

## 15章 散列类

作者 bluetea

网站 <https://github.com/bluetea>

### 15.2 散列的创建

#### 1.用符号创建

```
[29] pry(main)> h = {:a => "b", :c => "d"}
=> {:a=>"b", :c=>"d"}
[30] pry(main)> h[a]
=> nil
[31] pry(main)> h[:a]
=> "b"
[32] pry(main)> h = {a: "b", c: "d"}
=> {:a=>"b", :c=>"d"}
[33] pry(main)> h[:a]
=> "b"
```

#### 2.用字面量创建

```
[34] pry(main)> h = {"a" => "b", "c" => "d"}
=> {"a"=>"b", "c"=>"d"}
[35] pry(main)> h[a]
=> nil
[36] pry(main)> h[:a]
=> nil
[37] pry(main)> h["a"]
=> "b"
```

散列的键值可以使用各种对象,但是一般建议如下几个

- 1.字符串
- 2.数值
- 3.符号
- 4.日期

#### 值的设定与获取

```
[45] pry(main)> h = Hash.new
=> {}
[46] pry(main)> h.store("r", "ruby")
=> "ruby"
[47] pry(main)> h.fetch("r")
=> "ruby"
```

但是fetch方法如果找不到值不会返回 nil

```
1 h = {"r"=>"ruby"}
2 a = h.fetch("t")
3 p "aaaa"
```

2. bash

```
wangmjcdMacBook-Pro:ruby wangmjc$ ruby string.rb
string.rb:2:in `fetch': key not found: "t" (KeyError)
from string.rb:2:in `<main>'
wangmjcdMacBook-Pro:ruby wangmjc$
```

如果给fetch设置第二个参数,那么该参数数值会作为为空的时候的返回值,但不会改变原来的值

```
1 h = {"r"=>"ruby"}
2 a = h.fetch("t", "c++")
3 p a
4 p h
```

2. bash

```
wangmjcdMacBook-Pro:ruby wangmjc$ ruby string.rb
"c++"
{"r"=>"ruby"}
wangmjcdMacBook-Pro:ruby wangmjc$
```

#### 15.3.1 一次性获取所有的键值

```

[59] pry(main)> h = {a: "b", c: "d"}
=> {:a=>"b", :c=>"d"}
[60] pry(main)> h.keys
=> [:a, :c]
[61] pry(main)> h.values
=> ["b", "d"]
[62] pry(main)> h.to_a
=> [[:a, "b"], [:c, "d"]]

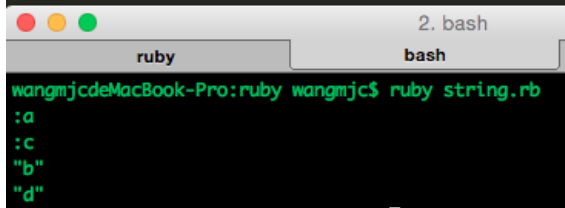
```

除了返回数组外，我们还可以使用迭代器来获取键值  
each\_key 和 each\_value 方法

```

1 h = {a: "b", c: "d"}
2 h.each_key {|i| p i}
3 h.each_value {|i| p i}

```



```

2. bash
ruby
wangmjcdMacBook-Pro:ruby wangmjc$ ruby string.rb
:a
:c
"b"
"d"

```

### 15.3.2 散列的默认值

散列的默认值是为了获取散列中不存在的键值时的返回值，因此不会因此而出错，共有3中方法来设定

1. Hash.new时候传递默认值，但是fetch不生效

```

[68] pry(main)> h = Hash.new(1)
=> {}
[69] pry(main)> h[:a] = 10
=> 10
[70] pry(main)> h[:a]
=> 10
[71] pry(main)> h[:b]
=> 1

```

2.通过Hash.new + 块的方式创建，只有在需要默认值的时候，才会调用的

```

1 h = Hash.new do |hash, key| #这样的方法创建散列后，只有在需要默认值的时候，才会执行块的
2   p "调用"
3   hash[key] = key.upcase
4 end
5 h[:a] = "b"
6 p h[:a]
7 p h[:b]
8 p h["c"]

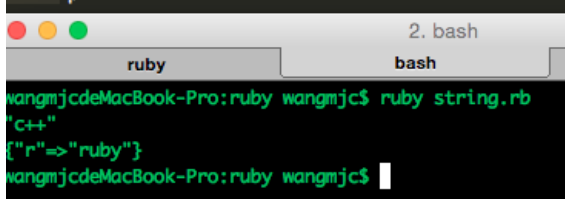
```

3.用fetch方法

```

1 h = {"r"=>"ruby"}
2 a = h.fetch("t", "c++")
3 p a
4 p h

```



```

2. bash
ruby
wangmjcdMacBook-Pro:ruby wangmjc$ ruby string.rb
"c++"
{"r"=>"ruby"}
wangmjcdMacBook-Pro:ruby wangmjc$

```

### 15.4 查看指定对象是否为散列的键或值

1.查看键值

h.key?(key)

h.has\_key(key)

h.include?(key)

h.member?(key)

```

[76] pry(main)> h = {"a"=> "b", "c"=> "d"}
=> {"a"=>"b", "c"=>"d"}
[77] pry(main)> h.key?("a")
=> true
[78] pry(main)> h.key?("z")
=> false
[79] pry(main)> h.has_key?("a")
=> true
[80] pry(main)> h.include?("a")
=> true
[81] pry(main)> h.member?("a")
=> true

```

## 2.查看value

h.value?

h.has\_value?

```
=> {"a"=>"b", "c"=>"d"}
[83] pry(main)> h.value?("b")
=> true
[84] pry(main)> h.has_value?("b")
=> true
```

## 15.5查看散列的大小

```
[86] pry(main)> h
=> {"a"=>"b", "c"=>"d"}
[87] pry(main)> h.size
=> 2
[88] pry(main)> h.length
=> 2
```

查看散列是否为空

```
[89] pry(main)> h.empty?
=> false
```

h.delete(key)删除键值

```
[90] pry(main)> h
=> {"a"=>"b", "c"=>"d"}
[91] pry(main)> h.delete("a")
=> "b"
[92] pry(main)> h
=> {"c"=>"d"}
```

delete和块配合，如果删除的键值没有东西，就返回块的执行结果

```
[99] pry(main)> h = {"a"=>"b", "c"=>"d"}
=> {"a"=>"b", "c"=>"d"}
[100] pry(main)> h.delete("p"){|key| "no #{key}" }
=> "no p"
[101] pry(main)> h
=> {"a"=>"b", "c"=>"d"}
[102] pry(main)> h.delete("a"){|key| "no #{key}" }
=> "b"
[103] pry(main)> h
=> {"c"=>"d"}
```

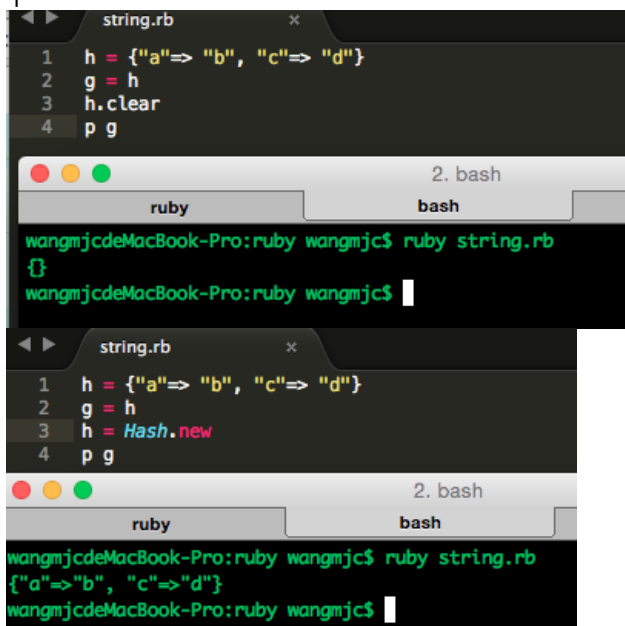
## 15.7 初始化散列

```
[105] pry(main)> h = {"a"=>"b", "c"=>"d"}
=> {"a"=>"b", "c"=>"d"}
[106] pry(main)> h.clear
=> {}
[107] pry(main)> h
=> {}
```

这有点累死将h重新Hash.new一下，

```
[112] pry(main)> h = {"a"=>"b", "c"=>"d"}
=> {"a"=>"b", "c"=>"d"}
[113] pry(main)> h = Hash.new
=> {}
```

实际上如果程序只有一个地方引用h的话，两者的效果是一样的，不过入股还有其他的地方应用h的话，那效果就不一样了，我们来比较一下



第一种情况，h和g都引用的是同一个散列，当h.clear的时候，所以g也空了

第二种情况，h和g都引用的是同一个散列，为h用Hash.new赋予了一个空的散列，但是g还是和老的散列连在一起，没有删除老的散

列，此时 h和g已经引用不同的散列了。

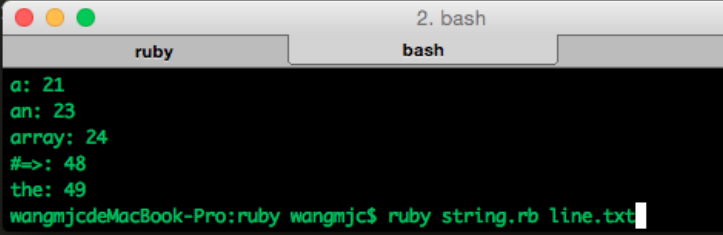
处理有两个键的散列，也就是散列的散列

```
[115] pry(main)> table
=> {"A"=>{"a"=>"x", "b"=>"y"}, "B"=>{"a"=>"v", "b"=>"w"}}
[116] pry(main)> table["A"]["b"]
=> "y"
```

## 15.8应用示例

需求，统计一个文章内每个word的使用次数，从小到大排列

```
1 #初始化单词数量，默认值为0
2 count = Hash.new(0)
3
4 #统计单词
5 File.open(ARGV[0]) do |f|
6   f.each_line do |line|
7     words = line.split
8     words.each do |w|
9       count[w] += 1 #默认为0，所以当有第一个单词的时候为1
10    end
11  end
12 end
13 count.sort{|a, b| a[1] <=> b[1]}.each do |key, value|
14   print "#{key}: #{value}\n"
15 end
16
```



```
2. bash
ruby
bash
a: 21
an: 23
array: 24
#=>: 48
the: 49
wangmjcdeMacBook-Pro:ruby wangmjc$ ruby string.rb line.txt
```

注意：用数值或者自己定义类等对象作为散列的键需要注意的地方

```
[140] pry(main)> h = Hash.new
=> {}
[141] pry(main)> n1 = 1
=> 1
[142] pry(main)> n2 = 1.0
=> 1.0
[143] pry(main)> n1 == n2
=> true
[144] pry(main)> h[n1] = "red"
=> "red"
[145] pry(main)> h[n1]
=> "red"
[146] pry(main)> h[n2]
=> nil
```

