Ruby 模块（Module） Ruby 数组（Array）

Ruby 字符串（String）

Ruby 中的 String 对象用于存储或操作一个或多个字节的序列。

Ruby 字符串分为单引号字符串（'）和双引号字符串（"），区别在于双引号字符串能够支持更多的转义字符。

单引号字符串

最简单的字符串是单引号字符串，即在单引号内存放字符串：

```
'这是一个 Ruby 程序的字符串'
```

如果您需要在单引号字符串内使用单引号字符，那么需要在单引号字符串使用反斜杠(\)，这样 Ruby 解释器就不会认为这个单引号字符是字符串的终止符号：

```
'Won\'t you read O\'Reilly\'s book?'
```

反斜杠也能转义另一个反斜杠，这样第二个反斜杠本身不会解释为转义字符。

以下是 Ruby 中字符串相关的特性。

双引号字符串

在双引号字符串中我们可以使用 **#{}** 井号和大括号来计算表达式的值：

字符串中嵌入变量：

实例

```
#!/usr/bin/ruby
# -*- coding: UTF-8 -*-
name1 = "Joe"
name2 = "Mary"
puts "你好 #{name1}, #{name2} 在哪?"
```

尝试一下 »

以上实例输出运行输出结果为：

```
你好 Joe, Mary 在哪?
```

字符串中进行数学运算：

实例

```
#!/usr/bin/ruby
# -*- coding: UTF-8 -*-
x, y, z = 12, 36, 72
puts "x 的值为 #{ x }"
puts "x + y 的值为 #{ x + y }"
puts "x + y + z 的平均值为 #{ (x + y + z)/3 }"
```

尝试一下 »

以上实例输出运行输出结果为：

```
x 的值为 12
x + y 的值为 48
x + y + z 的平均值为 40
```

Ruby 中还支持一种采用 %q 和 %Q 来引导的字符串变量，%q 使用的是单引号引用规则，而 %Q 是双引号引用规则，后面再接一个 (! [{ 等等的开始界定符和与 }]) 等等的末尾界定符。

跟在 q 或 Q 后面的字符是分界符.分界符可以是任意一个非字母数字的单字节字符.如:[,{,(,<,!等,字符串会一直读取到发现相匹配的结束符为止.

实例

```
#!/usr/bin/ruby
# -*- coding: UTF-8 -*-
desc1 = %Q{Ruby 的字符串可以使用 '' 和 ""。}
desc2 = %q|Ruby 的字符串可以使用 '' 和 ""。|
puts desc1
puts desc2
```

尝试一下 »

以上实例输出运行输出结果为：

```
Ruby 的字符串可以使用 '' 和 ""。
Ruby 的字符串可以使用 '' 和 ""。
```

转义字符

下标列出了可使用反斜杠符号转义的转义字符或非打印字符。

注意：在一个双引号括起的字符串内，转义字符会被解析。在一个单引号括起的字符串内，转义字符不会被解析，原样输出。

反斜杠符号	十六进制字符	描述
\a	0x07	报警符
\b	0x08	退格键
\cx		Control-x
\C-x		Control-x
\e	0x1b	转义符

\f	0x0c	换页符
\M-\C-x		Meta-Control-x
\n	0x0a	换行符
\nnn		八进制表示法，其中 n 的范围为 0.7
\r	0x0d	回车符
\s	0x20	空格符
\t	0x09	制表符
\v	0x0b	垂直制表符
\x		字符 x
\xnn		十六进制表示法，其中 n 的范围为 0.9、 a.f 或 A.F

字符编码

Ruby 的默认字符集是 ASCII，字符可用单个字节表示。如果您使用 UTF-8 或其他现代的字符集，字符可能是用一个到四个字节表示。

您可以在程序开头使用 $KCODE 改变字符集，如下所示：

```
$KCODE = 'u'
```

下面是 $KCODE 可能的值。

编码	描述
a	ASCII （与 none 相同）。这是默认的。
e	EUC。
n	None （与 ASCII 相同）。
u	UTF-8。

字符串内建方法

我们需要有一个 String 对象的实例来调用 String 方法。下面是创建 String 对象实例的方式：

```
new [String.new(str="")]
```

这将返回一个包含 *str* 副本的新的字符串对象。现在，使用 *str* 对象，我们可以调用任意可用的实例方法。例如：

实例

```
#!/usr/bin/ruby
myStr = String.new("THIS IS TEST")
foo = myStr.downcase
puts "#{foo}"
```

这将产生以下结果：

```
this is test
```

下面是公共的字符串方法（假设 str 是一个 String 对象）：

序号	方法 & 描述
1	**str % arg** 使用格式规范格式化字符串。如果 arg 包含一个以上的替代，那么 arg 必须是一个数组。如需了解更多格式规范的信息，请查看"内核模块"下的 sprintf。
2	**str * integer** 返回一个包含 integer 个 str 的新的字符串。换句话说，str 被重复了 integer 次。
3	**str + other_str** 连接 other_str 到 str。
4	**str << obj** 连接一个对象到字符串。如果对象是范围为 0.255 之间的固定数字 Fixnum，则它会被转换为一个字符。把它与 concat 进行比较。
5	**str <=> other_str** 把 str 与 other_str 进行比较，返回 -1（小于）、0（等于）或 1（大于）。比较是区分大小写的。
6	**str == obj** 检查 str 和 obj 的相等性。如果 obj 不是字符串，则返回 false，如果 str <=> obj，则返回 true，返回 0。
7	**str =~ obj** 根据正则表达式模式 obj 匹配 str。返回匹配开始的位置，否则返回 false。
8	**str[position] # 注意返回的是ASCII码而不是字符** **str[start, length]** **str[start..end]** **str[start...end]** 使用索引截取子串

9	**str.capitalize** 把字符串转换为大写字母显示。
10	**str.capitalize!** 与 capitalize 相同，但是 str 会发生变化并返回。
11	**str.casecmp** 不区分大小写的字符串比较。
12	**str.center** 居中字符串。
13	**str.chomp** 从字符串末尾移除记录分隔符（ $/ ），通常是 \n。如果没有记录分隔符，则不进行任何操作。
14	**str.chomp!** 与 chomp 相同，但是 str 会发生变化并返回。
15	**str.chop** 移除 str 中的最后一个字符。
16	**str.chop!** 与 chop 相同，但是 str 会发生变化并返回。
17	**str.concat(other_str)** 连接 other_str 到 str。
18	**str.count(str, ...)** 给一个或多个字符集计数。如果有多个字符集，则给这些集合的交集计数。
19	**str.crypt(other_str)** 对 str 应用单向加密哈希。参数是两个字符长的字符串，每个字符的范围为 a.z、 A.Z、 0.9、 . 或 /。
20	**str.delete(other_str, ...)** 返回 str 的副本，参数交集中的所有字符会被删除。
21	**str.delete!(other_str, ...)** 与 delete 相同，但是 str 会发生变化并返回。
22	**str.downcase** 返回 str 的副本，所有的大写字母会被替换为小写字母。
23	**str.downcase!**

	与 downcase 相同，但是 str 会发生变化并返回。
24	**str.dump** 返回 str 的版本，所有的非打印字符被替换为 \nnn 符号，所有的特殊字符被转义。
25	**str.each(separator=$/) { |substr| block }** 使用参数作为记录分隔符（默认是 $/）分隔 str，传递每个子字符串给被提供的块。
26	**str.each_byte { |fixnum| block }** 传递 str 的每个字节给 block，以字节的十进制表示法返回每个字节。
27	**str.each_line(separator=$/) { |substr| block }** 使用参数作为记录分隔符（默认是 $/）分隔 str，传递每个子字符串给被提供的 block。
28	**str.empty?** 如果 str 为空（即长度为 0），则返回 true。
29	**str.eql?(other)** 如果两个字符串有相同的长度和内容，则这两个字符串相等。
30	**str.gsub(pattern, replacement) [or]** **str.gsub(pattern) { |match| block }** 返回 str 的副本，pattern 的所有出现都替换为 replacement 或 block 的值。pattern 通常是一个正则表达式 Regexp；如果是一个字符串 String，则没有正则表达式元字符被解释（即，/\d/ 将匹配一个数字，但 '\d' 将匹配一个反斜杠后跟一个 'd'）。
31	**str[fixnum] [or] str[fixnum,fixnum] [or] str[range] [or] str[regexp] [or] str[regexp, fixnum] [or] str[other_str]** 使用下列的参数引用 str：参数为一个 Fixnum，则返回 fixnum 的字符编码；参数为两个 Fixnum，则返回一个从偏移（第一个 fixnum）开始截至到长度（第二个 fixnum）为止的子字符串；参数为 range，则返回该范围内的一个子字符串；参数为 regexp，则返回匹配字符串的部分；参数为带有 fixnum 的 regexp，则返回 fixnum 位置的匹配数据；参数为 other_str，则返回匹配 other_str 的子字符串。一个负数的 Fixnum 从字符串的末尾 -1 开始。
32	**str[fixnum] = fixnum [or] str[fixnum] = new_str [or] str[fixnum, fixnum] = new_str [or] str[range] = aString [or] str[regexp] =new_str [or] str[regexp, fixnum] =new_str [or] str[other_str] = new_str]** 替换整个字符串或部分字符串。与 slice! 同义。
33	**str.gsub!(pattern, replacement) [or] str.gsub!(pattern) { |match| block }** 执行 String#gsub 的替换，返回 str，如果没有替换被执行则返回 nil。
34	**str.hash** 返回一个基于字符串长度和内容的哈希。

35	**str.hex** 把 str 的前导字符当作十六进制数字的字符串（一个可选的符号和一个可选的 0x），并返回相对应的数字。如果错误则返回零。
36	**str.include? other_str [or] str.include? fixnum** 如果 str 包含给定的字符串或字符，则返回 true。
37	**str.index(substring [, offset]) [or]** **str.index(fixnum [, offset]) [or]** **str.index(regexp [, offset])** 返回给定子字符串、字符（fixnum）或模式（regexp）在 str 中第一次出现的索引。如果未找到则返回 nil。如果提供了第二个参数，则指定在字符串中开始搜索的位置。
38	**str.insert(index, other_str)** 在给定索引的字符前插入 other_str，修改 str。负值索引从字符串的末尾开始计数，并在给定字符后插入。其意图是在给定的索引处开始插入一个字符串。
39	**str.inspect** 返回 str 的可打印版本，带有转义的特殊字符。
40	**str.intern [or] str.to_sym** 返回与 str 相对应的符号，如果之前不存在，则创建符号。
41	**str.length** 返回 str 的长度。把它与 size 进行比较。
42	**str.ljust(integer, padstr=' ')** 如果 integer 大于 str 的长度，则返回长度为 integer 的新字符串，新字符串以 str 左对齐，并以 padstr 作为填充。否则，返回 str。
43	**str.lstrip** 返回 str 的副本，移除了前导的空格。
44	**str.lstrip!** 从 str 中移除前导的空格，如果没有变化则返回 nil。
45	**str.match(pattern)** 如果 pattern 不是正则表达式，则把 pattern 转换为正则表达式 Regexp，然后在 str 上调用它的匹配方法。
46	**str.oct** 把 str 的前导字符当作十进制数字的字符串（一个可选的符号），并返回相对应的数字。如果转换失败，则返回 0。

47	**str.replace(other_str)** 把 str 中的内容替换为 other_str 中的相对应的值。
48	**str.reverse** 返回一个新字符串，新字符串是 str 的倒序。
49	**str.reverse!** 逆转 str，str 会发生变化并返回。
50	**str.rindex(substring [, fixnum]) [or]** **str.rindex(fixnum [, fixnum]) [or]** **str.rindex(regexp [, fixnum])** 返回给定子字符串、字符（fixnum）或模式（regexp）在 str 中最后一次出现的索引。如果未找到则返回 nil。如果提供了第二个参数，则指定在字符串中结束搜索的位置。超出该点的字符将不被考虑。
51	**str.rjust(integer, padstr=' ')** 如果 integer 大于 str 的长度，则返回长度为 integer 的新字符串，新字符串以 str 右对齐，并以 padstr 作为填充。否则，返回 str。
52	**str.rstrip** 返回 str 的副本，移除了尾随的空格。
53	**str.rstrip!** 从 str 中移除尾随的空格，如果没有变化则返回 nil。
54	**str.scan(pattern) [or]** **str.scan(pattern) { |match, ...| block }** 两种形式匹配 pattern（可以是一个正则表达式 Regexp 或一个字符串 String）遍历 str。针对每个匹配，会生成一个结果，结果会添加到结果数组中或传递给 block。如果 pattern 不包含分组，则每个独立的结果由匹配的字符串、$& 组成。如果 pattern 包含分组，每个独立的结果是一个包含每个分组入口的数组。
55	**str.slice(fixnum) [or] str.slice(fixnum, fixnum) [or]** **str.slice(range) [or] str.slice(regexp) [or]** **str.slice(regexp, fixnum) [or] str.slice(other_str)** **See str[fixnum], etc.** **str.slice!(fixnum) [or] str.slice!(fixnum, fixnum) [or] str.slice!(range) [or] str.slice!(regexp) [or] str.slice!(other_str)** 从 str 中删除指定的部分，并返回删除的部分。如果值超出范围，参数带有 Fixnum 的形式，将生成一个 IndexError。参数为 range 的形式，将生成一个 RangeError，参数为 Regexp 和 String 的形式，将忽略执行动作。
56	**str.split(pattern=$;, [limit])**

	基于分隔符，把 str 分成子字符串，并返回这些子字符串的数组。 如果 *pattern* 是一个字符串 String，那么在分割 str 时，它将作为分隔符使用。如果 pattern 是一个单一的空格，那么 str 是基于空格进行分割，会忽略前导空格和连续空格字符。 如果 *pattern* 是一个正则表达式 Regexp，则 str 在 pattern 匹配的地方被分割。当 pattern 匹配一个玲长度的字符串时，str 被分割成单个字符。 如果省略了 *pattern* 参数，则使用 $; 的值。如果 $; 为 nil（默认的），str 基于空格进行分割，就像是指定了 ` ` 作为分隔符一样。 如果省略了 *limit* 参数，会抑制尾随的 null 字段。如果 limit 是一个正数，则最多返回该数量的字段（如果 limit 为 1，则返回整个字符串作为数组中的唯一入口）。如果 limit 是一个负数，则返回的字段数量不限制，且不抑制尾随的 null 字段。
57	**str.squeeze([other_str]*)** 使用为 String#count 描述的程序从 other_str 参数建立一系列字符。返回一个新的字符串，其中集合中出现的相同的字符会被替换为单个字符。如果没有给出参数，则所有相同的字符都被替换为单个字符。
58	**str.squeeze!([other_str]*)** 与 squeeze 相同，但是 str 会发生变化并返回，如果没有变化则返回 nil。
59	**str.strip** 返回 str 的副本，移除了前导的空格和尾随的空格。
60	**str.strip!** 从 str 中移除前导的空格和尾随的空格，如果没有变化则返回 nil。
61	**str.sub(pattern, replacement) [or]** **str.sub(pattern) { |match| block }** 返回 str 的副本，pattern 的第一次出现会被替换为 replacement 或 block 的值。pattern 通常是一个正则表达式 Regexp；如果是一个字符串 String，则没有正则表达式元字符被解释。
62	**str.sub!(pattern, replacement) [or]** **str.sub!(pattern) { |match| block }** 执行 String#sub 替换，并返回 str，如果没有替换执行，则返回 nil。
63	**str.succ [or] str.next** 返回 str 的继承。
64	**str.succ! [or] str.next!** 相当于 String#succ，但是 str 会发生变化并返回。
65	**str.sum(n=16)** 返回 str 中字符的 n-bit 校验和，其中 n 是可选的 Fixnum 参数，默认为 16。结果是简单地把 str 中每个字符的二进制

	值的总和，以 2n - 1 为模。这不是一个特别好的校验和。
66	**str.swapcase** 返回 str 的副本，所有的大写字母转换为小写字母，所有的小写字母转换为大写字母。
67	**str.swapcase!** 相当于 String#swapcase，但是 str 会发生变化并返回，如果没有变化则返回 nil。
68	**str.to_f** 返回把 str 中的前导字符解释为浮点数的结果。超出有效数字的末尾的多余字符会被忽略。如果在 str 的开头没有有效数字，则返回 0.0。该方法不会生成异常。
69	**str.to_i(base=10)** 返回把 str 中的前导字符解释为整数基数（基数为 2、 8、 10 或 16）的结果。超出有效数字的末尾的多余字符会被忽略。如果在 str 的开头没有有效数字，则返回 0。该方法不会生成异常。
70	**str.to_s [or] str.to_str** 返回接收的值。
71	**str.tr(from_str, to_str)** 返回 str 的副本，把 from_str 中的字符替换为 to_str 中相对应的字符。如果 to_str 比 from_str 短，那么它会以最后一个字符进行填充。两个字符串都可以使用 c1.c2 符号表示字符的范围。如果 from_str 以 ^ 开头，则表示除了所列出的字符以外的所有字符。
72	**str.tr!(from_str, to_str)** 相当于 String#tr，但是 str 会发生变化并返回，如果没有变化则返回 nil。
73	**str.tr_s(from_str, to_str)** 把 str 按照 String#tr 描述的规则进行处理，然后移除会影响翻译的重复字符。
74	**str.tr_s!(from_str, to_str)** 相当于 String#tr_s，但是 str 会发生变化并返回，如果没有变化则返回 nil。
75	**str.unpack(format)** 根据 format 字符串解码 str（可能包含二进制数据），返回被提取的每个值的数组。format 字符由一系列单字符指令组成。每个指令后可以跟着一个数字，表示重复该指令的次数。星号（*）将使用所有剩余的元素。指令 sSiIlL 每个后可能都跟着一个下划线（_），为指定类型使用底层平台的本地尺寸大小，否则使用独立于平台的一致的尺寸大小。format 字符串中的空格会被忽略。
76	**str.upcase** 返回 str 的副本，所有的小写字母会被替换为大写字母。操作是环境不敏感的，只有字符 a 到 z 会受影响。

77	**str.upcase!** 改变 str 的内容为大写，如果没有变化则返回 nil。
78	**str.upto(other_str) { |s| block }** 遍历连续值，以 str 开始，以 other_str 结束（包含），轮流传递每个值给 block。String#succ 方法用于生成每个值。

字符串 unpack 指令

下表列出了方法 String#unpack 的解压指令。

指令	返回	描述
A	String	移除尾随的 null 和空格。
a	String	字符串。
B	String	从每个字符中提取位（首先是最高有效位）。
b	String	从每个字符中提取位（首先是最低有效位）。
C	Fixnum	提取一个字符作为无符号整数。
c	Fixnum	提取一个字符作为整数。
D, d	Float	把 sizeof(double) 长度的字符当作原生的 double。
E	Float	把 sizeof(double) 长度的字符当作 littleendian 字节顺序的 double。
e	Float	把 sizeof(float) 长度的字符当作 littleendian 字节顺序的 float。
F, f	Float	把 sizeof(float) 长度的字符当作原生的 float。
G	Float	把 sizeof(double) 长度的字符当作 network 字节顺序的 double。
g	Float	把 sizeof(float) 长度的字符当作 network 字节顺序的 float。
H	String	从每个字符中提取十六进制（首先是最高有效位）。
h	String	从每个字符中提取十六进制（首先是最低有效位）。
I	Integer	把 sizeof(int) 长度（通过 _ 修改）的连续字符当作原生的 integer。
i	Integer	把 sizeof(int) 长度（通过 _ 修改）的连续字符当作有符号的原生的 integer。
L	Integer	把四个（通过 _ 修改）连续字符当作无符号的原生的 long integer。
l	Integer	把四个（通过 _ 修改）连续字符当作有符号的原生的 long integer。

M	String	引用可打印的。
m	String	Base64 编码。
N	Integer	把四个字符当作 network 字节顺序的无符号的 long。
n	Fixnum	把两个字符当作 network 字节顺序的无符号的 short。
P	String	把 sizeof(char *) 长度的字符当作指针，并从引用的位置返回 \emph{len} 字符。
p	String	把 sizeof(char *) 长度的字符当作一个空结束字符的指针。
Q	Integer	把八个字符当作无符号的 quad word（64 位）。
q	Integer	把八个字符当作有符号的 quad word（64 位）。
S	Fixnum	把两个（如果使用 _ 则不同）连续字符当作 native 字节顺序的无符号的 short。
s	Fixnum	把两个（如果使用 _ 则不同）连续字符当作 native 字节顺序的有符号的 short。
U	Integer	UTF-8 字符，作为无符号整数。
u	String	UU 编码。
V	Fixnum	把四个字符当作 little-endian 字节顺序的无符号的 long。
v	Fixnum	把两个字符当作 little-endian 字节顺序的无符号的 short。
w	Integer	BER 压缩的整数。
X		向后跳过一个字符。
x		向前跳过一个字符。
Z	String	和 * 一起使用，移除尾随的 null 直到第一个 null。
@		跳过 length 参数给定的偏移量。

实例

尝试下面的实例，解压各种数据。

```
"abc \0\0abc \0\0".unpack('A6Z6') #=> ["abc", "abc "]
"abc \0\0".unpack('a3a3') #=> ["abc", " \000\000"]
"abc \0abc \0".unpack('Z*Z*') #=> ["abc ", "abc "]
"aa".unpack('b8B8') #=> ["10000110", "01100001"]
"aaa".unpack('h2H2c') #=> ["16", "61", 97]
"\xfe\xff\xfe\xff".unpack('sS') #=> [-2, 65534]
```

```
"now=20is".unpack('M*') #=> ["now is"]
"whole".unpack('xax2aX2aX1aX2a') #=> ["h", "e", "l", "l", "o"]
```

Ruby 模块（Module）

Ruby 数组（Array）

点我分享笔记