# [Ruby语言学习系列--基本的ruby语法](http://www.cnblogs.com/sky_online/archive/2008/08/18/1270117.html" \t "_blank)

# 

# 1.     基本的ruby语法

# 1.1      变量、常量和类型

# 1)      定义变量

|  |  |  |
| --- | --- | --- |
| 变量类型 | 描述 | 示例 |
| 局部变量（或伪变量） | 以小写字母或下划线卡头 | var    \_var |
| 全局变量 | 以$开头 | $var |
| 类变量 | 类中定义，以@@开头 | @@var |
| 实例变量 | 对象中定义，以@开头 | @var |
| 常量 | 以大写字母开头 | Var |

# 2) 变量内插

# 在双引号内使用“#{变量名}”内插变量

|  |
| --- |
| a = 2b = 3puts "#{a} + #{b} = #{a+b}"  #输入结果为：2 + 3 = 5 |

# 

# 1.2      注释

# 1）单行注释：以#开头，如： #注释内容

# 2）多行注释：在=begin 和 =end 之间定义，如：

# =begin

# 注释内容

# =end

# 1.3    循环和分支

# 1.3.1        条件语句

|  |  |
| --- | --- |
| If 形式 | unless 形式 |
| a =1 if y==3 | a=1 unless y!=3 |
| x= if a > 0 then b else c end | x= unless a<=0 then a else b end |
| if x<5 then  a =1else   a =2end | unless x<5 then   a =2else   a =1end |

# 

# 1.3.2        循环结构

|  |  |
| --- | --- |
| #while循环i= 0while i< list.size doprint “#list[i] ”I += 1end | #until循环i= 0until i == list.size doprint “#list[i]”i += 1end |
| #for循环for x in lisy do print “#{x}”end | #each循环list.each do |x| print “#{x}”end |
| #loop循环i = 0n = list.size-1loop doprint “#{list[i]}”i += 1break id i > nend | #times循环n = list.sizen.times do |i| print “#{list[i]}”end |
| #upto循环n =list.size–10.upto(n) do |i| print “#{list[i]}”end | #each\_index循环list.each\_index do |x| print “#{list[x]}”end |

# 

# 1.3.3        异常

|  |
| --- |
| begin    x = Math.sqrt(y/z)rescue ArgumentError       #匹配错误类型    puts "Error taking square root"rescue ZeroDivisionError     #匹配错误类型    puts "Attempted division by zero"else    puts "Other Error"ensure    #这部分代码在最后一定会执行end |

# 

# 1.3.4        类

|  |
| --- |
| class Hello #类名必须以大写字母开头 @@myname = "John"   #类变量 def initialize(name,phone)      @name,@phone = name,phone #定义实例变量 end def hello #定义类方法    puts "#{@name}的电话是#{@phone}" endendf = Hello.new("Tom","13100000000") #创建类对象f.hello #调用类方法，结果为：Tom的电话是13100000000 |

# 

# 1.3.5        模块

|  |  |
| --- | --- |
| 方法 | 说明 |
| include ‘watir’ | 将模块的功能添加到当前空间中，不加载已加载的文件 |
| extend ‘watir’ | 将模块的函数添加到对象中 |
| load ‘watir’ | 读取文件，并将其插入到源文件的当前位置，以便从这个位置开始就可以使用其定义 |

# 

# 1.3.6        case语句

|  |
| --- |
| case x when 1..10              #匹配数字    puts "First branch" when foobar()           #批量方法返回的值    puts "Second branch" when /^hel.\*/           #匹配正则表达式    puts "Third branch" else    puts "Last branch"end |

# 

# 

# 

# 

# 

# [Ruby语言学习系列--String 类函数](http://www.cnblogs.com/sky_online/archive/2008/08/25/1275854.html" \t "_blank)

# 

# 

|  |  |  |
| --- | --- | --- |
| 函数名称 | 说明 | 示例 |
| \* | 将字符串拷贝N次 | “ha”\*4    >> “hahahaha” |
| +<<concat | 连接字符串 | “yes” + “no”  >> “yesno”"yes" << "no" >> “yesno”"yes".concat("no") |
| <=> | 比较字符串，返回值如下：大于：-1   等于：0    小于：1 | "Ab" <=> "ab" >> -1"ab" <=> "ab" >> 0"ab" <=> "Ab" >> 1 |
| == 或 === | 判断两个对象是否相等 | "1" == "1"   >>true"1" == 1     >>flase |
| =~ | 匹配正则表达式 | "abc123abc" =~ /"d/ >> 3 |
| [ ] 或 slice | 返回字符串的某一范围内的值 | “abc”[0,2]           >> “ab”"hello"[/llo/]          >> “llo”“abc” slice [0,2]      >> “ab” |
| [ ]= | 替换字符串的某一范围内的值 | a = “hello word”a[1,2]= “OO”puts a    >>” hOOlo word” |
| capitalizecapitalize! | 把字符串的首字母大写，其他字母小写 | "hi,Ruby".capitalize>> “Hi,ruby” |
| chompchomp! | 删除字符串后的空白字符 | "string"r"n".chomp>> ” string” |
| chop | 删除最后一个字符 | "string".chop >> “strin” |
| count | 返回该字符串中含的字符个数 | a = "hello world"a.count "lo" >> 5(l出现 3次，o出现 2次) |
| deletedelete! | 删除字符 | "hello".delete "l","lo"  » "heo""hello".delete "lo"      » "he" |
| downcasedowncase！ | 将大写字母改写为小写 | "hEllO".downcase  » "hello" |
| upcaseupcase！ | 将小写字母改写为大写 | "hEllO".upcase » "HELLO" |
| swapcase swapcase! | 将所有的大写字母改为小写字母，小写字母改为大写字母。 | "Hello".swapcase » "hELLO" |
| each | 对字符串中的各行进行迭代操作 | "Hi"nRuby". each { |s| puts s} |
| each\_byte | 对字符串中的各个字节进行迭代操作 | "Hi"nRuby". each\_byte { |s| puts s} |
| each\_line | 对字符串中的每一行进行迭代操作 | "Hi"nRuby". each\_line { |s| puts s} |
| empty? | 判断字符串是否为空 | "hello".empty?    » false"".empty?         » true |
| gsubgsub！ | 以replace来替换字符串中所有与pattern相匹配的部分 | "hello".gsub(/[aeiou]/, '\*')  » "h\*ll\*" |
| hash | 返回字符串的哈希值 | "h".hash    >> 107 |
| include? | 若字符串中包含substr子字符串的话,就返回真 | "hello".include? "lo" » true"hello".include? "ol" » false |
| index | 按照从左到右的顺序搜索子字符串,并返回搜索到的子字符串的左侧位置. 若没有搜索到则返回nil | "hello".index('lo')     » 3"hello".index('a')      » nil |
| length | 返回字符串的字节数 | "hello".length   >> 5 |
| replace | 替换字符串的内容 | s = "hello"           » "hello"s.replace "world"     » "world" |
| sub 或 sub! | 用replace来替换首次匹配pattern的部分。 | "hello".sub(/[aeiou]/, '\*')» "h\*llo" |
| reversereverse! | 对字符串进行反转 | "stressed".reverse » "desserts" |
| scan | 使用正则表达式re反复对self进行匹配操作,并以数组的形式返回匹配成功的子字符串 | a = "cruel world"a.scan(/"w+/)    » ["cruel", "world"]a.scan(/.../)      » ["cru", "el ", "wor"] |
| split | 使用sep指定的pattern来分割字符串,并将分割结果存入数组 | "mellow yellow".split("ello")» ["m", "w y", "w"] |
| squeezesqueeze! | 压缩由str所含字符构成的重复字符串 | "yellow moon".squeeze» "yelow mon"" now   is the".squeeze(" ")» " now is the" |
| stripstrip! | 删除头部和尾部的所有空白字符。空白字符是指" "t"r"n"f"v"。 | "    hello    ".strip » "hello"""tgoodbye"r"n".strip » "goodbye" |
| tr 或tr! | 若字符串中包含search字符串中的字符时，就将其替换为replace字符串中相应的字符 | hello".tr('aeiou', '\*')     » "h\*ll\*""hello".tr('^aeiou', '\*')   » "\*e\*\*o" |
| tr\_s 或tr\_s！ | 若字符串中包含search字符串中的字符时，就将其替换为replace字符串中相应的字符。同时，若替换部分中出现重复字符串时，就将其压缩为1个字符 | "hello".tr\_s('l', 'r')       » "hero""hello".tr\_s('el', '\*')      » "h\*o""hello".tr\_s('el', 'hx')    » "hhxo" |
| upto | 在从self到max的范围内，依次取出下一个字符串”后将其传给块，进行迭代操作 | "a1".upto("a3") {|s| puts s}» a1"na2"na3 |
| to\_f | 将字符串转为浮点数 | "45.67 degrees".to\_f » 45.67 |
| to\_i | 将字符串转为整数 | "99 red balloons".to\_i » 99 |
| to\_s | 将字符串转为字符串 |  |

# 

# 

# 

# [ruby学习系列--Hash类函数](http://www.cnblogs.com/sky_online/archive/2008/08/28/1278520.html" \t "_blank)

# 

# 

|  |  |  |
| --- | --- | --- |
| 函数名称 | 说明 | 示例 |
| == | 判断两个Hash是否相等 | h1 = {"a" => 1, "c" => 2}h2 = { "a" => 1, "c" => 2, 7 => 35 }h1 == h2    » false |
| [ ] | 返回指定键值对应的对象 | h = { "a" => 100, "b" => 200 }h["a"] » 100h["c"] » nil |
| [ ]= | 向Hash添加记录 | h = { "a" => 100, "b" => 200 }h["a"] = 9h["c"] = 4h » {"a"=>9, "b"=>200, "c"=>4} |
| clear | 清空哈希表的内容. 返回self. | h = { "a" => 100, "b" => 200 }h.clear    » {} |
| default | 返回哈希表的默认值 | h = { "a" => 100, "b" => 200 }h.default = "Go fish"h["a"] » 100h["z"] » "Go fish" |
| delete | 从词典中删除和键值相符的记录 | h = { "a" => 100, "b" => 200 }h.delete("a")           » 100h.delete("z")           » nilh.delete("z") { |el| "#{el} not found" } » "z not found" |
| delete\_if | 通过过程块来删除特定键值的记录 | h = { "a" => 100, "b" => 200 }h.delete\_if {|key, value| key >= "b" }» {"a"=>100} |
| each | Hash表的迭代操作，对表的每一个词对进行迭代操作 | h = { "a" => 100}h.each {|key, value| print key, " is ",value,""n"}         » a is 100 |
| each\_key | 对表的每一个键对进行迭代操作 | h = { "a" => 100, "b" => 200 }h.each\_key {|key| print key }     »  ab |
| each\_value | 针对value进行迭代操作 | h = { "a" => 100, "b" => 200 }h.each\_value {|value| print value }» 100200 |
| empty? | 判断哈希表是否为空，空则返回true | {}.empty?   » true |
| fetch | 如果能找到键值为key的hash值，则返回Hash值；如果找不到，则返回默认值或指定值；如果默认值和指定值都找不到，抛异常 | h = { "a" => 100, "b" => 200 }h.fetch("a")           » 100h.fetch("z", "go fish") » "go fish"h.fetch("k")»in `fetch': key not found (IndexError) |
| has\_key? | 判断是否存在相符的key值 | h = { "a" => 100, "b" => 200 }h.has\_key?("a") » trueh.has\_key?("z") » false |
| key？ | h. key?("z")   » false |
| include? | h.include?("b")   » true |
| has\_value? | 判断是否存在相符的value值 | h = { "a" => 100, "b" => 200 }h.has\_value?(100) » trueh.has\_value?(999) » false |
| value? | h.value?(100) » trueh.value?(999) » false |
| index | 返回给定值的键值，未找到返nil | h = { "a" => 100, "b" => 200 }h.index(200) » "b"h.index(999) » nil |
| indexes | 返回一系列给定的键值对应值组成的数组 | h = { "a" =>100, "b" =>200, "c" =>300 }h.indexes("a", "c")     » [100,300]h.indexes("a","z")      » [100,nil] |
| indices | h.indexes("a", "c")     » [100,300]h.indexes("a", "z")   » [100,nil] |
| invert | 将元素值和索引互换,返回变换后的哈希表.注意:若原哈希表中若干不同的索引对应相同的元素值时,其变换结果将无法预测. | h = { "n" => 100,"m" => 100,"y" => 300}h.invert   » {300=>"y",100=>"n"} |
| keys | 返回一个包含所有key的数组. | h = { "a" => 100, "b" => 200 }h.keys         » ["a", "b"] |
| values | 返回一个包含所有vakue的数组. | h = { "a" => 100,"b" => 200,"c" => 300 }h.values     » [100, 200, 300] |
| length | 返回词典中元素的个数 | h = { "d" =>100, "a"=> 200, "v" => 300}h.length     » 3h. size        » 3 |
| size |
| rehash | 重新计算索引对应的哈希表值。当与索引对应的哈希表值发生变化时，若不使用该方法来重新计算的话，将无法取出与索引对应的哈希表值。 | a = [ "a", "b" ]c = [ "c", "d" ]h = { a => 100, c => 300 }a[0] = "z"h.rehash» {["z", "b"]=>100, ["c", "d"]=>300}h[a] » 100 |
| replace | 以另外一张Hash表的内容来替换当前Hash表的内容 | h = {"a" =>100, "b" =>200 }h.replace({ "c" => 300, "d" => 400 })» {"c"=>300, "d"=>400} |
| shift | 删除一个哈希表元素后再以[key,value]数组的形式将其返回 | h = { 1 => "a", 2 => "b", 3 => "c" }h.shift » [1, "a"]h » {2=>"b", 3=>"c"} |
| sort | 对Hash进行排序按键值从小到大排序 | h = { "b" => 30, "a" => 20, "c" => 10 }h.sort » [["a", 20], ["b", 30], ["c", 10]]h.sort {|a,b| a[1]<=>b[1]}» [["c", 10], ["a", 20], ["b", 30]] |
| to\_a | 把Hash表转换为数组数组按Hash表的键值从小到大排序 | h = {"c" =>300,"a" =>100,"d" =>400}»  [["a", 100], ["c", 300], ["d", 400]] |
| to\_s | 把Hash表转换为字符串 | h = { "c" => 300, "a" => 100, "d" => 400}h.to\_s     » "a100c300d400" |
| update | 用一张Hash表去更新另外张Hash表 | h1 = { "a" => 100, "b" => 200 }h2 = { "b" => 254, "c" => 300 }h1.update(h2)» {"a"=>100, "b"=>254, "c"=>300} |

# 

# 

# 

# 

# 

# 

# 

# 

# 

# [Ruby语言学习系列--Array类函数](http://www.cnblogs.com/sky_online/archive/2008/08/25/1275855.html" \t "_blank)

# 

# 

|  |  |  |
| --- | --- | --- |
| 函数名称 | 说明 | 示例 |
| & | 数组与，返回两数组的交集 | [1,2] & [2,3] » [2] |
| \* | 复制数组n次 | [1,2]\*2       » [1,2,1,2] |
| + | 返回两数组的并集，但不排除重复元素 | [1,2]+[2,3]    » [1,2,2,3] |
| << | 追加元素，但不排除重复元素 | [1,2]<<[2,3]   » [1,2,2,3] |
| | | 追加元素，但排除重复元素 | 1,2] | [2,3]    » [1,2,3] |
| - | 返回第一个数组与第二个数组不同的元素 | [1,2]-[2,3]    » [1] |
| <=> | 比较数组 | [1,2]<=>[2,3] »flase |
| == | 比较数组，若所有元素均相等时返回真 | [1,2]==[2,1]    »flase |
| assoc | 从数组的每个元素中寻找指定对象 | [[1,2],[3,4]].assoc(2) » [1,2] |
| at | 找到数组的第N个元素负数表示逆向查找 | ["a","b","c","d","e"].at(0) » "a"["a","b","c","d","e"].at(-1)  » "e" |
| clear | 删除数组中的所有元素 | ["a","b","c","d","e"]. clear |
| collectcollect！ | 用一个过程块对数组的每个元素进行处理 | ["a","b","c","d"].collect {|x| x + "!" }» ["a!", "b!", "c!", "d!"] |
| compactcompact! | 删除值为nil的元素后生成新数组并返回它 | ["a",nil,"b",nil,"c",nil].compact» ["a", "b", "c"] |
| delete | 删除元素，如果元素重复，全部删除 | a = [ "a", "b", "b", "b", "c" ]a.delete("b")puts a    » ["a","c"] |
| delete\_at | 删除pos所指位置的元素并返回它。若pos超出数组范围则返回nil | a = %w( ant bat cat dog )a.delete\_at(2)   » "cat"a               » ["ant", "bat", "dog"]a.delete\_at(99)  » nil |
| delete\_if | 根据条件删除 | a = [ "a", "b", "c" ]a.delete\_if {|x| x >= "b" } » ["a"] |
| each | 对数组的每个元素按值进行迭代操作 | a = [ "a", "b", "c" ]a.each {|x| print x, " -- " }»  "a -- b -- c --" |
| each\_index | 对数组的每个元素按索引进行迭代操作 | a = [ "a", "b", "c" ]a.each\_index {|x| print x, " -- " }» "0 -- 1 -- 2 --" |
| empty? | 判断数组是否为空，为空则返回真 | [].empty? » true |
| eql！ | 比较两数组是否相等 | ["a","b","c"].eql?(["a","b","c"]) » true |
| fill | 填充数组 | ["a","b","c","d"].fill("x")» ["x","x","x","x"]["a","b","c","d"].fill("z", 2, 2)» ["x", "x", "z", "z"] |
| first | 返回数组的首元素。若没有首元素则返回nil | [ "q", "r", "s", "t" ].first    » "q" |
| last | 返回数组末尾的元素。若数组为空时，返回nil | ["w","x","y","z"].last      » "z" |
| include? | 判断数组中是否包含元素 | a = [ "a", "b", "c" ]a.include?("b") » truea.include?("z") » false |
| index | 返回数组中第一个==val的元素的位置 | a = [ "a", "b", "c" ]a.index("b") » 1a.index("z") » nil |
| indexes | 以数组形式返回其索引值与各参数值相等的元素 | a = [ "a", "b", "c", "d", "e", "f", "g" ]a.indexes(0, 2, 4)     » ["a", "c", "e"]a.indexes( 2, 4, 12)    » [ "c", "e", nil] |
| insert | 在索引为nth的元素前面插入第2参数以后的值 | ary = %w(foo bar baz)ary.insert 2,'a','b'p ary   » ["foo", "bar", "a", "b", "baz"] |
| join | 将数组元素按一定的分隔符连接起来 | [ "a", "b", "c" ].join » "abc"[ "a", "b", "c" ].join("-") » "a-b-c" |
| lengthsize | 返回数组长度。若数组为空则返回0 | [1,2,3].length     » 3[1,2,3].size       » 3 |
| nitems | 返回非nil元素的个数 | [ 1, nil, 3, nil, 5 ].nitems » 3 |
| pop | 删除末尾元素并返回它。若数组为空则返回nil | a = [ "a", "m", "z" ]a.pop               » "z"p a                 » ["a", "m"] |
| push | 添加新元素 | ["a","b"].push(['1','2'])» ["a", "b", ["1", "2"]] |
| rassoc | 遍历数组每个元素（元素必须是数组），匹配索引为1的值是否与查找的字符相等，返回第一个相等的元素 | a = [[15,1], [25,2], [35,2]]p a.rassoc(2)           » [25, 2] |
| replace | 替换数组元素 | a = ["a","b"]a.replace(["x","y","z"])p a                    »["x", "y", "z"] |
| reversereverse！ | 将所有元素以逆序重新排列生成新数组并返回它 | ["a","b","c" ].reverse » ["c", "b", "a"] |
| rindex | 返回最后一个值相等的元素的索引值 | a = [ "a","b","b","b","c"]a.rindex("b")              » 3 |
| shift | 删除数组的首元素并返回它。剩余元素依次提前。若数组为空返回nil。 | args = ["-m","-q","filename"]args.shift » "-m"args » ["-q", "filename"] |
| sortsort! | 从小到大排序 | a = [ "d", "a", "e", "c", "b" ]a.sort   » ["a", "b", "c", "d", "e"] |
| uniquniq！ | 删除数组中的重复元素后生成新数组并返回它 | a = [ "a", "a", "b", "b", "c" ]a.uniq      » ["a", "b", "c"] |
| unshift | 在数组第一个元素前添加元素 | a = [ "b", "c", "d" ]a.unshift("a")   » ["a", "b", "c", "d"] |
| to\_s | 将数组的所有元素连接成字符串 | ["a","e","i","o"].to\_s     » "aeio" |

# 

# 

# 

# 

# [Ruby语言学习系列--Numeric类函数](http://www.cnblogs.com/sky_online/archive/2008/08/25/1275851.html" \t "_blank)

# 

# 

|  |  |  |
| --- | --- | --- |
| 函数名称 | 说明 | 示例 |
| chr | 返回数字的ACSII码 | 65.chr   >> “A” |
| downto | 接收一个block,从大到小循环执行 | 5.downto(2) { |i| puts i}>> 5 4 3 2 |
| upto | 接收一个block,从小到大循环执行 | 2.upto(5) { |i| puts i}>> 2 3 4 5 |
| next或succ | 返回下一个数 | 1.next >> 21.succ >>2 |
| step | 以固定步长循环执行 | 1.step(10,2) { |i| puts i}>> 1 2 3 5 7 |
| times | 循环执行n次 | 5.times { |i| puts i}>> 1 2 3 4 5 |

# 

# 

# 

# [Ruby语言学习系列--Float类函数](http://www.cnblogs.com/sky_online/archive/2008/08/25/1275852.html" \t "_blank)

# 

# 

|  |  |  |
| --- | --- | --- |
| 函数名称 | 说明 | 示例 |
| ceil | 返回比float大的最小整数 | (2.98).ceil   >> 3 |
| floor | 返回比float小的最大整数 | (2.98). floor   >> 2 |
| round | 四舍五入到一个整数 | (2.98). round >> 3 |
| to\_i | 返回float截掉小数点后的整数 | (2.98).to\_i    >> 2 |

# --------------------- 本文来自 sunyllove 的CSDN 博客 ，全文地址请点击：https://blog.csdn.net/sunyllove/article/details/51670796?utm\_source=copy