Ruby 字符串（ String ） Ruby 哈希（ Hash ）

Ruby 数组（ Array ）

Ruby 数组是任何对象的有序整数索引集合。数组中的每个元素都与一个索引相关，并可通过索引进行获取。

数组的索引从 0 开始，这与 C 或 Java 中一样。一个负数的索相对于数组的末尾计数的，也就是说，索引为 -1 表示数组的最后一个元素，-2 表示数组中的倒数第二个元素，依此类推。

Ruby 数组可存储诸如 String、 Integer、 Fixnum、 Hash、 Symbol 等对象，甚至可以是其他 Array 对象。

Ruby 数组不需要指定大小，当向数组添加元素时，Ruby 数组会自动增长。

创建数组

有多种方式创建或初始化数组。一种方式是通过 *new* 类方法：

```
names = Array.new
```

您可以在创建数组的同时设置数组的大小：

```
names = Array.new(20)
```

数组 *names* 的大小或长度为 20 个元素。您可以使用 size 或 length 方法返回数组的大小：

实例

```
#!/usr/bin/ruby
names = Array.new(20)
puts names.size # 返回 20
puts names.length # 返回 20
```

尝试一下 »

以上实例运行输出结果为：

```
20
20
```

您可以给数组中的每个元素赋值，如下所示：

实例

```
#!/usr/bin/ruby
names = Array.new(4, "mac")
puts "#{names}"
```

尝试一下 »

以上实例运行输出结果为：

```
["mac", "mac", "mac", "mac"]
```

您也可以使用带有 new 的块，每个元素使用块中的计算结果来填充：

实例

```
#!/usr/bin/ruby
nums = Array.new(10) { |e| e = e * 2 }
puts "#{nums}"
```

尝试一下 »

以上实例运行输出结果为：

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

数组还有另一种方法，[]，如下所示：

```
nums = Array.[](1, 2, 3, 4,5)
```

数组创建的另一种形式如下所示：

```
nums = Array[1, 2, 3, 4,5]
```

在 Ruby 核心模块中可以有一个只接收单个参数的 Array 方法，该方法使用一个范围作为参数来创建一个数字数组：

实例

```
#!/usr/bin/ruby
digits = Array(0..9)
puts "#{digits}"
```

以上实例运行输出结果为：

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

数组内建方法

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

```
Array.[](...) [or] Array[...] [or] [...]
```

这将返回一个使用给定对象进行填充的新数组。现在，使用创建的对象，我们可以调用任意可用的方法。例如：

实例

```
#!/usr/bin/ruby
digits = Array(0..9)
num = digits.at(6)
puts "#{num}"
```

以上实例运行输出结果为：

```
6
```

下面是公共的数组方法（假设 *array* 是一个 Array 对象）：

序号	方法 & 描述
1	**array & other_array** 返回一个新的数组，包含两个数组中共同的元素，没有重复。
2	**array * int [or] array * str** 返回一个新的数组，新数组通过连接 self 的 int 副本创建的。带有 String 参数时，相当于 self.join(str)。
3	**array + other_array** 返回一个新的数组，新数组通过连接两个数组产生第三个数组创建的。
4	**array - other_array** 返回一个新的数组，新数组是从初始数组中移除了在 other_array 中出现的项的副本。
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	**array | other_array** 通过把 other_array 加入 array 中，移除重复项，返回一个新的数组。
7	**array << obj** 把给定的对象附加到数组的末尾。该表达式返回数组本身，所以几个附加可以连在一起。
8	**array <=> other_array** 如果数组小于、等于或大于 other_array，则返回一个整数（-1、 0 或 +1）。
9	**array == other_array** 如果两个数组包含相同的元素个数，且每个元素与另一个数组中相对应的元素相等（根据 Object.==），那么这两个数组相等。
10	**array[index] [or] array[start, length] [or]** **array[range] [or] array.slice(index) [or]** **array.slice(start, length) [or] array.slice(range)** 返回索引为 *index* 的元素，或者返回从 *start* 开始直至 *length* 个元素的子数组，或者返回 *range* 指定的子数组。负值索引从数组末尾开始计数（-1 是最后一个元素）。如果 *index*（或开始索引）超出范围，则返回 *nil*。
11	**array[index] = obj [or]**

	array[start, length] = obj or an_array or nil [or] **array[range] = obj or an_array or nil** 设置索引为 *index* 的元素，或者替换从 *start* 开始直至 *length* 个元素的子数组，或者替换 *range* 指定的子数组。如果索引大于数组的当前容量，那么数组会自动增长。负值索引从数组末尾开始计数。如果 *length* 为零则插入元素。如果在第二种或第三种形式中使用了 *nil*，则从 *self* 删除元素。
12	**array.abbrev(pattern = nil)** 为 *self* 中的字符串计算明确的缩写集合。如果传递一个模式或一个字符串，只考虑当字符串匹配模式或者以该字符串开始时的情况。
13	**array.assoc(obj)** 搜索一个数组，其元素也是数组，使用 obj.== 把 obj 与每个包含的数组的第一个元素进行比较。如果匹配则返回第一个包含的数组，如果未找到匹配则返回 *nil*。
14	**array.at(index)** 返回索引为 index 的元素。一个负值索引从 *self* 的末尾开始计数。如果索引超出范围则返回 nil。
15	**array.clear** 从数组中移除所有的元素。
16	**array.collect { |item| block } [or]** **array.map { |item| block }** 为 *self* 中的每个元素调用一次 *block*。创建一个新的数组，包含 block 返回的值。
17	**array.collect! { |item| block } [or]** **array.map! { |item| block }** 为 *self* 中的每个元素调用一次 *block*，把元素替换为 *block* 返回的值。
18	**array.compact** 返回 *self* 的副本，移除了所有的 *nil* 元素。
19	**array.compact!** 从数组中移除所有的 *nil* 元素。如果没有变化则返回 *nil*。
20	**array.concat(other_array)** 追加 other_array 中的元素到 *self* 中。
21	**array.delete(obj) [or]** **array.delete(obj) { block }** 从 *self* 中删除等于 *obj* 的项。如果未找到相等项，则返回 *nil*。如果未找到相等项且给出了可选的代码 *block*，则返回 *block* 的结果。

22	**array.delete_at(index)** 删除指定的 *index* 处的元素，并返回该元素。如果 index 超出范围，则返回 *nil*。
23	**array.delete_if { |item| block }** 当 *block* 为 true 时，删除 *self* 的每个元素。
24	**array.each { |item| block }** 为 *self* 中的每个元素调用一次 *block*，传递该元素作为参数。
25	**array.each_index { |index| block }** 与 Array#each 相同，但是传递元素的 *index*，而不是传递元素本身。
26	**array.empty?** 如果数组本身没有包含元素，则返回 true。
27	**array.eql?(other)** 如果 *array* 和 *other* 是相同的对象，或者两个数组带有相同的内容，则返回 true。
28	**array.fetch(index) [or]** **array.fetch(index, default) [or]** **array.fetch(index) { |index| block }** 尝试返回位置 *index* 处的元素。如果 *index* 位于数组外部，则第一种形式会抛出 *IndexError* 异常，第二种形式会返回 *default*，第三种形式会返回调用 *block* 传入 *index* 的值。负值的 *index* 从数组末尾开始计数。
29	**array.fill(obj) [or]** **array.fill(obj, start [, length]) [or]** **array.fill(obj, range) [or]** **array.fill { |index| block } [or]** **array.fill(start [, length]) { |index| block } [or]** **array.fill(range) { |index| block }** 前面三种形式设置 *self* 的被选元素为 *obj*。以 *nil* 开头相当于零。*nil* 的长度相当于 *self.length*。最后三种形式用 block 的值*填充*数组。*block* 通过带有被填充的每个元素的绝对索引来传递。
30	**array.first [or]** **array.first(n)** 返回数组的第一个元素或前 *n* 个元素。如果数组为空，则第一种形式返回 *nil*，第二种形式返回一个空的数组。
31	**array.flatten** 返回一个新的数组，新数组是一个一维的扁平化的数组（递归）。
32	**array.flatten!**

	把 *array* 进行扁平化。如果没有变化则返回 *nil*。（数组不包含子数组。）
33	**array.frozen?** 如果 *array* 被冻结（或排序时暂时冻结），则返回 true。
34	**array.hash** 计算数组的哈希代码。两个具有相同内容的数组将具有相同的哈希代码。
35	**array.include?(obj)** 如果 *self* 中包含 *obj*，则返回 true，否则返回 false。
36	**array.index(obj)** 返回 *self* 中第一个等于 obj 的对象的 *index*。如果未找到匹配则返回 *nil*。
37	**array.indexes(i1, i2, ... iN) [or]** **array.indices(i1, i2, ... iN)** 该方法在 Ruby 的最新版本中被废弃，所以请使用 Array#values_at。
38	**array.indices(i1, i2, ... iN) [or]** **array.indexes(i1, i2, ... iN)** 该方法在 Ruby 的最新版本中被废弃，所以请使用 Array#values_at。
39	**array.insert(index, obj...)** 在给定的 *index* 的元素前插入给定的值，index 可以是负值。
40	**array.inspect** 创建一个数组的可打印版本。
41	**array.join(sep=$,)** 返回一个字符串，通过把数组的每个元素转换为字符串，并使用 *sep* 分隔进行创建的。
42	**array.last [or] array.last(n)** 返回 *self* 的最后一个元素。如果数组为空，则第一种形式返回 *nil*。
43	**array.length** 返回 *self* 中元素的个数。可能为零。
44	**array.map { |item| block } [or]** **array.collect { |item| block }** 为 *self* 的每个元素调用一次 *block*。创建一个新的数组，包含 block 返回的值。
45	**array.map! { |item| block } [or]** **array.collect! { |item| block }**

	为 *array* 的每个元素调用一次 *block*，把元素替换为 block 返回的值。
46	**array.nitems** 返回 *self* 中 non-nil 元素的个数。可能为零。
47	**array.pack(aTemplateString)** 根据 aTemplateString 中的指令，把数组的内容压缩为二进制序列。指令 A、 a 和 Z 后可以跟一个表示结果字段宽度的数字。剩余的指令也可以带有一个表示要转换的数组元素个数的数字。如果数字是一个星号（*），则所有剩余的数组元素都将被转换。任何指令后都可以跟一个下划线（_），表示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。在模板字符串中空格会被忽略。
48	**array.pop** 从 *array* 中移除最后一个元素，并返回该元素。如果 *array* 为空则返回 *nil*。
49	**array.push(obj, ...)** 把给定的 obj 附加到数组的末尾。该表达式返回数组本身，所以几个附加可以连在一起。
50	**array.rassoc(key)** 搜索一个数组，其元素也是数组，使用 == 把 *key* 与每个包含的数组的第二个元素进行比较。如果匹配则返回第一个包含的数组。
51	**array.reject { |item| block }** 返回一个新的数组，包含当 block 不为 true 时的数组项。
52	**array.reject! { |item| block }** 当 block 为真时，从 *array* 删除元素，如果没有变化则返回 *nil*。相当于 Array#delete_if。
53	**array.replace(other_array)** 把 *array* 的内容替换为 *other_array* 的内容，必要的时候进行截断或扩充。
54	**array.reverse** 返回一个新的数组，包含倒序排列的数组元素。
55	**array.reverse!** 把 *array* 进行逆转。
56	**array.reverse_each {|item| block }** 与 Array#each 相同，但是把 *array* 进行逆转。
57	**array.rindex(obj)** 返回 array 中最后一个等于 obj 的对象的索引。如果未找到匹配，则返回 *nil*。
58	**array.select {|item| block }**

	调用从数组传入连续元素的 block，返回一个数组，包含 block 返回 *true* 值时的元素。
59	**array.shift** 返回 *self* 的第一个元素，并移除该元素（把所有的其他元素下移一位）。如果数组为空，则返回 *nil*。
60	**array.size** 返回 *array* 的长度（元素的个数）。length 的别名。
61	**array.slice(index) [or] array.slice(start, length) [or]** **array.slice(range) [or] array[index] [or]** **array[start, length] [or] array[range]** 返回索引为 *index* 的元素，或者返回从 *start* 开始直至 *length* 个元素的子数组，或者返回 *range* 指定的子数组。负值索引从数组末尾开始计数（-1 是最后一个元素）。如果 *index*（或开始索引）超出范围，则返回 *nil*。
62	**array.slice!(index) [or] array.slice!(start, length) [or]** **array.slice!(range)** 删除 *index*（长度是可选的）或 *range* 指定的元素。返回被删除的对象、子数组，如果 *index* 超出范围，则返回 *nil*。
63	**array.sort [or] array.sort { | a,b | block }** 返回一个排序的数组。
64	**array.sort! [or] array.sort! { | a,b | block }** 把数组进行排序。
65	**array.to_a** 返回 *self*。如果在 *Array* 的子类上调用，则把接收参数转换为一个 Array 对象。
66	**array.to_ary** 返回 self。
67	**array.to_s** 返回 self.join。
68	**array.transpose** 假设 self 是数组的数组，且置换行和列。
69	**array.uniq** 返回一个新的数组，移除了 *array* 中的重复值。
70	**array.uniq!** 从 *self* 中移除重复元素。如果没有变化（也就是说，未找到重复），则返回 *nil*。
71	**array.unshift(obj, ...)**

	把对象前置在数组的前面，其他元素上移一位。
72	**array.values_at(selector,...)** 返回一个数组，包含 self 中与给定的 *selector*（一个或多个）相对应的元素。选择器可以是整数索引或者范围。
73	**array.zip(arg, ...) [or]** **array.zip(arg, ...){ | arr | block }** 把任何参数转换为数组，然后把 *array* 的元素与每个参数中相对应的元素合并。

数组 pack 指令

下表列出了方法 Array#pack 的压缩指令。

指令	描述
@	移动到绝对位置。
A	ASCII 字符串（填充 space，count 是宽度）。
a	ASCII 字符串（填充 null，count 是宽度）。
B	位字符串（降序）
b	位字符串（升序）。
C	无符号字符。
c	字符。
D, d	双精度浮点数，原生格式。
E	双精度浮点数，little-endian 字节顺序。
e	单精度浮点数，little-endian 字节顺序。
F, f	单精度浮点数，原生格式。
G	双精度浮点数，network（big-endian）字节顺序。
g	单精度浮点数，network（big-endian）字节顺序。
H	十六进制字符串（高位优先）。
h	十六进制字符串（低位优先）。
I	无符号整数。

i	整数。
L	无符号 long。
l	Long。
M	引用可打印的，MIME 编码。
m	Base64 编码字符串。
N	Long，network（big-endian）字节顺序。
n	Short，network（big-endian）字节顺序。
P	指向一个结构（固定长度的字符串）。
p	指向一个空结束字符串。
Q, q	64 位数字。
S	无符号 short。
s	Short。
U	UTF-8。
u	UU 编码字符串。
V	Long，little-endian 字节顺序。
v	Short，little-endian 字节顺序。
w	BER 压缩的整数 \fnm。
X	向后跳过一个字节。
x	Null 字节。
Z	与 a 相同，除了 null 会被加上 *。

实例

尝试下面的实例，压缩各种数据。

实例

```
a = [ "a", "b", "c" ]
n = [ 65, 66, 67 ]
puts a.pack("A3A3A3") #=> "a  b  c  "
```

```
puts a.pack("a3a3a3") #=> "a\000\000b\000\000c\000\000"
puts n.pack("ccc") #=> "ABC"
```

以上实例运行输出结果为：

```
a  b  c
abc
ABC
```

Ruby 字符串（String）

Ruby 哈希（Hash）

点我分享笔记