Ruby 数组（ Array ） Ruby 日期 & 时间（ Date & Time ）

Ruby 哈希（ Hash ）

哈希（ Hash ）是类似 "key" => "value" 这样的键值对集合。哈希类似于一个数组，只不过它的索引不局限于使用数字。

Hash 的索引（或者叫"键"）几乎可以是任何对象。

Hash 虽然和数组类似，但却有一个很重要的区别：Hash 的元素没有特定的顺序。 如果顺序很重要的话就要使用数组了。

创建哈希

与数组一样，有各种不同的方式来创建哈希。您可以通过 *new* 类方法创建一个空的哈希：

```
months = Hash.new
```

您也可以使用 *new* 创建带有默认值的哈希，不带默认值的哈希是 *nil*：

```
months = Hash.new( "month" )
或
months = Hash.new "month"
```

当您访问带有默认值的哈希中的任意键时，如果键或值不存在，访问哈希将返回默认值：

实例

```
#!/usr/bin/ruby
months = Hash.new( "month" )
puts "#{months[0]}"
puts "#{months[72]}"
```

尝试一下 »

以上实例运行输出结果为：

```
month
month
```

实例

```
#!/usr/bin/ruby
H = Hash["a" => 100, "b" => 200]
puts "#{H['a']}"
puts "#{H['b']}"
```

尝试一下 »

以上实例运行输出结果为：

```
100
200
```

您可以使用任何的 Ruby 对象作为键或值，甚至可以使用数组，如下实例所示：

```
[1,"jan"] => "January"
```

哈希内置方法

如果需要调用 Hash 方法，需要先实例化一个 Hash 对象。下面是创建 Hash 对象实例的方式：

```
Hash[[key =>|, value]* ] or
Hash.new [or] Hash.new(obj) [or]
Hash.new { |hash, key| block }
```

这将返回一个使用给定对象进行填充的新的哈希。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
$, = ", "
months = Hash.new( "month" )
months = {"1" => "January", "2" => "February"}
keys = months.keys
puts "#{keys}"
```

尝试一下 »

以上实例运行输出结果为：

```
["1", "2"]
```

下面是公共的哈希方法（假设 *hash* 是一个 Hash 对象）：

序号	方法 & 描述
1	**hash == other_hash** 检查两个哈希是否具有相同的键值对个数，键值对是否相互匹配，来判断两个哈希是否相等。
2	**hash.[key]** 使用键，从哈希引用值。如果未找到键，则返回默认值。
3	**hash.[key]=value** 把 *value* 给定的值与 *key* 给定的键进行关联。
4	**hash.clear** 从哈希中移除所有的键值对。

5	**hash.default(key = nil)** 返回 *hash* 的默认值，如果未通过 default= 进行设置，则返回 nil。（如果键在 *hash* 中不存在，则 [] 返回一个默认值。）
6	**hash.default = obj** 为 *hash* 设置默认值。
7	**hash.default_proc** 如果 *hash* 通过块来创建，则返回块。
8	**hash.delete(key) [or]** **array.delete(key) { |key| block }** 通过 *key* 从 *hash* 中删除键值对。如果使用了块 且未找到匹配的键值对，则返回块的结果。把它与 *delete_if* 进行比较。
9	**hash.delete_if { |key,value| block }** block 为 *true* 的每个块，从 *hash* 中删除键值对。
10	**hash.each { |key,value| block }** 遍历 *hash*，为每个 *key* 调用一次 block，传递 key-value 作为一个二元素数组。
11	**hash.each_key { |key| block }** 遍历 *hash*，为每个 *key* 调用一次 block，传递 *key* 作为参数。
12	**hash.each_key { |key_value_array| block }** 遍历 *hash*，为每个 *key* 调用一次 block，传递 *key* 和 *value* 作为参数。
13	**hash.each_value { |value| block }** 遍历 *hash*，为每个 *key* 调用一次 block，传递 *value* 作为参数。
14	**hash.empty?** 检查 hash 是否为空（不包含键值对），返回 *true* 或 *false*。
15	**hash.fetch(key [, default]) [or]** **hash.fetch(key) { | key | block }** 通过给定的 *key* 从 *hash* 返回值。如果未找到 *key*，且未提供其他参数，则抛出 *IndexError* 异常；如果给出了 *default*，则返回 *default*；如果指定了可选的 block，则返回 block 的结果。
16	**hash.has_key?(key) [or] hash.include?(key) [or]** **hash.key?(key) [or] hash.member?(key)** 检查给定的 *key* 是否存在于哈希中，返回 *true* 或 *false*。

17	**hash.has_value?(value)** 检查哈希是否包含给定的 *value*。
18	**hash.index(value)** 为给定的 *value* 返回哈希中的 *key*，如果未找到匹配值则返回 *nil*。
19	**hash.indexes(keys)** 返回一个新的数组，由给定的键的值组成。找不到的键将插入默认值。该方法已被废弃，请使用 select。
20	**hash.indices(keys)** 返回一个新的数组，由给定的键的值组成。找不到的键将插入默认值。该方法已被废弃，请使用 select。
21	**hash.inspect** 返回哈希的打印字符串版本。
22	**hash.invert** 创建一个新的 *hash*，倒置 *hash* 中的 *keys* 和 *values*。也就是说，在新的哈希中，*hash* 中的键将变成值，值将变成键。
23	**hash.keys** 创建一个新的数组，带有 *hash* 中的键。/td>
24	**hash.length** 以整数形式返回 *hash* 的大小或长度。
25	**hash.merge(other_hash) [or]** **hash.merge(other_hash) { |key, oldval, newval| block }** 返回一个新的哈希，包含 *hash* 和 *other_hash* 的内容，重写 hash 中与 *other_hash* 带有重复键的键值对。
26	**hash.merge!(other_hash) [or]** **hash.merge!(other_hash) { |key, oldval, newval| block }** 与 merge 相同，但实际上 hash 发生了变化。
27	**hash.rehash** 基于每个 *key* 的当前值重新建立 *hash*。如果插入后值发生了改变，该方法会重新索引 *hash*。
28	**hash.reject { |key, value| block }** 类似 delete_if, 但作用在一个拷贝的哈希上。相等于 hsh.dup.delete_if。
29	**hash.reject! { |key, value| block }** 相等于 delete_if, 但是如果没有修改，返回 nil。
30	**hash.replace(other_hash)**

	把 *hash* 的内容替换为 *other_hash* 的内容。
31	**hash.select { |key, value| block }** 返回一个新的数组，由 *block* 返回 *true* 的 *hash* 中的键值对组成。
32	**hash.shift** 从 *hash* 中移除一个键值对，并把该键值对作为二元素数组返回。
33	**hash.size** 以整数形式返回 *hash* 的 *size* 或 length。
34	**hash.sort** 把 *hash* 转换为一个包含键值对数组的二维数组，然后进行排序。
35	**hash.store(key, value)** 存储 *hash* 中的一个键值对。
36	**hash.to_a** 从 hash 中创建一个二维数组。每个键值对转换为一个数组，所有这些数组都存储在一个数组中。
37	**hash.to_hash** 返回 *hash*（self）。
38	**hash.to_s** 把 *hash* 转换为一个数组，然后把该数组转换为一个字符串。
39	**hash.update(other_hash) [or]** **hash.update(other_hash) {|key, oldval, newval| block}** 返回一个新的哈希，包含 *hash* 和 *other_hash* 的内容，重写 *hash* 中与 *other_hash* 带有重复键的键值对。
40	**hash.value?(value)** 检查 *hash* 是否包含给定的 *value*。
41	**hash.values** 返回一个新的数组，包含 *hash* 的所有值。
42	**hash.values_at(obj, ...)** 返回一个新的数组，包含 *hash* 中与给定的键相关的值。

← Ruby 数组（Array） Ruby 日期 & 时间（Date & Time） →

点我分享笔记