13章 数组类

作者 bluetea
网站:https://github.com/bluetea

通过Array.new创建

```
[113] pry(main)> a= Array.new
=> []
[114] pry(main)> a= Array.new(5)
=> [nil, nil, nil, nil, nil]
[115] pry(main)> a= Array.new(5, 2)
=> [2, 2, 2, 2, 2]
```

13.2.2 使用%w和%i
%w可节省输入”“和使程序简洁
%i 创建符号数组

```
[116] pry(main)> lang = %w(ruby perl python scheme pike rebol)
=> ["ruby", "perl", "python", "scheme", "pike", "rebol"]
[117] pry(main)> lang = %i(ruby perl python scheme pike rebol)
=> [:ruby, :perl, :python, :scheme, :pike, :rebol]
```

在创建数组中用（ ）将元素括了起来，其实还可以使用<>,||, !!,@@, AA等成对出现创建，但是常用的就只有 (), <> , ||

```
[1] pry(main)> lang = %i(ruby perl python scheme pike rebol)
=> [:ruby, :perl, :python, :scheme, :pike, :rebol]
[2] pry(main)> lang = %i|ruby perl python scheme pike rebol|
=> [:ruby, :perl, :python, :scheme, :pike, :rebol]
[3] pry(main)> lang = %i<ruby perl python scheme pike rebol>
=> [:ruby, :perl, :python, :scheme, :pike, :rebol]
```

13.2.3 to_a方法

```
[6] pry(main)> h1 = {black: "#000000", white: "#FFFFF"}
=> {:black=>"#000000", :white=>"#FFFFF"}
[7] pry(main)> h1.to_a
=> [[:black, "#000000"], [:white, "#FFFFF"]]
```

13.2.4 split方法

```
[11] pry(main)> column = "2011/12/13 foo.html proxy.jpt"
=> "2011/12/13 foo.html proxy.jpt"
[12] pry(main)> column.split()
=> ["2011/12/13", "foo.html", "proxy.jpt"]
[13] pry(main)> column = "2011/12/13,foo.html,proxy.jpt"
=> "2011/12/13,foo.html,proxy.jpt"
[14] pry(main)> column.split(",")
=> ["2011/12/13", "foo.html", "proxy.jpt"]
```

13.3 索引的使用
13.3.1.获取全速
a[n]
a[n..m] 或者 a[m..n]
a[n, len]

```
[15] pry(main)> a = %w(a, b, c, d, e)
=> ["a,", "b,", "c,", "d,", "e"]
[16] pry(main)> a[3]
=> "d,"
[17] pry(main)> a[2..3]
=> ["c,", "d,"]
[18] pry(main)> a[1, 2]
=> ["b,", "c,"]
[19] pry(main)> a[1, 3]
=> ["b,", "c,", "d,"]
```

索引为负数

```
[21] pry(main)> a
=> ["a,", "b,", "c,", "d,", "e"]
[22] pry(main)> a[-1]
=> "e"
[23] pry(main)> a
=> ["a,", "b,", "c,", "d,", "e"]
[24] pry(main)> a[1..7]
=> ["b,", "c,", "d,", "e"]
```

13.3.2元素的赋值

```
[25] pry(main)> a
=> ["a,", "b,", "c,", "d,", "e"]
[26] pry(main)> a[2..4] = ["C", "D", "E"]
=> ["C", "D", "E"]
[27] pry(main)> a
=> ["a,", "b,", "C", "D", "E"]
[28] pry(main)>
```

13.3.3 插入元素

插入元素其实也可以被认为是对第0个元素进行赋值，因此指定[n, 0]后，就会在索引为n的元素前插入新元素,0表示纯插入，如果这个数字为2，表示这个元素的后2位替换为新的

```
[35] pry(main)> a = [1,2,3,4,5,6,7]
=> [1, 2, 3, 4, 5, 6, 7]
[36] pry(main)> a[2, 3] = [22]
=> [22]
[37] pry(main)> a
=> [1, 2, 22, 6, 7]
```

13.3.4通过多个索引生成新的数组

通过Array#valus_at获得零散的数据

```
[40] pry(main)> a = %w(a b c d e f g)
=> ["a", "b", "c", "d", "e", "f", "g"]
[41] pry(main)> a.values_at(1,3,5,6)
=> ["b", "d", "f", "g"]
```

13.4 把数组作为集合来看待

1.取出同时属于两个集合的元素， 并创建新的集合

交集 ary = ary1 & ary2

```
[48] pry(main)> a
=> ["a", "b", "c", "d", "e"]
[49] pry(main)> b
=> ["c", "d", "e", "f", "g", "h"]
[50] pry(main)> a & b
=> ["c", "d", "e"]
```

2.取出两个集合中所有元素， 并创建新的集合

并集 ary = ary1 | ary2

```
[45] pry(main)> a
=> ["a", "b", "c", "d", "e"]
[46] pry(main)> b
=> ["c", "d", "e", "f", "g", "h"]
[47] pry(main)> a | b
=> ["a", "b", "c", "d", "e", "f", "g", "h"]
```

注意 + 与 | 的不同

```
[51] pry(main)> a
=> ["a", "b", "c", "d", "e"]
[52] pry(main)> b
=> ["c", "d", "e", "f", "g", "h"]
[53] pry(main)> a +b
=> ["a", "b", "c", "d", "e", "c", "d", "e", "f", "g", "h"]
[54] pry(main)> a | b
=> ["a", "b", "c", "d", "e", "f", "g", "h"]
```

13.5 作为列的数组

数组结构的队列（queue）和栈（stack）都是典型的列结构

queue 是FIFO (first-in first-out)模型

Stack 是LIFO（last-in frist-out）模型

对数组头的元素操作

1.追加元素 unshift

2.删除元素 shift

3.引用元素 first

```
[61] pry(main)> a
=> ["a", "b", "c", "d", "e"]
[62] pry(main)> a.unshift("a1")
=> ["a1", "a", "b", "c", "d", "e"]
[63] pry(main)> a.shift
=> "a1"
```

```
[66] pry(main)> a
=> ["a", "b", "c", "d", "e"]
[67] pry(main)> a.first
=> "a"
```

对数组尾的元素操作

```
[69] pry(main)> a.push("f")
=> ["a", "b", "c", "d", "e", "f"]
[70] pry(main)> a.pop
=> "f"
[71] pry(main)> a.last
=> "e"
[72] pry(main)> a
=> ["a", "b", "c", "d", "e"]
```

数组的一些操作方法

```
[85] pry(main)> a
=> ["a", "b"]
[86] pry(main)> a.concat(["c"])
=> ["a", "b", "c"]
[87] pry(main)> a.concat(["c", "d"])
=> ["a", "b", "c", "c", "d"]
```

```
[91] pry(main)> a
=> ["a", "b", "c", "c", "d"]
[92] pry(main)> a[1..3] = 0
=> 0
[93] pry(main)> a
=> ["a", 0, "d"]
```

13.6.2 从数组中删除nil

```
[94] pry(main)> a = [1,nil, 3, nil ,nil, 4]
=> [1, nil, 3, nil, nil, 4]
[95] pry(main)> a.compact
=> [1, 3, 4]
[96] pry(main)> a
=> [1, nil, 3, nil, nil, 4]
[97] pry(main)> a.compact!
=> [1, 3, 4]
[98] pry(main)> a
=> [1, 3, 4]
```

从数组中删除x元素

```
[99] pry(main)> a = [1,2,3,4,3,2]
=> [1, 2, 3, 4, 3, 2]
[100] pry(main)> a.delete(3)
=> 3
[101] pry(main)> a
=> [1, 2, 4, 2]
```

删除第n个元素

```
[102] pry(main)> a = [1,2,3,4,5,6]
=> [1, 2, 3, 4, 5, 6]
[103] pry(main)> a.delete_at(3)
=> 4
[104] pry(main)> a
=> [1, 2, 3, 5, 6]
```

根据块判断删除

```
[107] pry(main)> a
=> [1, 2, 4, 3, 5, 6]
[108] pry(main)> a.delete_if{|i| i >3}
=> [1, 2, 3]
[109] pry(main)> a
=> [1, 2, 3]
```

删除指定的n..m元素

```
[110] pry(main)> a = [1,2,3,4,5,6]
=> [1, 2, 3, 4, 5, 6]
[111] pry(main)> a.slice(2..4)
=> [3, 4, 5]
[112] pry(main)> a
=> [1, 2, 3, 4, 5, 6]
[113] pry(main)> a.slice!(2..4)
=> [3, 4, 5]
[114] pry(main)> a
=> [1, 2, 6]
```

删除数组中的重复元素

```
[115] pry(main)> a = [1,1,2,2,3,3]
=> [1, 1, 2, 2, 3, 3]
[116] pry(main)> a.uniq
=> [1, 2, 3]
[117] pry(main)> a
=> [1, 1, 2, 2, 3, 3]
[118] pry(main)> a.uniq!
=> [1, 2, 3]
[119] pry(main)> a
=> [1, 2, 3]
```

13.6.3 替换数组元素

collect方法, collect!会替换原数组

```
[123] pry(main)> a
=> [1, 2, 3]
[124] pry(main)> a.collect {|item| item * 2}
=> [2, 4, 6]
[125] pry(main)> a
=> [1, 2, 3]
[126] pry(main)> a.collect! {|item| item * 2}
=> [2, 4, 6]
[127] pry(main)> a
=> [2, 4, 6]
```

map方法

```
[130] pry(main)> a
=> [1, 2, 3]
[131] pry(main)> a.map! {|item| item*3}
=> [3, 6, 9]
```

fill方法 都会替换原有值

a.fill(value)

a.fill(value, begin)

a.fill(value, begin, len)

a.fill(value, n..m)

```
[141] pry(main)> a = [1,2,3,4,5]
=> [1, 2, 3, 4, 5]
[142] pry(main)> a.fill(0)
=> [0, 0, 0, 0, 0]
```

```
[135] pry(main)> a = [1,2,3,4,5]
=> [1, 2, 3, 4, 5]
[136] pry(main)> a.fill(0, 2)
=> [1, 2, 0, 0, 0]
[137] pry(main)> a = [1,2,3,4,5]
=> [1, 2, 3, 4, 5]
[138] pry(main)> a.fill(0, 2, 1)
=> [1, 2, 0, 4, 5]
[139] pry(main)> a = [1,2,3,4,5]
=> [1, 2, 3, 4, 5]
[140] pry(main)> a.fill(0, 2..4)
=> [1, 2, 0, 0, 0]
```

a.flatten 将套嵌数组变为一个大数组

```
[146] pry(main)> a = [[1,2], [3,4]]
=> [[1, 2], [3, 4]]
[147] pry(main)> a.flatten
=> [1, 2, 3, 4]
[148] pry(main)> a
=> [[1, 2], [3, 4]]
[149] pry(main)> a.flatten!
=> [1, 2, 3, 4]
[150] pry(main)> a
=> [1, 2, 3, 4]
```

a.reverse 翻转数组内容

```
[151] pry(main)> a
=> [1, 2, 3, 4]
[152] pry(main)> a.reverse
=> [4, 3, 2, 1]
[153] pry(main)> a
=> [1, 2, 3, 4]
[154] pry(main)> a.reverse!
=> [4, 3, 2, 1]
[155] pry(main)> a
=> [4, 3, 2, 1]
```

a.sort

a.sort!

a.sort {|i,j| ...}

a.sort! {|i,j| ...}

```
[161] pry(main)> a
=> [3, 56, 1, 99]
[162] pry(main)> a.sort{|i,j| i <=> j}
=> [1, 3, 56, 99]
[163] pry(main)> a.sort{|i,j| j <=> i}
=> [99, 56, 3, 1]
```

a.sort_by{|i| ...}

```
[161] pry(main)> a
=> [3, 56, 1, 99]
[162] pry(main)> a.sort{|i,j| i <=> j}
=> [1, 3, 56, 99]
[163] pry(main)> a.sort{|i,j| j <=> i}
=> [99, 56, 3, 1]
[164] pry(main)> a
=> [3, 56, 1, 99]
[165] pry(main)> a.sort_by {|i| i}
=> [1, 3, 56, 99]
[166] pry(main)> a.sort_by {|i| -i}
=> [99, 56, 3, 1]
[167] pry(main)> a = %w(ruby pthon c++ ObjectC Perl)
=> ["ruby", "pthon", "c++", "ObjectC", "Perl"]
[168] pry(main)> a.sort_by{|i| i.size}
=> ["c++", "ruby", "Perl", "pthon", "ObjectC"]
```

13.8 处理数组中的元素

1.for循环与索引

```
random.rb
list =["ruby", "pthon", "c++", "ObjectC", "Perl"]
for i in 0..4
  print "第 #{i+1} 个元素是 #{list[i]}.\n"
end
```

```
2. bash
ruby    bash
wangmjcdeMacBook-Pro:ruby wangmjc$ ruby random.rb
第 1 个元素是 ruby.
第 2 个元素是 pthon.
第 3 个元素是 c++.
第 4 个元素是 ObjectC.
第 5 个元素是 Perl.
```

while 方法

```
random.rb
list =["ruby", "pthon", "c++", "ObjectC", "Perl"]
i = 0
while list[i] != nil
  p list[i]
  i += 1
end
```

```
2. bash
ruby    bash
wangmjcdeMacBook-Pro:ruby wangmjc$ ruby random.rb
"ruby"
"pthon"
"c++"
"ObjectC"
"Perl"
wangmjcdeMacBook-Pro:ruby wangmjc$
```

each方法

each_with_index方法

```
random.rb
list =["ruby", "pthon", "c++", "ObjectC", "Perl"]
list.each_with_index do |value, index|
  print "#{index+1} = #{value}\n"
end
```

```
2. bash
ruby    bash
wangmjcdeMacBook-Pro:ruby wangmjc$ ruby random.rb
1 = ruby
2 = pthon
3 = c++
4 = ObjectC
5 = Perl
```

13.9数组的元素

13.9.1使用简单的矩阵

数组的各个元素还是数组，就叫矩阵

```
[173] pry(main)> a
=> [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
[174] pry(main)> a[1][1]
=> 5
```

把数组对象或者散列对象作为数组元素时，注意

```
[11] pry(main)> a = Array.new(3, [0, 0, 0])
=> [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

下面只想改变一个值，都改变为什么？

```
[11] pry(main)> a = Array.new(3, [0, 0, 0])
=> [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
[12] pry(main)> a[0][1] = 2
=> 2
[13] pry(main)> a
=> [[0, 2, 0], [0, 2, 0], [0, 2, 0]]
```

这是因为a数组内的3个[0,0,0]引用的都是同一个数组，所以为了避免这个问题，只能这么干

```
[15] pry(main)> a = Array.new(3) do
[15] pry(main)*   [0,0,0]
[15] pry(main)* end
=> [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
[16] pry(main)> a[0][1] =2
=> 2
[17] pry(main)> a
=> [[0, 2, 0], [0, 0, 0], [0, 0, 0]]
```

,

调用块去生成对象，这样一来，各个元素引用同一个对象的问题就不再发生了

```
[18] pry(main)> a = Array.new(5) {|i| i}
=> [0, 1, 2, 3, 4]
```

13.10同时访问多个数组

```
ary1 = [1,2,3,4,5]
ary2 = [10, 20, 30, 40,50]
ary3 = [100, 200, 300, 400, 500]
i = 0
result = []
while i <ary1.length
  result << ary1[i] + ary2[i] + ary3[i]
  i += 1
end
p result
```

```
2. bash
ruby    bash
wangmjcdeMacBook-Pro:ruby wangmjc$
```

使用zip方法可以让程序变得更简单,zip方法会将接收器和参数传来的数组元素逐一取出

```
ary1 = [1,2,3,4,5]
ary2 = [10, 20, 30, 40,50]
ary3 = [100, 200, 300, 400, 500]
i = 0
result = []
ary1.zip(ary2, ary3) do |a, b, c|
  result << a+ b+ c
end

p result
```

```
2. bash
ruby    bash
wangmjcdeMacBook-Pro:ruby wangmjc$ ruby random.rb
[111, 222, 333, 444, 555]
```

Enumerable 模块，英语意思是可以被计数的，可以被列举的

如下的类中都包含了Enumerale的

Array

Dir

File

IO

Range

Enumerator