

14章 字符串类

作者 bluetea

网站 <https://github.com/bluetea>

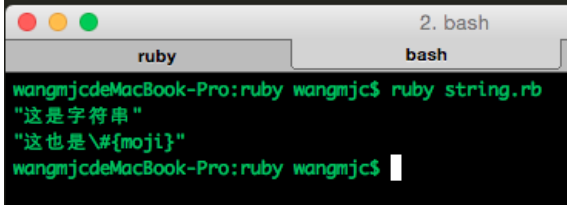
14.1 字符串的创建爱你

“ ” 和 ‘ ’ 的区别

“ ” 内的 #{表达式} 会展开

‘ ’ 内的 #{表达式} 不会展开

```
1 moji = "字符串"
2 str1 = "这是#{moji}"
3 str2 = '这也是#{moji}'
4 p str1
5 p str2
```



```
wangmjcdMacBook-Pro:ruby wangmjc$ ruby string.rb
"这是字符串"
"这也是\\#{moji}"
wangmjcdMacBook-Pro:ruby wangmjc$
```

使用 \ 的转义符 书 189页

14.1.1 使用 %Q 与 %q 的模式创建字符串

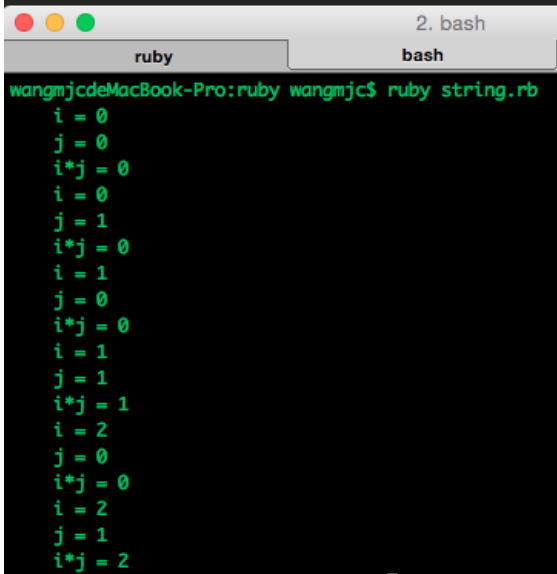
```
aa = "大家好"
str1 = %Q( #{aa} Ruby的字符串可以使用" 和 ' ') #相当于 "" 创建字符串
str2 = %q( #{aa} ruby said . 'hello world') #相当于 '' 创建字符串
p str1
p str2
```



```
wangmjcdMacBook-Pro:ruby wangmjc$ ruby string.rb
" 大家好 Ruby的字符串可以使用\" \"和' '"
" \\#{aa} ruby said . 'hello world'"
wangmjcdMacBook-Pro:ruby wangmjc$
```

14.1.2 使用 Here Document 方式创建字符串

```
1 3.times do |i|
2    2.times do |j|
3      print(<<-"EOB")
4      i = #{i}
5      j = #{j}
6      i*j = #{i*j}
7      EOB
8    end
9  end
```



```
wangmjcdMacBook-Pro:ruby wangmjc$ ruby string.rb
i = 0
j = 0
i*j = 0
i = 0
j = 1
i*j = 0
i = 1
j = 0
i*j = 0
i = 1
j = 1
i*j = 1
i = 2
j = 0
i*j = 0
i = 2
j = 1
i*j = 2
```

14.1.3 使用 `` esc 下面的那个

通过 ` 命令 ` 的形式，我们可以得到命令的标准输出，并将其转换为字符串对象例如

```
[9] pry(main)> puts `ls -l`
total 0
drwx----- 3 wangmjc staff 102 12 17 21:44 Applications
drwx-----+ 8 wangmjc staff 272 1 4 18:04 Desktop
drwx-----+ 7 wangmjc staff 238 12 18 18:24 Documents
drwx-----+ 16 wangmjc staff 544 1 4 17:48 Downloads
drwx-----@ 50 wangmjc staff 1700 1 2 19:17 Library
drwx-----+ 3 wangmjc staff 102 12 17 21:29 Movies
drwx-----+ 5 wangmjc staff 170 1 2 19:17 Music
drwx-----+ 4 wangmjc staff 136 12 20 23:18 Pictures
drwxr-xr-x+ 5 wangmjc staff 170 12 17 21:29 Public
drwxr-xr-x@ 13 wangmjc staff 442 12 30 20:34 快盘
=> nil
```

注意 printf 和sprintf方法

printf是输出到标准输出

sprintf 是输出到字符串

```
[11] pry(main)> a = printf("%d", 22)
22=> nil
[12] pry(main)> a
=> nil
[13] pry(main)> a = sprintf("%d", 22)
=> "22"
[14] pry(main)> a
=> "22"
```

14.2 获取字符串的长度

length 和size方法

```
1 str = "面向对象编程"
2 p str.length #返回字符数
3 p str.size   #返回字符数
4 p str.bytesize #返回字节数
```

```
wangmjcdeMacBook-Pro:ruby wangmjc$ ruby string.rb
6
6
18
```

判断一个字符串是否为空 empty? 函数

```
15] pry(main)> str = "面向对象编程"
> "面向对象编程"
16] pry(main)> str.empty?
> false
```

14.4 字符串的连接

1.将2个字符串合并为新的字符串

```
[20] pry(main)> str1 = "你好"
=> "你好"
[21] pry(main)> str2 = "世界"
=> "世界"
[22] pry(main)> str = str1 +str2
=> "你好世界"
[23] pry(main)> str1
=> "你好"
[24] pry(main)> str2
=> "世界"
```

2.扩展原有的字符串

```
[25] pry(main)> str = str1 << str2
=> "你好世界"
[26] pry(main)> str1
=> "你好世界"
[27] pry(main)> str2
=> "世界"
```

或者用concat方法和 <<效果一样

```
[28] pry(main)> str1
=> "你好世界"
[29] pry(main)> str2
=> "世界"
[30] pry(main)> str1.concat(str2)
=> "你好世界世界"
[31] pry(main)> str1
=> "你好世界世界"
[32] pry(main)> str2
=> "世界"
```

14.5字符串的比较,如果想判断包含或者复杂的,用正则表达式

```
[33] pry(main)> "aa" == "bb"
=> false
[34] pry(main)> "aa" != "bb"
=> true
```

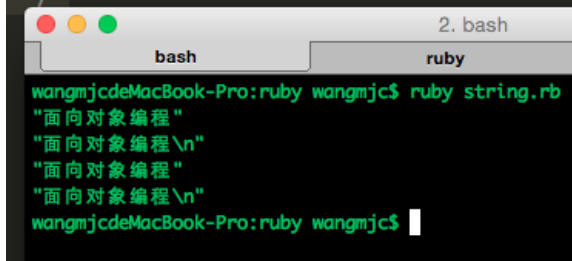
14.6字符串的分割

```
[35] pry(main)> str = "北京:平谷:101200"
=> "北京:平谷:101200"
[36] pry(main)> str.split(":")
=> ["北京", "平谷", "101200"]
```

14.7换行符的使用方法

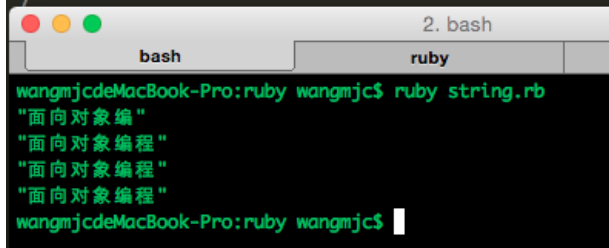
用each_line从标准输入读取字符串时,末尾肯定有换行符,然而在实际处理字符串时,换行符很碍事,我们要删除多余的换行符
非破坏性的 chop(删除最后一个字符) chomp(删除换行符)
破坏性的 chop! chomp!

```
1 str1 = "面向对象编程\n"
2 str2 = "面向对象编程\r\n"
3 p str1.chop
4 p str1
5 p str1.chomp
6 p str1
7
```



注意对比, chop无论如何都会去掉一个最后的字符

```
1 str1 = "面向对象编程"
2 str2 = "面向对象编程"
3 p str1.chop
4 p str1
5 p str1.chomp
6 p str1
7
```



14.8字符串的检索

1.index方法和rindex方法, index从左往右检查字符串是否存在,找到后返回首个字符串的索引

```
[57] pry(main)> str
=> "hello world what are you doing?"
[58] pry(main)> str.rindex("world")
=> 7
```

注意:关于换行符

不同os换行符不同

常见的换行符

1. \n LF模式换行符 LineFeed

2. \r CR模式换行符 Carriage Return

不同操作系统的换行符

Unix \n

window \r\n

Mac 9以前 \r

字符串与数组共同的方法

```
[57] pry(main)> str
=> "helllo world what are you doing?"
[58] pry(main)> str.rindex"world"
=> 7
[59] pry(main)> str.reverse
=> "?gniod uoy era tahw dlrow olleh"
[60] pry(main)> str[1]
=> "e"
```

```
[63] pry(main)> str
=> "helllo world what are you doing?"
[64] pry(main)> str[3..8]
=> "llo wo"
[65] pry(main)> str[3, 5]
=> "llo w"
```

14.9.2 返回Enumerator对象的方法

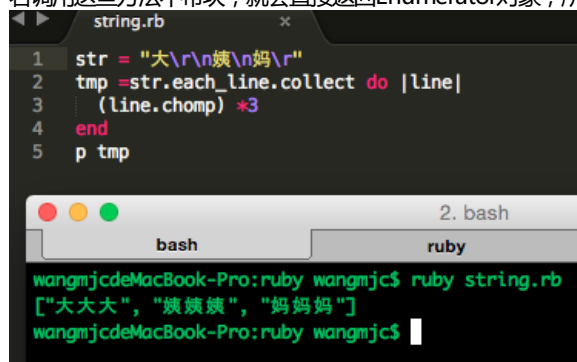
处理字符串的时候

each_line

each_byte

each_char

若调用这些方法不带块，就会直接返回Enumerator对象，所以Enumerator Module模块定义的方法就可以用了



```
1 str = "大\r\n姨\r\n妈\r"
2 tmp =str.each_line.collect do |line|
3   (line.chomp) *3
4 end
5 p tmp
```

2. bash

wangmjcdeMacBook-Pro:ruby wangmjc\$ ruby string.rb

["大大大", "姨姨姨", "妈妈妈"]

wangmjcdeMacBook-Pro:ruby wangmjc\$

string 的这些方法：

each_line

each_byte

each_char

都是由Enumerator这个模块提供的，Enumerator类能以each方法以外的方法为基础，执行Enumerator模块定义的方法，不带快的模块下，大部分Ruby原生的迭代器在调用时都会返回Enumerator对象，正因为如此，我们才能对each_line each_byte等方法的返回结果继续使用collect map方法

```
[5] pry(main)> str = "abcde"
=> "abcde"
[6] pry(main)> a = str.each_byte
=> #<Enumerator: ...>
[7] pry(main)> a.collect{|i| p i}
97
98
99
100
101
=> [97, 98, 99, 100, 101]
```

14.9.3 链接，反转，相关的方法

```
[10] pry(main)> s = "欢迎"
=> "欢迎"
[11] pry(main)> s.concat("光临")
=> "欢迎光临"
[12] pry(main)> s
=> "欢迎光临"
[13] pry(main)> s.delete("光")
=> "欢迎临"
[14] pry(main)> s
=> "欢迎光临"
[15] pry(main)> s.reverse
=> "临光迎欢"
```

删除字符串开头和结尾空格的方法

```
[17] pry(main)> s
=> "  欢迎光临哈哈  "
[18] pry(main)> s.strip
=> "欢迎光临哈哈"
```

转换字符大小写

```
[19] pry(main)> s = "aaaBBB"  
=> "aaaBBB"  
[20] pry(main)> s.swapcase  
=> "AAAbbb"
```

开头字符大写的方法

```
[22] pry(main)> s = "wang manjun"  
=> "wang manjun"  
[23] pry(main)> s.capitalize  
=> "Wang manjun"
```