注：本文档为Underscore.js (1.8.2) 中文文档

最新版 Underscore.js (1.8.3) 中文文档请点击:<http://www.css88.com/doc/underscore/>

查看1.7.0版本的文档请点击:[http://www.css88.com/doc/underscore1.7.0/](http://www.css88.com/doc/underscore1.7.0)   
查看1.6.0版本的文档请点击:[http://www.css88.com/doc/underscore1.6.0/](http://www.css88.com/doc/underscore1.6.0)   
查看1.5.2版本的文档请点击:[http://www.css88.com/doc/underscore1.5.2/](http://www.css88.com/doc/underscore1.5.2)

其他前端相关文档：[jQuery API中文文档](http://www.css88.com/jqapi-1.9/) 、[jQuery UI API中文文档](http://www.css88.com/jquery-ui-api/) 、[Zepto.js API 中文版](http://www.css88.com/doc/zeptojs_api/)

[Underscore](http://github.com/jashkenas/underscore/)一个JavaScript实用库，提供了一整套函数式编程的实用功能，但是没有扩展任何JavaScript内置对象。它是这个问题的答案：“如果我在一个空白的HTML页面前坐下， 并希望立即开始工作， 我需要什么？“...它弥补了部分[jQuery](http://docs.jquery.com/)没有实现的功能,同时又是[Backbone.js](http://backbonejs.org/)必不可少的部分。 （感谢[@小邓子daj](http://weibo.com/xiaodengzidaj)的翻译建议）

Underscore提供了100多个函数,包括常用的: **map**, **filter**, **invoke** — 当然还有更多专业的辅助函数,如:函数绑定, JavaScript模板功能,创建快速索引, 强类型相等测试, 等等.

为了你能仔细研读，这里包含了一个完整的[测试套件](https://www.css88.com/doc/underscore1.8.2/test/)。

您也可以通过[注释阅读源代码](https://www.css88.com/doc/underscore1.8.2/docs/underscore.html)。

享受Underscore，并希望获得更多的使用功能（感谢[@Jaward华仔](http://weibo.com/bananajaward) 的翻译建议），可以尝试使用[Underscore-contrib](http://documentcloud.github.io/underscore-contrib/)（愚人码头注：Underscore-contrib是一个Underscore的代码贡献库）。

该项目代码[托管在GitHub](http://github.com/documentcloud/underscore/)上，你可以通过[issues页](http://github.com/documentcloud/underscore/issues)、Freenode的 #documentcloud 频道、发送tweets给[@documentcloud](http://twitter.com/documentcloud)三个途径报告bug以及参与特性讨论。

*Underscore是*[*DocumentCloud*](http://documentcloud.org/)*的一个开源组件。*

**下载 *(右键另存为)***

|  |  |
| --- | --- |
| [开发版 (1.8.2)](https://www.css88.com/doc/underscore1.8.2/underscore.js) | *51kb, 未压缩版, 含大量注释* |
| [生产版 (1.8.2)](https://www.css88.com/doc/underscore1.8.2/underscore-min.js) | *5.7kb, 最简化并用Gzip压缩*  ([Source Map](https://www.css88.com/doc/underscore1.8.2/underscore-min.map)) |
|  | |
| [不稳定版](https://raw.github.com/jashkenas/underscore/master/underscore.js) | *未发布版本, 当前开发中的*master*分支, 如果使用此版本, 风险自负* |

**安装**

**Node.js** npm install underscore

**Meteor.js** meteor add underscore

**Require.js** require(["underscore"], ...

**Bower** bower install underscore

**Component** component install jashkenas/underscore

**集合函数 (数组 或对象)**

### each

.each(list, iteratee, [context]) *Alias:****forEach***   
遍历**list**中的所有元素，按顺序用遍历输出每个元素。如果传递了**context**参数，则把**iteratee**绑定到**context**对象上。每次调用**iteratee**都会传递三个参数：(element, index, list)。如果**list**是个JavaScript对象，**iteratee**的参数是 (value, key, list))。返回**list**以方便链式调用。（愚人码头注：如果存在原生的**forEach**方法，Underscore就使用它代替。）

\_.each([1, 2, 3], alert);

=> alerts each number in turn...

\_.each({one: 1, two: 2, three: 3}, alert);

=> alerts each number value in turn...

*注意：集合函数能在数组，对象，和类数组对象，比如*arguments, NodeList*和类似的数据类型上正常工作。 但是它通过*[*鸭子类型*](http://zh.wikipedia.org/zh-cn/%E9%B8%AD%E5%AD%90%E7%B1%BB%E5%9E%8B)*工作，所以要避免传递一个不固定*length*属性的对象（愚人码头注：对象或数组的长度（length）属性要固定的）。每个循环不能被破坏 - 打破， 使用****\_.find****代替，这也是很好的注意。*

### Map

\_.map(list, iteratee, [context]) *Alias:****collect***   
通过转换函数(**iteratee**迭代器)映射列表中的每个值产生价值的新数组。iteratee传递三个参数：value，然后是迭代 index(或 key 愚人码头注：如果**list**是个JavaScript对象是，这个参数就是key)，最后一个是引用指向整个list。

\_.map([1, 2, 3], function(num){ return num \* 3; });

=> [3, 6, 9]

\_.map({one: 1, two: 2, three: 3}, function(num, key){ return num \* 3; });

=> [3, 6, 9]

\_.map([[1, 2], [3, 4]], \_.first);

=> [1, 3]

### Reduce

\_.reduce(list, iteratee, [memo], [context]) *Aliases:****inject, foldl***   
别名为 **inject** 和 **foldl**, **reduce**方法把**list**中元素归结为一个单独的数值。**Memo**是reduce函数的初始值，reduce的每一步都需要由**iteratee**返回。这个迭代传递4个参数：memo,value 和 迭代的index（或者 key）和最后一个引用的整个 list。

如果没有**memo**传递给**reduce**的初始调用，**iteratee**不会被列表中的第一个元素调用。第一个元素将取代 传递给列表中下一个元素调用**iteratee**的**memo**参数。

var sum = \_.reduce([1, 2, 3], function(memo, num){ return memo + num; }, 0);

=> 6

### reduceRight

\_.reduceRight(list, iteratee, memo, [context]) *Alias:****foldr***   
**reducRight**是从右侧开始组合的元素的**reduce**函数，如果存在JavaScript 1.8版本的**reduceRight**，则用其代替。**Foldr**在javascript中不像其它有懒计算的语言那么有用（愚人码头注：lazy evaluation：一种求值策略，只有当表达式的值真正需要时才对表达式进行计算）。

var list = [[0, 1], [2, 3], [4, 5]];

var flat = \_.reduceRight(list, function(a, b) { return a.concat(b); }, []);

=> [4, 5, 2, 3, 0, 1]

### Find

\_.find(list, predicate, [context]) *Alias:****detect***   
在**list**中逐项查找，返回第一个通过**predicate**迭代函数真值检测的元素值，如果没有值传递给测试迭代器将返回undefined。 如果找到匹配的元素，函数将立即返回，不会遍历整个list。

var even = \_.find([1, 2, 3, 4, 5, 6], function(num){ return num % 2 == 0; });

=> 2

### Filter

\_.filter(list, predicate, [context]) *Alias:****select***   
遍历**list**中的每个值，返回包含所有通过**predicate**真值检测的元素值。（愚人码头注：如果存在原生**filter**方法，则用原生的**filter**方法。）

var evens = \_.filter([1, 2, 3, 4, 5, 6], function(num){ return num % 2 == 0; });

=> [2, 4, 6]

### Where

\_.where(list, properties)   
遍历**list**中的每一个值，返回一个数组，这个数组包含**properties**所列出的属性的所有的 键 - 值对。

\_.where(listOfPlays, {author: "Shakespeare", year: 1611});

=> [{title: "Cymbeline", author: "Shakespeare", year: 1611},

{title: "The Tempest", author: "Shakespeare", year: 1611}]

### findWhere

\_.findWhere(list, properties)   
遍历整个**list**，返回匹配 **properties**参数所列出的所有 键 - 值 对的第一个值。

如果没有找到匹配的属性，或者**list**是空的，那么将返回*undefined*。

\_.findWhere(publicServicePulitzers, {newsroom: "The New York Times"});

=> {year: 1918, newsroom: "The New York Times",

reason: "For its public service in publishing in full so many official reports,

documents and speeches by European statesmen relating to the progress and

conduct of the war."}

### reject

\_.reject(list, predicate, [context])   
返回**list**中没有通过**predicate**真值检测的元素集合，与**filter**相反。

var odds = \_.reject([1, 2, 3, 4, 5, 6], function(num){ return num % 2 == 0; });

=> [1, 3, 5]

### Every

\_.every(list, [predicate], [context]) *Alias:****all***   
如果**list**中的所有元素都通过**predicate**的真值检测就返回*true*。（愚人码头注：如果存在原生的**every**方法，就使用原生的**every**。）

\_.every([true, 1, null, 'yes'], \_.identity);

=> false

### Some

\_.some(list, [predicate], [context]) *Alias:****any***   
如果**list**中有任何一个元素通过 **predicate** 的真值检测就返回*true*。一旦找到了符合条件的元素, 就直接中断对list的遍历. （愚人码头注：如果存在原生的**some**方法，就使用原生的**some**。）

\_.some([null, 0, 'yes', false]);

=> true

### Contains

\_.contains(list, value, [fromIndex]) *Alias:****includes***   
如果**list**包含指定的**value**则返回*true*（愚人码头注：使用===检测）。如果**list** 是数组，内部使用**indexOf**判断。使用**fromIndex**来给定开始检索的索引位置。

\_.contains([1, 2, 3], 3);

=> true

### Invoke

\_.invoke(list, methodName, \*arguments)   
在**list**的每个元素上执行**methodName**方法。 任何传递给**invoke**的额外参数，**invoke**都会在调用**methodName**方法的时候传递给它。

\_.invoke([[5, 1, 7], [3, 2, 1]], 'sort');

=> [[1, 5, 7], [1, 2, 3]]

### Pluck

\_.pluck(list, propertyName)   
**pluck**也许是**map**最常使用的用例模型的简化版本，即萃取数组对象中某属性值，返回一个数组。

var stooges = [{name: 'moe', age: 40}, {name: 'larry', age: 50}, {name: 'curly', age: 60}];

\_.pluck(stooges, 'name');

=> ["moe", "larry", "curly"]

### Max

\_.max(list, [iteratee], [context])   
返回**list**中的最大值。如果传递**iteratee**参数，**iteratee**将作为**list**中每个值的排序依据。如果**list**为空，将返回*-Infinity*，所以你可能需要事先用[isEmpty](https://www.css88.com/doc/underscore1.8.2/#isEmpty)检查 **list** 。

var stooges = [{name: 'moe', age: 40}, {name: 'larry', age: 50}, {name: 'curly', age: 60}];

\_.max(stooges, function(stooge){ return stooge.age; });

=> {name: 'curly', age: 60};

### Min

\_.min(list, [iteratee], [context])   
返回**list**中的最小值。如果传递**iteratee**参数，**iteratee**将作为**list**中每个值的排序依据。如果**list**为空，将返回*-Infinity*，所以你可能需要事先用[isEmpty](https://www.css88.com/doc/underscore1.8.2/#isEmpty)检查 **list** 。

var numbers = [10, 5, 100, 2, 1000];

\_.min(numbers);

=> 2

### sortBy

\_.sortBy(list, iteratee, [context])   
返回一个排序后的**list**拷贝副本。如果传递**iteratee**参数，**iteratee**将作为**list**中每个值的排序依据。迭代器也可以是字符串的属性的名称进行排序的(比如 length)。

\_.sortBy([1, 2, 3, 4, 5, 6], function(num){ return Math.sin(num); });

=> [5, 4, 6, 3, 1, 2]

var stooges = [{name: 'moe', age: 40}, {name: 'larry', age: 50}, {name: 'curly', age: 60}];

\_.sortBy(stooges, 'name');

=> [{name: 'curly', age: 60}, {name: 'larry', age: 50}, {name: 'moe', age: 40}];

### groupBy

\_.groupBy(list, iteratee, [context])   
把一个集合分组为多个集合，通过 **iterator** 返回的结果进行分组. 如果 **iterator** 是一个字符串而不是函数, 那么将使用 **iterator** 作为各元素的属性名来对比进行分组.

\_.groupBy([1.3, 2.1, 2.4], function(num){ return Math.floor(num); });

=> {1: [1.3], 2: [2.1, 2.4]}

\_.groupBy(['one', 'two', 'three'], 'length');

=> {3: ["one", "two"], 5: ["three"]}

### indexBy

\_.indexBy(list, iteratee, [context])   
给定一个**list**，和 一个用来返回一个在列表中的每个元素键 的**iterator** 函数（或属性名）， 返回一个每一项索引的对象。和[groupBy](https://www.css88.com/doc/underscore1.8.2/#groupBy)非常像，但是当你知道你的键是唯一的时候可以使用**indexBy** 。

var stooges = [{name: 'moe', age: 40}, {name: 'larry', age: 50}, {name: 'curly', age: 60}];

\_.indexBy(stooges, 'age');

=> {

"40": {name: 'moe', age: 40},

"50": {name: 'larry', age: 50},

"60": {name: 'curly', age: 60}

}

### countBy

\_.countBy(list, iteratee, [context])   
排序一个列表组成一个组，并且返回各组中的对象的数量的计数。类似groupBy，但是不是返回列表的值，而是返回在该组中值的数目。

\_.countBy([1, 2, 3, 4, 5], function(num) {

return num % 2 == 0 ? 'even': 'odd';

});

=> {odd: 3, even: 2}

### Shuffle

\_.shuffle(list)   
返回一个随机乱序的 **list** 副本, 使用 [Fisher-Yates shuffle](http://en.wikipedia.org/wiki/Fisher%E2%80%93Yates_shuffle) 来进行随机乱序.

\_.shuffle([1, 2, 3, 4, 5, 6]);

=> [4, 1, 6, 3, 5, 2]

### Sample

\_.sample(list, [n])   
从 **list**中产生一个随机样本。传递一个数字表示从**list**中返回**n**个随机元素。否则将返回一个单一的随机项。

\_.sample([1, 2, 3, 4, 5, 6]);

=> 4

\_.sample([1, 2, 3, 4, 5, 6], 3);

=> [1, 6, 2]

**toArray**\_.toArray(list)   
把**list**(任何可以迭代的对象)转换成一个数组，在转换 **arguments** 对象时非常有用。

(function(){ return \_.toArray(arguments).slice(1); })(1, 2, 3, 4);

=> [2, 3, 4]

### Size

\_.size(list)   
返回**list**的长度。

\_.size({one: 1, two: 2, three: 3});

=> 3

### Partition

\_.partition(array, predicate)   
拆分一个数组（**array**）为两个数组：  第一个数组其元素都满足**predicate**迭代函数， 而第二个的所有元素均不能满足**predicate**迭代函数。

\_.partition([0, 1, 2, 3, 4, 5], isOdd);

=> [[1, 3, 5], [0, 2, 4]]

**数组函数**

*注：****arguments（参数）****对象将在所有数组函数中工作 。然而, Underscore 函数的设计并不只是针对稀疏（"sparse" ）数组的.*

### First

\_.first(array, [n]) *Alias:****head****,****take***   
返回**array（数组）**的第一个元素。传递 **n**参数将返回数组中从第一个元素开始的**n**个元素（愚人码头注：返回数组中前 **n** 个元素.）。

\_.first([5, 4, 3, 2, 1]);

=> 5

### Initial

\_.initial(array, [n])   
返回数组中除了最后一个元素外的其他全部元素。 在arguments对象上特别有用。传递 **n**参数将从结果中排除从最后一个开始的**n**个元素（愚人码头注：排除数组后面的 **n** 个元素）。

\_.initial([5, 4, 3, 2, 1]);

=> [5, 4, 3, 2]

### Last

\_.last(array, [n])   
返回**array（数组）**的最后一个元素。传递 **n**参数将返回数组中从最后一个元素开始的**n**个元素（愚人码头注：返回数组里的后面的**n**个元素）。

\_.last([5, 4, 3, 2, 1]);

=> 1

### Rest

\_.rest(array, [index]) *Alias:****tail, drop***   
返回数组中除了第一个元素外的其他全部元素。传递 **index** 参数将返回从**index**开始的剩余所有元素 。（感谢[@德德德德撸](http://weibo.com/heerake) 指出错误）

\_.rest([5, 4, 3, 2, 1]);

=> [4, 3, 2, 1]

### Compact

\_.compact(array)   
返回一个除去所有*false*值的 **array**副本。 在javascript中, *false*, *null*, *0*, *""*, *undefined* 和 *NaN* 都是*false*值.

\_.compact([0, 1, false, 2, '', 3]);

=> [1, 2, 3]

### Flatten

\_.flatten(array, [shallow])   
将一个嵌套多层的数组 **array（数组）** (嵌套可以是任何层数)转换为只有一层的数组。 如果你传递 **shallow**参数，数组将只减少一维的嵌套。

\_.flatten([1, [2], [3, [[4]]]]);

=> [1, 2, 3, 4];

\_.flatten([1, [2], [3, [[4]]]], true);

=> [1, 2, 3, [[4]]];

### Without

\_.without(array, \*values)   
返回一个删除所有**values**值后的 **array**副本。（愚人码头注：使用*===*表达式做相等测试。）

\_.without([1, 2, 1, 0, 3, 1, 4], 0, 1);

=> [2, 3, 4]

### Union

\_.union(\*arrays)   
返回传入的 **arrays（数组）**并集：按顺序返回，返回数组的元素是唯一的，可以传入一个或多个 **arrays（数组）**。

\_.union([1, 2, 3], [101, 2, 1, 10], [2, 1]);

=> [1, 2, 3, 101, 10]

### Intersection

\_.intersection(\*arrays)   
返回传入 **arrays（数组）**交集。结果中的每个值是存在于传入的每个**arrays（数组）**里。

\_.intersection([1, 2, 3], [101, 2, 1, 10], [2, 1]);

=> [1, 2]

### Difference

\_.difference(array, \*others)   
类似于**without**，但返回的值来自**array**参数数组，并且不存在于**other** 数组.

\_.difference([1, 2, 3, 4, 5], [5, 2, 10]);

=> [1, 3, 4]

### Uniq

\_.uniq(array, [isSorted], [iteratee]) *Alias:****unique***   
返回 **array**去重后的副本, 使用 *===* 做相等测试. 如果您确定 **array** 已经排序, 那么给 **isSorted** 参数传递 *true*值, 此函数将运行的更快的算法. 如果要处理对象元素, 传递 **iteratee**函数来获取要对比的属性.

\_.uniq([1, 2, 1, 3, 1, 4]);

=> [1, 2, 3, 4]

### Zip

\_.zip(\*arrays)   
将 每个**arrays**中相应位置的值合并在一起。在合并分开保存的数据时很有用. 如果你用来处理矩阵嵌套数组时, \_.zip.apply 可以做类似的效果。

\_.zip(['moe', 'larry', 'curly'], [30, 40, 50], [true, false, false]);

=> [["moe", 30, true], ["larry", 40, false], ["curly", 50, false]]

### Unzip

\_.unzip(\*arrays)   
与[zip](https://www.css88.com/doc/underscore1.8.2/#zip)功能相反的函数，给定若干**arrays**，返回一串联的新数组，其第一元素个包含所有的输入数组的第一元素，其第二包含了所有的第二元素，依此类推。通过apply用于传递数组的数组。

\_.unzip(["moe", 30, true], ["larry", 40, false], ["curly", 50, false]);

=> [['moe', 'larry', 'curly'], [30, 40, 50], [true, false, false]]

### Object

\_.object(list, [values])   
将数组转换为对象。传递任何一个单独[key, value]对的列表，或者一个键的列表和一个值得列表。 如果存在重复键，最后一个值将被返回。

\_.object(['moe', 'larry', 'curly'], [30, 40, 50]);

=> {moe: 30, larry: 40, curly: 50}

\_.object([['moe', 30], ['larry', 40], ['curly', 50]]);

=> {moe: 30, larry: 40, curly: 50}

### indexOf

\_.indexOf(array, value, [isSorted])   
返回**value**在该 **array** 中的索引值，如果**value**不存在 **array**中就返回*-1*。使用原生的**indexOf** 函数，除非它失效。如果您正在使用一个大数组，你知道数组已经排序，传递true给**isSorted**将更快的用二进制搜索..,或者，传递一个数字作为第三个参数，为了在给定的索引的数组中寻找第一个匹配值。

\_.indexOf([1, 2, 3], 2);

=> 1

### lastIndexOf

\_.lastIndexOf(array, value, [fromIndex])   
返回**value**在该 **array** 中的从最后开始的索引值，如果**value**不存在 **array**中就返回*-1*。如果支持原生的**lastIndexOf**，将使用原生的**lastIndexOf**函数。传递**fromIndex**将从你给定的索性值开始搜索。

\_.lastIndexOf([1, 2, 3, 1, 2, 3], 2);

=> 4

### sortedIndex

\_.sortedIndex(list, value, [iteratee], [context])   
使用二分查找确定**value**在**list**中的位置序号，**value**按此序号插入能保持**list**原有的排序。如果提供**iterator**函数，**iterator**将作为list排序的依据，包括你传递的**value** 。**iterator**也可以是字符串的属性名用来排序(比如length)。

\_.sortedIndex([10, 20, 30, 40, 50], 35);

=> 3

var stooges = [{name: 'moe', age: 40}, {name: 'curly', age: 60}];

\_.sortedIndex(stooges, {name: 'larry', age: 50}, 'age');

=> 1

### findIndex

\_.findIndex(array, predicate, [context])   
类似于[\_.indexOf](https://www.css88.com/doc/underscore1.8.2/#indexOf)，当**predicate**通过真检查时，返回第一个索引值；否则返回*-1*。

\_.findIndex([4, 6, 8, 12], isPrime);

=> -1 // not found

\_.findIndex([4, 6, 7, 12], isPrime);

=> 2

### findLastIndex

\_.findLastIndex(array, predicate, [context])   
和[\_.findIndex](https://www.css88.com/doc/underscore1.8.2/#findIndex)类似，但反向迭代数组，当**predicate**通过真检查时，最接近末端的索引值将被返回。

var users = [{'id': 1, 'name': 'Bob', 'last': 'Brown'},

{'id': 2, 'name': 'Ted', 'last': 'White'},

{'id': 3, 'name': 'Frank', 'last': 'James'},

{'id': 4, 'name': 'Ted', 'last': 'Jones'}];

\_.findLastIndex(users, {

name: 'Ted'

});

=> 3

### Range

\_.range([start], stop, [step])   
一个用来创建整数灵活编号的列表的函数，便于each 和 map循环。如果省略**start**则默认为 *0*；**step** 默认为 *1*.返回一个从**start** 到**stop**的整数的列表，用**step**来增加 （或减少）独占。值得注意的是，如果**stop**值在**start**前面（也就是**stop**值小于**start**值），那么值域会被认为是零长度，而不是负增长。-如果你要一个负数的值域 ，请使用负数**step**.

\_.range(10);

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

\_.range(1, 11);

=> [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

\_.range(0, 30, 5);

=> [0, 5, 10, 15, 20, 25]

\_.range(0, -10, -1);

=> [0, -1, -2, -3, -4, -5, -6, -7, -8, -9]

\_.range(0);

=> []

**与函数有关的函数**

### Bind

\_.bind(function, object, \*arguments)   
绑定函数 **function** 到对象 **object** 上, 也就是无论何时调用函数, 函数里的 *this* 都指向这个 **object**.任意可选参数 **arguments** 可以传递给函数 **function** , 可以填充函数所需要的参数,这也被称为 **partial application**。对于没有结合上下文的**partial application**绑定，请使用[partial](https://www.css88.com/doc/underscore1.8.2/#partial)。   
(愚人码头注：partial application翻译成“部分应用”或者“偏函数应用”。partial application可以被描述为一个函数，它接受一定数目的参数，绑定值到一个或多个这些参数，并返回一个新的函数，这个返回函数只接受剩余未绑定值的参数。参见：<http://en.wikipedia.org/wiki/Partial_application>。感谢[@一任风月忆秋年](http://weibo.com/u/2654434300)的建议)。

var func = function(greeting){ return greeting + ': ' + this.name };

func = \_.bind(func, {name: 'moe'}, 'hi');

func();

=> 'hi: moe'

### bindAll

\_.bindAll(object, \*methodNames)   
把**methodNames**参数指定的一些方法绑定到**object**上，这些方法就会在对象的上下文环境中执行。绑定函数用作事件处理函数时非常便利，否则函数被调用时*this*一点用也没有。**methodNames**参数是必须的。

var buttonView = {

label : 'underscore',

onClick: function(){ alert('clicked: ' + this.label); },

onHover: function(){ console.log('hovering: ' + this.label); }

};

\_.bindAll(buttonView, 'onClick', 'onHover');

// When the button is clicked, this.label will have the correct value.

jQuery('#underscore\_button').bind('click', buttonView.onClick);

### partial

\_.partial(function, \*arguments)   
局部应用一个函数填充在任意个数的 **arguments**，*不*改变其动态this值。和[bind](https://www.css88.com/doc/underscore1.8.2/#bind)方法很相近。你可以传递\_ 给**arguments**列表来指定一个不预先填充，但在调用时提供的参数。

var subtract = function(a, b) { return b - a; };

sub5 = \_.partial(subtract, 5);

sub5(20);

=> 15

// Using a placeholder

subFrom20 = \_.partial(subtract, \_, 20);

subFrom20(5);

=> 15

### Memorize

\_.memoize(function, [hashFunction])   
Memoizes方法可以缓存某函数的计算结果。对于耗时较长的计算是很有帮助的。如果传递了 **hashFunction** 参数，就用 **hashFunction** 的返回值作为key存储函数的计算结果。**hashFunction** 默认使用function的第一个参数作为key。memoized值的缓存可作为返回函数的cache属性。

var fibonacci = \_.memoize(function(n) {

return n < 2 ? n: fibonacci(n - 1) + fibonacci(n - 2);

});

### Delay

\_.delay(function, wait, \*arguments)   
类似**setTimeout**，等待**wait**毫秒后调用**function**。如果传递可选的参数**arguments**，当函数**function**执行时， **arguments** 会作为参数传入。

var log = \_.bind(console.log, console);

\_.delay(log, 1000, 'logged later');

=> 'logged later' // Appears after one second.

### Defer

\_.defer(function, \*arguments)   
延迟调用**function**直到当前调用栈清空为止，类似使用延时为0的**setTimeout**方法。对于执行开销大的计算和无阻塞UI线程的HTML渲染时候非常有用。 如果传递**arguments**参数，当函数**function**执行时， **arguments** 会作为参数传入。

\_.defer(function(){ alert('deferred'); });

// Returns from the function before the alert runs.

### Throttle

\_.throttle(function, wait, [options])   
创建并返回一个像节流阀一样的函数，当重复调用函数的时候，最多每隔 **wait**毫秒调用一次该函数。对于想控制一些触发频率较高的事件有帮助。（愚人码头注：详见：[javascript函数的throttle和debounce](http://www.css88.com/archives/5256)）

默认情况下，**throttle**将在你调用的第一时间尽快执行这个**function**，并且，如果你在**wait**周期内调用任意次数的函数，都将尽快的被覆盖。如果你想禁用第一次首先执行的话，传递{leading: false}，还有如果你想禁用最后一次执行的话，传递{trailing: false}。

var throttled = \_.throttle(updatePosition, 100);

$(window).scroll(throttled);

### Debounce

\_.debounce(function, wait, [immediate])   
返回 **function** 函数的防反跳版本, 将延迟函数的执行(真正的执行)在函数最后一次调用时刻的 **wait** 毫秒之后. 对于必须在一些输入（多是一些用户操作）停止到达*之后*执行的行为有帮助。 例如: 渲染一个Markdown格式的评论预览, 当窗口停止改变大小之后重新计算布局, 等等.

传参 **immediate** 为 true， **debounce**会在 **wait** 时间间隔的开始调用这个函数 。（愚人码头注：并且在 waite 的时间之内，不会再次调用。）在类似不小心点了提交按钮两下而提交了两次的情况下很有用。 （感谢 [@ProgramKid](http://weibo.com/u/1768962955)的翻译建议）

var lazyLayout = \_.debounce(calculateLayout, 300);

$(window).resize(lazyLayout);

### Once

\_.once(function)   
创建一个只能调用一次的函数。重复调用改进的方法也没有效果，只会返回第一次执行时的结果。 作为初始化函数使用时非常有用, 不用再设一个boolean值来检查是否已经初始化完成.

var initialize = \_.once(createApplication);

initialize();

initialize();

// Application is only created once.

### After

\_.after(count, function)   
创建一个函数, 只有在运行了 **count** 次之后才有效果. 在处理同组异步请求返回结果时, 如果你要确保同组里所有异步请求完成之后才 执行这个函数, 这将非常有用。

var renderNotes = \_.after(notes.length, render);

\_.each(notes, function(note) {

note.asyncSave({success: renderNotes});

});

// renderNotes is run once, after all notes have saved.

### Before

\_.before(count, function)   
创建一个函数,调用不超过**count** 次。 当**count**已经达到时，最后一个函数调用的结果将被记住并返回。

var monthlyMeeting = \_.before(3, askForRaise);

monthlyMeeting();

monthlyMeeting();

monthlyMeeting();

// the result of any subsequent calls is the same as the second call

### Wrap

\_.wrap(function, wrapper)   
将第一个函数 **function** 封装到函数 **wrapper** 里面, 并把函数 **function** 作为第一个参数传给 **wrapper**. 这样可以让 **wrapper** 在 **function** 运行之前和之后 执行代码, 调整参数然后附有条件地执行.

var hello = function(name) { return "hello: " + name; };

hello = \_.wrap(hello, function(func) {

return "before, " + func("moe") + ", after";

});

hello();

=> 'before, hello: moe, after'

### Negate

\_.negate(predicate)   
返回一个新的**predicate**函数的否定版本。

var isFalsy = \_.negate(Boolean);

\_.find([-2, -1, 0, 1, 2], isFalsy);

=> 0

### Compose

\_.compose(\*functions)   
返回函数集 **functions** 组合后的复合函数, 也就是一个函数执行完之后把返回的结果再作为参数赋给下一个函数来执行. 以此类推. 在数学里, 把函数 *f()*, *g()*, 和 *h()* 组合起来可以得到复合函数 *f(g(h()))*。

var greet = function(name){ return "hi: " + name; };

var exclaim = function(statement){ return statement.toUpperCase() + "!"; };

var welcome = \_.compose(greet, exclaim);

welcome('moe');

=> 'hi: MOE!'

**对象函数**

### Keys

\_.keys(object)   
检索**object**拥有的所有可枚举属性的名称。

\_.keys({one: 1, two: 2, three: 3});

=> ["one", "two", "three"]

### allKeys

\_.allKeys(object)   
检索**object**拥有的和继承的所有属性的名称。

function Stooge(name) {

this.name = name;

}

Stooge.prototype.silly = true;

\_.allKeys(new Stooge("Moe"));

=> ["name", "silly"]

### Values

\_.values(object)   
返回**object**对象所有的属性值。

\_.values({one: 1, two: 2, three: 3});

=> [1, 2, 3]

### mapObject

\_.mapObject(object, iteratee, [context])   
它类似于[map](https://www.css88.com/doc/underscore1.8.2/#map)，但是这用于对象。转换每个属性的值。

\_.mapObject({start: 5, end: 12}, function(val, key) {

return val + 5;

});

=> {start: 10, end: 17}

### Pairs

\_.pairs(object)   
把一个对象转变为一个[key, value]形式的数组。

\_.pairs({one: 1, two: 2, three: 3});

=> [["one", 1], ["two", 2], ["three", 3]]

### Invert

\_.invert(object)   
返回一个**object**副本，使其键（keys）和值（values）对换。对于这个操作，必须确保object里所有的值都是唯一的且可以序列号成字符串.

\_.invert({Moe: "Moses", Larry: "Louis", Curly: "Jerome"});

=> {Moses: "Moe", Louis: "Larry", Jerome: "Curly"};

### Functions

\_.functions(object) *Alias:****methods***   
返回一个对象里所有的方法名, 而且是已经排序的 — 也就是说, 对象里每个方法(属性值是一个函数)的名称.

\_.functions(\_);

=> ["all", "any", "bind", "bindAll", "clone", "compact", "compose" ...

### findKey

\_.findKey(object, predicate, [context])   
Similar to [\_.findIndex](https://www.css88.com/doc/underscore1.8.2/#findIndex) but for keys in objects. Returns the *key* where the **predicate** truth test passes or *undefined*.

### Extend

\_.extend(destination, \*sources)   
复制**source**对象中的所有属性覆盖到**destination**对象上，并且返回 **destination** 对象. 复制是按顺序的, 所以后面的对象属性会把前面的对象属性覆盖掉(如果有重复).

\_.extend({name: 'moe'}, {age: 50});

=> {name: 'moe', age: 50}

### extendOwn

\_.extendOwn(destination, \*sources) *Alias:****assign***   
类似于 **extend**, 但只复制*自己的*属性覆盖到目标对象。（愚人码头注：不包括继承过来的属性）

### Pick

\_.pick(object, \*keys)   
返回一个**object**副本，只过滤出**keys**(有效的键组成的数组)参数指定的属性值。或者接受一个判断函数，指定挑选哪个key。

\_.pick({name: 'moe', age: 50, userid: 'moe1'}, 'name', 'age');

=> {name: 'moe', age: 50}

\_.pick({name: 'moe', age: 50, userid: 'moe1'}, function(value, key, object) {

return \_.isNumber(value);

});

=> {age: 50}

### Omit

\_.omit(object, \*keys)   
返回一个**object**副本，只过滤出除去**keys**(有效的键组成的数组)参数指定的属性值。 或者接受一个判断函数，指定忽略哪个key。

\_.omit({name: 'moe', age: 50, userid: 'moe1'}, 'userid');

=> {name: 'moe', age: 50}

\_.omit({name: 'moe', age: 50, userid: 'moe1'}, function(value, key, object) {

return \_.isNumber(value);

});

=> {name: 'moe', userid: 'moe1'}

### Defaults

\_.defaults(object, \*defaults)   
用**defaults**对象填充**object** 中的undefined属性。 并且返回这个**object**。一旦这个属性被填充，再使用defaults方法将不会有任何效果。（感谢[@一任风月忆秋年](http://weibo.com/u/2654434300)的拍砖）

var iceCream = {flavor: "chocolate"};

\_.defaults(iceCream, {flavor: "vanilla", sprinkles: "lots"});

=> {flavor: "chocolate", sprinkles: "lots"}

### Clone

\_.clone(object)   
创建 一个浅复制（浅拷贝）的克隆**object**。任何嵌套的对象或数组都通过引用拷贝，不会复制。

\_.clone({name: 'moe'});

=> {name: 'moe'};

### Tap

\_.tap(object, interceptor)   
用 **object**作为参数来调用函数**interceptor**，然后返回**object**。这种方法的主要意图是作为函数链式调用 的一环, 为了对此对象执行操作并返回对象本身。

\_.chain([1,2,3,200])

.filter(function(num) { return num % 2 == 0; })

.tap(alert)

.map(function(num) { return num \* num })

.value();

=> // [2, 200] (alerted)

=> [4, 40000]

### Has

\_.has(object, key)   
对象是否包含给定的键吗？等同于object.hasOwnProperty(key)，但是使用hasOwnProperty 函数的一个安全引用，以防[意外覆盖](http://www.devthought.com/2012/01/18/an-object-is-not-a-hash/)。

\_.has({a: 1, b: 2, c: 3}, "b");

=> true

### Property

\_.property(key)   
返回一个函数，这个函数返回任何传入的对象的key属性。

var stooge = {name: 'moe'};

'moe' === \_.property('name')(stooge);

=> true

propertyOf

\_.propertyOf(object)   
和\_.property相反。需要一个对象，并返回一个函数,这个函数将返回一个提供的属性的值。

var stooge = {name: 'moe'};

\_.propertyOf(stooge)('name');

=> 'moe'

### Matcher

\_.matcher(attrs)   
返回一个断言函数，这个函数会给你一个断言可以用来辨别给定的对象是否匹配**attrs**指定键/值属性。

var ready = \_.matcher({selected: true, visible: true});

var readyToGoList = \_.filter(list, ready);

### isEqual

\_.isEqual(object, other)   
执行两个对象之间的优化深度比较，确定他们是否应被视为相等。

var stooge = {name: 'moe', luckyNumbers: [13, 27, 34]};

var clone = {name: 'moe', luckyNumbers: [13, 27, 34]};

stooge == clone;

=> false

\_.isEqual(stooge, clone);

=> true

### isMatch

\_.isMatch(object, properties)   
告诉你**properties**中的键和值是否包含在**object**中。

var stooge = {name: 'moe', age: 32};

\_.isMatch(stooge, {age: 32});

=> true

### isEmpty

\_.isEmpty(object)   
如果**object** 不包含任何值(没有可枚举的属性)，返回*true*。 对于字符串和类数组（array-like）对象，如果length属性为0，那么\_.isEmpty检查返回*true*。

\_.isEmpty([1, 2, 3]);

=> false

\_.isEmpty({});

=> true

### isElement

\_.isElement(object)   
如果**object**是一个DOM元素，返回*true*。

\_.isElement(jQuery('body')[0]);

=> true

### isArray

\_.isArray(object)   
如果**object**是一个数组，返回*true*。

(function(){ return \_.isArray(arguments); })();

=> false

\_.isArray([1,2,3]);

=> true

### isObject

\_.isObject(value)   
如果**object**是一个对象，返回*true*。需要注意的是JavaScript数组和函数是对象，字符串和数字不是。

\_.isObject({});

=> true

\_.isObject(1);

=> false

### isArguments

\_.isArguments(object)   
如果**object**是一个参数对象，返回*true*。

(function(){ return \_.isArguments(arguments); })(1, 2, 3);

=> true

\_.isArguments([1,2,3]);

=> false

### isFunction

\_.isFunction(object)   
如果**object**是一个函数（Function），返回*true*。

\_.isFunction(alert);

=> true

### isString

\_.isString(object)   
如果**object**是一个字符串，返回*true*。

\_.isString("moe");

=> true

isNumber\_.isNumber(object)   
如果**object**是一个数值，返回*true* (包括 NaN)。

\_.isNumber(8.4 \* 5);

=> true

### isFinite

\_.isFinite(object)   
如果**object**是一个有限的数字，返回*true*。

\_.isFinite(-101);

=> true

\_.isFinite(-Infinity);

=> false

### isBoolean

\_.isBoolean(object)   
如果**object**是一个布尔值，返回*true*，否则返回*false*。

\_.isBoolean(null);

=> false

### isDate

\_.isDate(object)   
Returns *true* if **object** is a Date.

\_.isDate(new Date());

=> true

### isRegExp

\_.isRegExp(object)   
如果**object**是一个正则表达式，返回*true*。

\_.isRegExp(/moe/);

=> true

### isError

\_.isError(object)   
如果**object**继承至 Error 对象，那么返回 *true*。

try {

throw new TypeError("Example");

} catch (o\_O) {

\_.isError(o\_O)

}

=> true

### isNaN

\_.isNaN(object)   
如果**object**是 *NaN*，返回*true*。   
注意： 这和原生的**isNaN** 函数不一样，如果变量是*undefined*，原生的**isNaN** 函数也会返回 *true* 。

\_.isNaN(NaN);

=> true

isNaN(undefined);

=> true

\_.isNaN(undefined);

=> false

### isNull

\_.isNull(object)   
如果**object**的值是 *null*，返回*true*。

\_.isNull(null);

=> true

\_.isNull(undefined);

=> false

### isUndefined

\_.isUndefined(value)   
如果**value**是*undefined*，返回*true*。

\_.isUndefined(window.missingVariable);

=> true

**实用功能**

### noConflict

\_.noConflict()   
放弃**Underscore** 的控制变量"\_"。返回**Underscore** 对象的引用。

var underscore = \_.noConflict();

### identity

\_.identity(value)   
返回与传入参数相等的值. 相当于数学里的: f(x) = x  
这个函数看似无用, 但是在Underscore里被用作默认的迭代器iterator.

var stooge = {name: 'moe'};

stooge === \_.identity(stooge);

=> true

Constant

\_.constant(value)   
创建一个函数，这个函数 返回相同的值 用来作为\_.constant的参数。

var stooge = {name: 'moe'};

stooge === \_.constant(stooge)();

=> true

### Noop

\_.noop()   
返回undefined，不论传递给它的是什么参数。 可以用作默认可选的回调参数。

obj.initialize = \_.noop;

### times

\_.times(n, iteratee, [context])   
调用给定的迭代函数**n**次,每一次调用**iteratee**传递index参数。生成一个返回值的数组。   
*注意: 本例使用*[*链式语法*](https://www.css88.com/doc/underscore1.8.2/#chaining)。

\_(3).times(function(n){ genie.grantWishNumber(n); });

### Random

\_.random(min, max)   
返回一个**min** 和 **max**之间的随机整数。如果你只传递一个参数，那么将返回0和这个参数之间的整数。

\_.random(0, 100);

=> 42

### Mixin

\_.mixin(object)   
允许用您自己的实用程序函数扩展Underscore。传递一个 {name: function}定义的哈希添加到Underscore对象，以及面向对象封装。

\_.mixin({

capitalize: function(string) {

return string.charAt(0).toUpperCase() + string.substring(1).toLowerCase();

}

});

\_("fabio").capitalize();

=> "Fabio"

### Iterate

\_.iteratee(value, [context])   
一个重要的内部函数用来生成可应用到集合中每个元素的回调， 返回想要的结果 - 无论是等式，任意回调，属性匹配，或属性访问。   
通过\_.iteratee转换判断的Underscore 方法的完整列表是 map, find, filter, reject, every, some, max, min, sortBy, groupBy, indexBy, countBy, sortedIndex, partition, 和 unique.

var stooges = [{name: 'curly', age: 25}, {name: 'moe', age: 21}, {name: 'larry', age: 23}];

\_.map(stooges, \_.iteratee('age'));

=> [25, 21, 23];

### uniqueId

\_.uniqueId([prefix])   
为需要的客户端模型或DOM元素生成一个全局唯一的id。如果**prefix**参数存在， id 将附加给它。

\_.uniqueId('contact\_');

=> 'contact\_104'

### Escape

\_.escape(string)   
转义HTML字符串，替换&, <, >, ", ', 和 /字符。

\_.escape('Curly, Larry & Moe');

=> "Curly, Larry &amp; Moe"

### Unescape

\_.unescape(string)   
和[**escape**](https://www.css88.com/doc/underscore1.8.2/#escape)相反。转义HTML字符串，替换&, &lt;, &gt;, &quot;, &#96;, 和 &#x2F;字符。

\_.unescape('Curly, Larry &amp; Moe');

=> "Curly, Larry & Moe"

### Result

\_.result(object, property, [defaultValue])   
如果指定的**property** 的值是一个函数，那么将在**object**上下文内调用它;否则，返回它。如果提供默认值，并且属性不存在，那么默认值将被返回。如果设置defaultValue是一个函数，它的结果将被返回。

var object = {cheese: 'crumpets', stuff: function(){ return 'nonsense'; }};

\_.result(object, 'cheese');

=> "crumpets"

\_.result(object, 'stuff');

=> "nonsense"

\_.result(object, 'meat', 'ham');

=> "ham"

### Now

\_.now()   
一个优化的方式来获得一个当前时间的整数时间戳。可用于实现定时/动画功能。

\_.now();

=> 1392066795351

### Template

\_.template(templateString, [settings])   
将 JavaScript 模板编译为可以用于页面呈现的函数, 对于通过JSON数据源生成复杂的HTML并呈现出来的操作非常有用。 模板函数可以使用 <%= … %>插入变量, 也可以用<% … %>执行任意的 JavaScript 代码。 如果您希望插入一个值, 并让其进行HTML转义,请使用<%- … %>。 当你要给模板函数赋值的时候，可以传递一个含有与模板对应属性的**data**对象 。 如果您要写一个一次性的, 您可以传对象 **data** 作为第二个参数给模板 **template** 来直接呈现, 这样页面会立即呈现而不是返回一个模板函数. 参数 **settings** 是一个哈希表包含任何可以覆盖的设置 \_.templateSettings.

var compiled = \_.template("hello: <%= name %>");

compiled({name: 'moe'});

=> "hello: moe"

var template = \_.template("<b><%- value %></b>");

template({value: '<script>'});

=> "<b>&lt;script&gt;</b>"

您也可以在JavaScript代码中使用 print. 有时候这会比使用 <%= ... %> 更方便.

var compiled = \_.template("<% print('Hello ' + epithet); %>");

compiled({epithet: "stooge"});

=> "Hello stooge"

如果ERB式的分隔符您不喜欢, 您可以改变Underscore的模板设置, 使用别的符号来嵌入代码.定义一个 **interpolate** 正则表达式来逐字匹配嵌入代码的语句, 如果想插入转义后的HTML代码则需要定义一个 **escape** 正则表达式来匹配,还有一个 **evaluate** 正则表达式来匹配您想要直接一次性执行程序而不需要任何返回值的语句.您可以定义或省略这三个的任意一个.例如, 要执行[Mustache.js](http://github.com/janl/mustache.js#readme)类型的模板:

\_.templateSettings = {

interpolate: /\{\{(.+?)\}\}/g

};

var template = \_.template("Hello {{ name }}!");

template({name: "Mustache"});

=> "Hello Mustache!"

默认的, **template** 通过 with 语句来取得 data 所有的值. 当然, 您也可以在 **variable** 设置里指定一个变量名. 这样能显著提升模板的渲染速度.

\_.template("Using 'with': <%= data.answer %>", {variable: 'data'})({answer: 'no'});

=> "Using 'with': no"

预编译模板对调试不可重现的错误很有帮助. 这是因为预编译的模板可以提供错误的代码行号和堆栈跟踪, 有些模板在客户端(浏览器)上是不能通过编译的 在编译好的模板函数上, 有 **source** 属性可以提供简单的预编译功能.

<script>

JST.project = <%= \_.template(jstText).source %>;

</script>

**链式语法**

您可以在面向对象或者函数的风格下使用Underscore, 这取决于您的个人偏好. 以下两行代码都可以 把一个数组里的所有数字乘以2.

\_.map([1, 2, 3], function(n){ return n \* 2; });

\_([1, 2, 3]).map(function(n){ return n \* 2; });

对一个对象使用 chain 方法, 会把这个对象封装并 让以后每次方法的调用结束后都返回这个封装的对象, 当您完成了计算, 可以使用 value 函数来取得最终的值. 以下是一个同时使用了 **map/flatten/reduce** 的链式语法例子, 目的是计算一首歌的歌词里每一个单词出现的次数.

var lyrics = [

{line: 1, words: "I'm a lumberjack and I'm okay"},

{line: 2, words: "I sleep all night and I work all day"},

{line: 3, words: "He's a lumberjack and he's okay"},

{line: 4, words: "He sleeps all night and he works all day"}

];

\_.chain(lyrics)

.map(function(line) { return line.words.split(' '); })

.flatten()

.reduce(function(counts, word) {

counts[word] = (counts[word] || 0) + 1;

return counts;

}, {})

.value();

=> {lumberjack: 2, all: 4, night: 2 ... }

In addition, the 此外, [数组原型方法](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Array/prototype) 也通过代理加入到了链式封装的Underscore对象, 所以您可以 在链式语法中直接使用 reverse 或 push 方法, 然后再接着其他的语句.

### Chain

\_.chain(obj)   
返回一个封装的对象. 在封装的对象上调用方法会返回封装的对象本身, 直道 value 方法调用为止.

var stooges = [{name: 'curly', age: 25}, {name: 'moe', age: 21}, {name: 'larry', age: 23}];

var youngest = \_.chain(stooges)

.sortBy(function(stooge){ return stooge.age; })

.map(function(stooge){ return stooge.name + ' is ' + stooge.age; })

.first()

.value();

=> "moe is 21"

### Value

\_(obj).value()   
获取封装对象的最终值.

\_([1, 2, 3]).value();

=> [1, 2, 3]

**更多链接 & 推荐阅读**

Underscore文档也有 [简体中文](http://www.css88.com/doc/underscore/) 版

[Underscore.lua](http://mirven.github.com/underscore.lua/), 一个Lua版本的Underscore, 函数都通用. 包含面向对象封装和链式语法. ([源码](http://github.com/mirven/underscore.lua))

[Underscore.m](http://underscorem.org/), 一个 Objective-C 版本的 Underscore.js, 实现了大部分函数, 它的语法鼓励使用链式语法. ([源码](https://github.com/robb/Underscore.m))

[\_.m](http://kmalakoff.github.com/_.m/), 另一个 Objective-C 版本, 这个版本与原始的 Underscore.js API 比较相近. ([源码](https://github.com/kmalakoff/_.m))

[Underscore.php](http://brianhaveri.github.com/Underscore.php/), 一个PHP版本的Underscore, 函数都通用. 包含面向对象封装和链式语法. ([源码](http://github.com/brianhaveri/Underscore.php))

[Underscore-perl](http://vti.github.com/underscore-perl/), 一个Perl版本的Underscore, 实现了大部分功能, 主要针对于Perl的哈希表和数组. ([源码](https://github.com/vti/underscore-perl/))

[Underscore.cfc](http://russplaysguitar.github.com/UnderscoreCF/), 一个 Coldfusion 版本的 Underscore.js, 实现了大部分函数. ([源码](https://github.com/russplaysguitar/underscorecf))

[Underscore.cfc](http://russplaysguitar.github.io/UnderscoreCF/), a Coldfusion port of many of the Underscore.js functions. ([source](https://github.com/russplaysguitar/underscorecf))

[Underscore.string](https://github.com/edtsech/underscore.string), 一个Underscore的扩展, 添加了多个字符串操作的函数, 如: trim, startsWith, contains, capitalize, reverse, sprintf, 还有更多.

Ruby的 [枚举](http://ruby-doc.org/core/classes/Enumerable.html) 模块.

[Prototype.js](http://www.prototypejs.org/), 提供类似于Ruby枚举方式的JavaScript集合函数.

Oliver Steele的 [Functional JavaScript](http://osteele.com/sources/javascript/functional/), 包含全面的高阶函数支持以及字符串的匿名函数.

Michael Aufreiter的 [Data.js](http://github.com/michael/data), 一个JavaScript的数据操作和持久化的类库.

Python的 [迭代工具](http://docs.python.org/library/itertools.html).

[PyToolz](http://toolz.readthedocs.org/), 一个Python端口         扩展了itertools和functools，包括很多的Underscore API。

[Funcy](https://github.com/Suor/funcy), a practical collection of functional helpers for Python, partially inspired by Underscore.

**Change Log**

**1.8.2** — *Feb. 22, 2015* — [Diff](https://github.com/jashkenas/underscore/compare/1.8.1...1.8.2) — [Docs](https://cdn.rawgit.com/jashkenas/underscore/1.8.2/index.html)

恢复先前在1.8.1改变的老版本 Internet Explorer的边界情况。

\_.contains添加了一个 fromIndex 参数。

**1.8.1** — *Feb. 19, 2015* — [Diff](https://github.com/jashkenas/underscore/compare/1.8.0...1.8.1) — [Docs](https://cdn.rawgit.com/jashkenas/underscore/1.8.1/index.html)

修复/改变一些老版本Internet Explorer和边界情况的行为。 用老版本Internet Explorer和Underscore 1.8.1 测试一下您的应用，让我们知道发生了什么...

**1.8.0** — *Feb. 19, 2015* — [Diff](https://github.com/jashkenas/underscore/compare/1.7.0...1.8.0) — [Docs](https://cdn.rawgit.com/jashkenas/underscore/1.8.0/index.html)

新增\_.mapObject，它类似于\_.map，但只是对于在对象中的值。

新增\_.allKeys，它返回一个对象上*所有*枚举属性名称。

恢复一个1.7.0的变更，\_.extend仅复制“自己”的属性（愚人码头注：不包括继承过来的属性）。 希望没有伤害到你 — 如果它这次伤害到了你，我向你道歉。

添加\_.extendOwn — \_.extend的一个不太有用的形式，仅复制“自己”的属性，（愚人码头注：不包括继承过来的属性）。

添加\_.findIndex和\_.findLastIndex功能，很好地补充了\_.indexOf 和 \_.lastIndexOf。

增加了一个\_.isMatch 断言（判断）函数，它会告诉你，如果一个对象匹配的key-value属性。和\_.isEqual 和 \_.matcher非常接近。

增加了一个\_.isError功能。

恢复\_.unzip功能，做为zip相反的功能。回滚。

\_.result现在有一个可选的回退值（或函数提供回退值）。

添加\_.propertyOf 函数发生器作为\_.property的镜像版本。

弃用\_.matches。目前他已经有一个更加和谐的名称 -\_.matcher。

简化各种多样的代码，以提高跨平台兼容性的变化，修复边界情况的bug。

**1.6.0** — *February 10, 2014* — [Diff](https://github.com/jashkenas/underscore/compare/1.5.2...1.6.0) — [Docs](http://htmlpreview.github.io/?https://raw.github.com/jashkenas/underscore/1.6.0/index.html)

Underscore 现在将自己注册为AMD（Require.js），Bower和Component， 以及作为一个CommonJS的模块和常规（Java）的脚本。             虽然比较丑陋，但也许是必要的。

添加了 \_.partition, 一个拆分一个集合为两个结果列表，第一个数组其元素都满足**predicate**迭代函数， 而第二个的所有元素均不能满足**predicate**迭代函数。

添加了 \_.property, 创建一个迭代器，轻松从对象中获取特定属性。 与其他 Underscore 集合函数结合使用时很有用。

添加了 \_.matches, 一个函数，它会给你一个断言可以用来辨别 给定的对象是否匹配指定键/值属性的列表。

添加了 \_.constant, 作为\_.identity高阶.

添加了 \_.now, 一个优化的方式来获得一个时间戳 — 在内部用来加快debounce 和 throttle。

\_.partial函数 现在可以用来部分适用的任何参数， 通过传递\_，无论你想要一个占位符变量， 稍后填充。

\_.each 函数现在 返回一个列表的引用，方便链式调用。

The \_.keys 函数 现在 当空对象传入的时候返回一个空数组。

… 更多杂项重构.

**1.5.2** — *Sept. 7, 2013* — [Diff](https://github.com/jashkenas/underscore/compare/1.5.1...1.5.2)

增加了indexBy函数，他是countBy and groupBy功能相辅相成。

增加了sample函数，从数组中产生随机元素。

一些有关函数的优化，\_.keys 方面的实现（包含大幅提升的对象上each 函数）。另外debounce中一个紧密的循环。

**1.5.1** — *Jul. 8, 2013* — [Diff](https://github.com/jashkenas/underscore/compare/1.5.0...1.5.1)

删除unzip，因为她简单的应用了zip参数的一个数组。使用\_.zip.apply(\_, list)代替。

**1.5.0** — *Jul. 6, 2013* — [Diff](https://github.com/jashkenas/underscore/compare/1.4.4...1.5.0)

添加一个unzip新函数，作为\_.zip功能相反的函数。

throttle函数现在增加一个options参数，如果你想禁用第一次首先执行的{leading: false}，还有如果你想禁用最后一次执行的话，传递{trailing: false}。

Underscore现在提供了一个source map 方便压缩文件的调试。

defaults函数现在只 重写undefined值，不再重写null值。

删除不带方法名参数调用\_.bindAll的能力。

删除计数为0，调用 \_.after 的能力。调用的最小数量现在是1（自然数）

**1.4.4** — *Jan. 30, 2013* — [Diff](https://github.com/jashkenas/underscore/compare/1.4.3...1.4.4)

添加\_.findWhere，在列表中找到的第一个元素，一组特定的键和值相匹配。

添加\_.partial，局部应用一个函数填充在任意数值的参数， *不*改变其动态this值。

通过去掉了一些的边缘案件涉包括构造函数来简化bind。总之：不要\_.bind 你的构造器。

一个invoke的小优化。

修改压缩版本中由于不当压缩引起的isFunctionBUG。

**1.4.3** — *Dec. 4, 2012* — [Diff](https://github.com/jashkenas/underscore/compare/1.4.2...1.4.3)

改进Underscore和 与Adobe的JS引擎的兼容性，可用于script Illustrator，Photoshop和相关产品。

添加一个默认的\_.identity迭代到countBy和groupBy中。

uniq函数现在接受array, iterator, context作为参数列表。

times函数现在放回迭代函数结果的映射数组。

简化和修复throttleBUG。

**1.4.2** — *2012年10月1日* — [比较文件](https://github.com/documentcloud/underscore/compare/1.4.1...1.4.2)

为了保证向下兼容, 恢复了 1.4.0 候选版时的一些特性 当传 null 到迭代函数时. 现在又变回非可选参数了.

**1.4.1** — *Oct. 1, 2012* — [比较文件](https://github.com/documentcloud/underscore/compare/1.4.0...1.4.1)

修复 1.4.0 版本里 lastIndexOf 函数的退化.

**1.4.0** — *Sept. 27, 2012* — [比较文件](https://github.com/documentcloud/underscore/compare/1.3.3...1.4.0)

增加 pairs 函数, 把一个 JavaScript 对象转换成 [key, value] 的组合 ... 同样地, 也有 object 函数, 把 [key, value] 的数组组合转换成对象.

增加 countBy 函数, 可以计算数组内符合条件的对象个数.

增加 invert 函数, 在对象里实现一个简单的键值对调.

增加 where 函数, 以便于筛选出一个数组里包含指定键值的对象数组.

增加 omit 函数, 可以过滤掉对象里的对应key的属性.

增加 random 函数, 生成指定范围内的随机数.

用 \_.debounce 创建的函数现在会返回上一次更新后的值, 就像 \_.throttle 加工过的函数一样.

sortBy 函数现在使用了稳定的排序算法.

增加可选参数 fromIndex 到 indexOf 和 lastIndexOf 函数里.

Underscore 的迭代函数里不再支持稀疏数组. 请使用 for 循环来代替 (或者会更好).

min 和 max 函数现在可以用在 *非常*大的数组上.

模板引擎里插入变量现在可以使用 null 和 undefined 作为空字符串.

Underscore 的迭代函数不再接受

一些小幅修复和调整, 可以在此查看与之前版本的 [比较](https://github.com/documentcloud/underscore/compare/1.3.3...1.4.0). **1.4.0** 可能比较不向下兼容, 这取决于您怎么使用Underscore — 请在升级后进行测试。

**1.3.3** — *2012年4月10日*

\_.template的多处改进, 现在为潜在的更有效的服务器端预编译 提供模板的源(source)作为属性. 您现在也可以在创建模板的时候 设置 variable 选项, 之后可以通过这个变量名取到模板传入的数据, 取代了 with 语句 — 显著的改进了模板的渲染速度.

增加了 pick 函数, 它可以过滤不在所提供的白名单之内的其他属性.

增加 result 函数, 在与API工作时很方便, 允许函数属性或原始属性(非函数属性).

增加 isFinite 函数, 因为有时候仅仅知道某变量是一个 数的时候还不够, 还要知道它是否是有限的数.

sortBy 函数现在可以传属性名作为对象的排序标准.

修复 uniq 函数, 现在可以在稀疏数组上使用了.

difference 函数现在在对比数组差异的时候只执行浅度的flatten, 取代之前的深度flatten.

debounce 函数现在多了一个参数 immediate, 会影响到达时间间隔后执行的是最先的函数调用还是最后的函数调用.

**1.3.1** — *2012年1月23日*

增加 \_.has 函数, 作为 hasOwnProperty 更安全的版本.

增加 \_.collect , 作为 \_.map 的别名.

恢复一个旧的修改, \_.extend 将再次可以正确复制 拥有undefined值的属性.

修复在 \_.template 的嵌入语句里反转义斜杠的bug.

**1.3.0** — *2012年1月11日*

移除Underscore对AMD(RequireJS)的支持. 如果您想继续在 RequireJS里使用Underscore, 可以作为一个普通的script加载, 封装或修改您的Underscore副本, 或者下载一个Underscore别的fork版本.

**1.2.4** — *Jan. 4, 2012*

您现在可以写 (您应该会这样用, 因为这样更简单) \_.chain(list) 来代替 \_(list).chain().

修复已反转义的字符在Underscore模板里的错误, 并增加了支持自定义支持, 使用\_.templateSettings, 只需要定义一到两个必备的正则表达式.

修复以数组作为第一参数传给\_.wrap函数的错误.

改进与ClojureScript的兼容性, 增加call 函数到String.prototype里.

**1.2.3** — *2011年12月7日*

动态范围在已编译的 \_.template 函数中保留, 所以您可以使用 this 属性, 如果您喜欢的话.

\_.indexOf 和 \_.lastIndexOf 增加对稀疏数组的支持.

\_.reduce 和 \_.reduceRight 现在都可以传一个明确的 undefined 值. (您为什么要这样做并没有任何原因)

**1.2.2** — *2011年11月14日*

继续改进 \_.isEqual , 要让它和语义上所说的一样. 现在原生的JavaScript会一个对象与它的封装起来的对象视为相等的, 还有, 数组只会对比他们数字元素 (#351).

\_.escape 不再尝试在非双重转义的转义HTML实体上进行转换. 现在不管怎样只会反转义一次 (#350).

在 \_.template 里, 如果愿意的话您可以省略嵌入表达式后面的分号: <% }) %> (#369).

\_.after(callback, 0) 现在会立即触发callback函数, 把"after"做得更易于使用在异步交互的API上 (#366).

**1.2.1** — *2011年10月24日*

\_.isEqual 函数的几个重要bug修复, 现在能更好地用在复杂的数组上, 和拥有 length 属性的非数组对象上了. (#329)

**jrburke** 提供了导出Underscore以便AMD模块的加载器可以加载, 还有 **tonylukasavage** 提供了导出Underscore给Appcelerator Titanium使用. (#335, #338)

您现在可以使用 \_.groupBy(list, 'property') 作为 以指定的共同属性来分组的快捷方法.

\_.throttle 函数现在调用的时候会立即自行一次, 此后才是再每隔指定时间再执行一次 (#170, #266).

大多数 \_.is[类型] 函数不再使用ducktype写法(详见Ruby的duck type).

\_.bind 函数现在在构造函数(constructor)也能用了, 兼容ECMAScript 5标准. 不过您可能永远也用不到 \_.bind 来绑定一个构造函数.

\_.clone 函数不再封装对象里的非对象属性.

\_.find 和 \_.filter 现在作为 \_.detect 和 \_.select 的首选函数名.

**1.2.0** — *2011年10月5日*

\_.isEqual 函数现在支持深度相等性对比, 检测循环结构, 感谢Kit Cambridge.

Underscore模版现在支持嵌入HTML转义字符了, 使用 <%- ... %> 语句.

Ryan Tenney 提供了 \_.shuffle 函数, 它使用 Fisher-Yates算法的修改版, 返回一个乱序后的数组副本.

\_.uniq 现在可以传一个可选的迭代器iterator, 用来确定一个数组以什么样的标准来确定它是否唯一的.

\_.last 现在增加了一个可选参数, 可以设置返回集合里的最后N个元素.

增加了一个新函数 \_.initial, 与 \_.rest 函数相对, 它会返回一个列表除了最后N个元素以外的所有元素.

**1.1.7** — *2011年7月13日*  
增加 \_.groupBy, 它可以将一个集合里的元素进行分组. 增加 \_.union 和 \_.difference, 用来补充 (重命名过的) \_.intersection 函数. 多方面的改进以支持稀疏数组. \_.toArray 现在如果直接传数组时, 将会返回此数组的副本.\_.functions 现在会返回存在于原型链中的函数名.

**1.1.6** — *2011年4月18日*  
增加 \_.after 函数, 被它改造过的函数只有在执行指定次数之后才会生效. \_.invoke 现在将使用函数的直接引用.\_.every 现在必须传如迭代器函数, 为了符合ECMAScript 5标准. \_.extend 当值为undefined的时候不再复制键值.\_.bind 现在如果试图绑定一个undefined值的时候将报错.

**1.1.5** — *2011年3月20日*  
增加 \_.defaults 函数, 用来合并JavaScript对象, 一般用来做生成默认值使用. 增加 \_.once 函数, 用来把函数改造成只能运行一次的函数. \_.bind 函数现在委托原生的ECMAScript 5版本(如可用). \_.keys 现在传非对象的值时, 将会抛出一个错误, 就和ECMAScript 5标准里的一样. 修复了 \_.keys 函数在传入稀疏数组时的bug.

**1.1.4** — *2011年1月9日*  
改进所有数组函数当传值 null 时候的行为, 以符合ECMAScript 5标准. \_.wrap 函数现在能正确地 给封装的函数设置 this 关键字了. \_.indexOf 函数增加了可选参数isSorted, 寻找索引的时候会将数组作为已排序处理, 将使用更快的二进制搜索. 避免使用 .callee, 保证 \_.isArray 函数 在ECMAScript 5严格模式下能正常使用.

**1.1.3** — *2010年12月1日*  
在CommonJS里, Underscore可以像这样引入:   
var \_ = require("underscore"). 增加 \_.throttle 和 \_.debounce 函数. 移除 \_.breakLoop 函数, 为了符合ECMAScript 5标准里所说的每一种实现形式都是不能*break*的 — 这将去掉try/catch块, 现在, 您遇到Underscore迭代器的抛出的异常时, 将会有更完善的堆栈跟踪来检查错误所在之处. 改进 **isType** 一类函数, 以便更好地兼容Internet Explorer浏览器. \_.template 函数现在可以正确的反转义模板中的反斜杠了. 改进 \_.reduce 函数以兼容ECMAScript 5标准: 如果您不传初始值, 将使用集合里的第一项作为初始值. \_.each 不再返回迭代后的集合, 为了与ECMAScript 5的 forEach 保持一致.

**1.1.2**  
修复 \_.contains 指向 \_.intersect 函数的错误, 应该是指向 \_.include 函数(\_.cotains应该是\_.include的别名), 增加 \_.unique, 作为 \_.uniq 函数的别名.

**1.1.1**  
改进 \_.template 函数的运行速度, 和处理多行插入值的性能. Ryan Tenney 提供了许多Underscore函数的优化方案. 增加了带注释版本的源代码.

**1.1.0**  
修改了 \_.reduce 函数以符合ECMAScript 5规范, 取代了之前Ruby/Prototype.js版本的 \_.reduce. 这是一个不向下兼容的修改. \_.template 函数现在可以不传参了, 并保留空格. \_.contains 是一个 \_.include 函数新的别名.

**1.0.4**  
[Andri Möll](http://themoell.com/) 提供了 \_.memoize函数, 以缓存计算结果, 来优化的耗时较长的函数, 使得运行速度变快.

**1.0.3**  
修复了 \_.isEqual 函数在对比包含 NaN 的对象时返回 false 的问题. 技术上改良后理论上是正确的, 但是语义上似乎有矛盾, 所以要注意避免对比含有NaN的对象.

**1.0.2**  
修复 \_.isArguments 在新版本Opera浏览器里的bug, Opera里会把arguments对象当作数组.

**1.0.1**  
修复了 \_.isEqual 函数的bug: 这个bug出现在当对比特定因素两个对象时, 这两个对象有着相同个数的值为undefined的key, 但不同名.

**1.0.0**  
Underscore在这几个月里算是相对稳定了, 所以现在打算出测试版, 版本号为**1.0**. 从**0.6**版本开始进行改进, 包括\_.isBoolean的改进, 和\_.extend允许传多个source对象.

**0.6.0**  
主要版本, 整合了一系列的功能函数, 包括 [Mile Frawley](http://github.com/ratbeard)写的在保留援用功能的基础上, 对集合函数进行重构, 内部代码更加简洁. 新的 \_.mixin 函数, 允许您自己的功能函数继承Underscore对象. 增加 \_.times 函数, 跟Ruby或Prototype.js里的times的功能一样. 对ECMAScript 5的 Array.isArray函数提供原生支持, 还有Object.keys.

**0.5.8**  
修复了Underscore的集合函数, 以便可以用于DOM的 [节点列表(NodeList)](https://developer.mozilla.org/En/DOM/NodeList) 和 [HTML集合(HTMLCollection)](https://developer.mozilla.org/En/DOM/HTMLCollection) 再一次地感谢 [Justin Tulloss](http://github.com/jmtulloss).

**0.5.7**  
修改 \_.isArguments 函数, 使用了更安全的实现方式, 还有 加快了 \_.isNumber 的运行速度,  
感谢 [Jed Schmidt](http://jedschmidt.com/).

**0.5.6**  
增加了 \_.template 对自定义分隔符的支持, 由 [Noah Sloan](http://github.com/iamnoah)提供.

**0.5.5**  
修复了一个在移动版Safari里关于arguments对象的面向对象封装的bug.

**0.5.4**  
修复了\_.template函数里多个单引号在模板里造成的错误. 了解更多请阅读: [Rick Strahl的博客文章](http://www.west-wind.com/Weblog/posts/509108.aspx).

**0.5.2**  
几个函数的重写: isArray, isDate, isFunction, isNumber, isRegExp, 和 isString, 感谢[Robert Kieffer](http://www.broofa.com/)提供的建议. 取代了 Object#toString 的对比方式, 现在以属性来进行对比, 虽然说安全性有所降低, 但是速度比以前快了有一个数量级. 因此其他大多数的Underscore函数也有小幅度的速度提升. 增加了 \_.tap 函数, 由[Evgeniy Dolzhenko](http://dolzhenko.org/) 提供, 与[Ruby 1.9](http://ruby-doc.org/core-1.9.3/Object.html#method-i-tap)的tap方法相似, 对链式语法里嵌入其他功能(如登录)很方便.

**0.5.1**  
增加了 \_.isArguments 函数. 许多小的安全检查和优化由 [Noah Sloan](http://github.com/iamnoah/) 和 [Andri Möll](http://themoell.com/)提供.

**0.5.0**  
**[API变更]** \_.bindAll 现在会将context对象作为第一个参数. 如果不传方法名, context对象的所有方法都会绑定到context, 支持链式语法和简易绑定. \_.functions 现在只要一个参数, 然后返回所有的方法名(类型为Function的属性). 调用 \_.functions(\_) 会列出所有的Underscore函数. 增加 \_.isRegExp 函数, isEqual 现在也可以检测两个RegExp对象是否相等了. 所有以"is"开头的函数已经缩减到同一个定义里面, 由[Karl Guertin](http://github.com/grayrest/) 提供的解决方案.

**0.4.7**  
增加 isDate, isNaN, 和 isNull. 优化 isEqual 函数对比两个数组或两个时间对象时的性能. 优化了 \_.keys 函数, 现在的运行速度比以前加快了***25%–2倍*** (取决于您所使用的浏览器)会加速其所依赖的函数, 如 \_.each.

**0.4.6**  
增加 range 函数, [Python里同名函数range](http://docs.python.org/library/functions.html#range) 的移植版, 用于生成灵活的整型数组. 原始版由[Kirill Ishanov](http://github.com/kylichuku)提供.

**0.4.5**  
增加 rest 函数, 可以对数组和arguments对象使用, 增加了两个函数的别名, first 的别名为 head, 还有 rest 的别名为 tail, 感谢 [Luke Sutton](http://github.com/lukesutton/)的解决方案. 增加测试文件, 以确保所有Underscore的数组函数都可以在用在 *arguments* 对象上.

**0.4.4**  
增加 isString, 和 isNumber 函数. 修复了 \_.isEqual(NaN, NaN) 会返回 *true* 的问题.

**0.4.3**  
开始使用原生的 StopIteration 浏览器对象(如果浏览器支持). 修复Underscore在CommonJS环境上的安装.

**0.4.2**  
把解除封装的函数unwrapping改名为value, 更清晰.

**0.4.1**  
链式语法封装的Underscore对象支持函数原型方法的调用, 您可以在封装的数组上连续调用任意函数. 增加 breakLoop方法, 可以随时在Underscore的迭代中 **中断** 并跳出迭代. 增加 isEmpty 函数, 在数组和对象上都有用.

**0.4.0**  
现在所有的Underscore函数都可以用面向对象的风格来调用了, 比如: \_([1, 2, 3]).map(...);. [Marc-André Cournoyer](http://macournoyer.com/)提供了原始的解决方案. 封装对象可以用链式语法连续调用函数. 添加了 [functions](https://www.css88.com/doc/underscore1.8.2/#object-functions) 方法, 能以正序方式列出所有的Underscore函数.

**0.3.3**  
增加JavaScript 1.8的函数 reduceRight. 别名为 foldr, 另外 reduce 的别名为 foldl.

**0.3.2**  
可以在 [Rhino](http://www.mozilla.org/rhino/) 上运行了. 只要在编译器里输入: load("underscore.js"). 增加功能函数 [identity](https://www.css88.com/doc/underscore1.8.2/#identity).

**0.3.1**  
所有迭代器在原始集合里现在都作为第三个参数传入, 和JavaScript 1.6的 **forEach** 一致. 迭代一个对象现在会以 (value, key, collection) 来调用, 更多详情, 请查看 [\_.each](https://www.css88.com/doc/underscore1.8.2/#each).

**0.3.0**  
增加 [Dmitry Baranovskiy](http://github.com/dmitryBaranovskiy)的 综合优化, 合并 [Kris Kowal](http://github.com/kriskowal/)的解决方案让Underscore符合 [CommonJS](http://wiki.commonjs.org/wiki/CommonJS) 标准,并和 [Narwhal](http://narwhaljs.org/)兼容.

**0.2.0**  
添加 compose 和 lastIndexOf, 重命名 inject 为 reduce, 添加 inject, filter, every, some, 和 forEach 的别名.

**0.1.1**  
添加 noConflict, 以便 "Underscore" 对象可以分配给其他变量.

**0.1.0**  
Underscore.js 首次发布.