", "r")

第一个参数是文件的路径和名字；

第二个参数以文件的模式，可以是：

# r 只读

# w 只写

# a 追加只写

# r+, w+, a+都是读写模式，但是有区别

# r+是覆盖写，w+是清除后写，a+是追加写

# 所有模式加b就是对应的二进制模式

1. 打开文件：

File.open("testfile", "r") do |file|

#处理文件

end

1. 写入文件

File.open("E:/22.txt", "w"){|file|

file.puts("line1")

file.puts("line1")

file.close

}

1. 读取文件

file = File.open("E:/20101221.txt")

file.each{ |line|

puts line

}

file.close

## 文件路径的处理Dir类

Dir.chdir(“/tmp”) # 将当前的工作路径改为tmp

Dir.getwd # 获取当前的工作路径

Dir.rmdir("/tmp") # 删除文件夹

Dir.foreach(“testdir”) {|x| puts “Got #{x}” } # 循环文件夹中的所有文件及文件夹

Dir.mkdir(“test”) # 创建文件夹

Dir.glob(“\*.rb”) # 找出与“\*.rb”匹配的文件，返回一个数组

## 文件处理的拓展模块FileUtils模块

#生成dir目录及其所有上级目录

Require ‘fileutils’

FileUtils.mkdir\_p 'E:/usr/local/lib/ruby'

#删除目录

FileUtils.rm\_rDir.glob('E:/usr/local/lib/ruby')

# cp文件拷贝

FileUtils.cp("F:\\test1.rb", "F:\\Test“)

# 该方法经测试可以进行覆写操作，同名文件被拷贝的文件覆盖

# mv 将文件进行移动

FileUtils.mv("F:\\test1.rb", "E:\\test2.rb")

# 该方法将 test1.rb 改名为 test2.rb 并移动到 E 分区

FileUtils.mv("F:\\Ruby", "F:\\Good")

# 该方法将 Ruby 文件夹移动到 Good 文件夹下

FileUtils.mv("F:\\Good", "E:\\Good")

# >>Error!Permission denied - F:\Good or E:\Good,跨区的文件夹移动发生了错误

# cmp文件比较

FileUtils.cmp("F:\\test1.rb", "F:\\test2.rb")

# 比较两个文件是否相同，返回 true 或者 false

## Time, Date, DateTime

1. 时间的定义与使用

p t1 = Time.new

p t2 = Time.now

p t1 = Time.mktime(2010) #=> Fri Jan 01 00:00:00 +0800 2010

p t2 = Time.mktime(2010, 12) #=> Wed Dec 01 00:00:00 +0800 2010

p t3 = Time.mktime(2010, 12, 21) #=> Tue Dec 21 00:00:00 +0800 2010

p t4 = Time.mktime(2010, 12, 21, 8) #=> Tue Dec 21 08:00:00 +0800 2010

p t5 = Time.mktime(2010, 12, 21, 8, 50) #=> Tue Dec 21 08:50:00 +0800 2010

p t6 = Time.mktime(2010, 12, 21, 8, 50, 30) #=> Tue Dec 21 08:50:30 +0800 2010

p t8 = Time.gm(2010, 12, 21, 8, 51, 30) #=> Tue Dec 21 08:51:30 UTC 2010

p t9 = Time.utc(2010, 12, 21, 8, 51, 30) #=> Tue Dec 21 08:51:30 UTC 2010

p t10 = Time.local(\*Time.now.to\_a)

p t11 = Time.gm(0, 01, 9, 21, 12, 2010, 3, 324, false, "GMT-8:00")

#用秒数来创建日期

p epoch = Time.at(0)

pnewmil = Time.at(978307200)

# 转化成为秒

now = Time.now

p now.to\_i

# 获取当天是星期几

now = Time.now

p now.wday

p now.strftime("%a")

p now.strftime("%A")

# 获取一年中当天所在的周

t1 = Time.local(2010, 12, 21)

p t1.strftime("%U").to\_i

p t1.strftime("%W").to\_i

t2 =Date.new(2010, 12, 21)

p d1.cweek

# 获取当前的时区

puts Time.gm(2000,11,10,22,5,0).zone # "UTC"

puts Time.local(2000,11,10,22,5,0).zone # "中国标准时间”

t = Time.now

putst.strftime("%H:%M:%S") # 22:07:45

putst.strftime("%H:%M") # 22:07

puts (t + 30).strftime("%H:%M") # 22:08 (加了30秒)

putst.strftime("%I:%M %p") # 10:07 PM

DateTime.parse("2010-12-21 08:53:25")

DateTime.parse("Tur Dec 21 08:55:12 +0800 2010")

DateTime.parse("tur Dec 21 08:55:12 +0900 2010")

## 元编程

你编写了一个能够生成其它程序的程序，那么，你就在做元编程的工作。

Ruby 语言强大的动态特征，赋予了我们灵活地进行元编程的能力。

Wiki资料：

[http://192.168.75.10/cn/index.php?cmd=read&page=Trash%2FTraining-201012%2Flxd%20%E6%97%A5%E6%9C%9F%E5%A4%84%E7%90%86%E7%9A%84%E5%9F%BA%E6%9C%AC%E6%96%B9%E6%B3%95%2FHash%E7%B1%BB%2FArray%E7%B1%BB&word=Array#sba530e1](http://192.168.75.10/cn/index.php?cmd=read&page=Trash%2FTraining-201012%2Flxd%20%E6%97%A5%E6%9C%9F%E5%A4%84%E7%90%86%E7%9A%84%E5%9F%BA%E6%9C%AC%E6%96%B9%E6%B3%95%2FHash%E7%B1%BB%2FArray%E7%B1%BB&word=Array)

<http://192.168.75.10/cn/index.php?Trash%2FTraining-201012%2Flxd%20Array%E8%BD%AC%E6%8D%A2%E6%88%90Hash%2FFile%E7%B1%BB%E6%93%8D%E4%BD%9C%2FDir%E7%B1%BB%2FFileUtils%E6%A8%A1%E5%9D%97%2FYAML>

任务：

选出两个测试文件夹A、B

1、将文件夹A、B中所有有的文件夹和文件记录到一个txt文件中

2、将文件夹A中有，B中没有的文件夹和文件记录到一个txt文件中

3、将文件夹B中有，A中没有的文件夹和文件记录到一个txt文件中

4、将文件夹A中有，B中没有；B中有，A中没有的文件夹和文件拷贝到文件夹C中

5、将文件夹A中的文件的创建时间的范围和文件夹B中的文件的创建时间的范围记录到一个txt文件中