# lodash

## [JavaScript工具库之Lodash](http://www.cnblogs.com/whitewolf/p/4417873.html)

2015-04-11 16:08 by 破狼, 9583 阅读, 2 评论, [收藏](http://www.cnblogs.com/whitewolf/p/4417873.html), [编辑](http://i.cnblogs.com/EditPosts.aspx?postid=4417873)

你还在为JavaScript中的数据转换、匹配、查找等烦恼吗？一堆看似简单的foreach，却冗长无趣，可仍还在不停的repeat it！也许你已经用上了Underscore.js，不错，你已经进步很大一步了。然而今天我希望你能更进一步，利用lodash替换掉Underscore。

lodash一开始是Underscore.js库的一个fork，因为和其他(Underscore.js的)贡献者意见相左。John-David Dalton的最初目标，是提供更多“一致的跨浏览器行为……，并改善性能”。之后，该项目在现有成功的基础之上取得了更大的成果。最近lodash也发布了3.5版，成为了npm包仓库中依赖最多的库。它正在摆脱屌丝身份，成为开发者的常规的选择之一。

现在我们所熟知的很多开源项目都已经使用或者转到了lodash阵营之上。比如JavaScript转译器Babel、博客平台Ghost，和项目脚手架工具Yeoman。特别Ghost是从Underscore迁移到了lodash，Ghost的创始人John O’Nolan对于此曾评价到：“这是一个非常明智的选择，它几乎完全是由我们开源开发社区推动的。我们发现lodash包含更多的功能，更好的性能、恰到好处地使用了semver，并且在Node.js社区（以及其他依赖）中越来越抢眼“。

## lodash演练

lodash主要使用了延迟计算，使得lodash其性能远远超过Underscore。在lodash中延迟计算意味着在我们的链式方法在显示或隐式的value()调用之前是不会执行的。由于这种执行的延后，因此lodash可以进行shortcut fusion这样的优化，通过合并链式iteratee大大降低迭代的次数。从而大大提供其执行性能。

百说不如一练，下面我们以用户信息为例：

var users = [

{ 'user': 'barney', 'age': 36 },

{ 'user': 'fred', 'age': 40 },

{ 'user': 'pebbles', 'age': 18 }

];

#### 1.获取所有用户名字，并以”，“分割

var names = \_.chain(users)

.map(function(user){

return user.user;

})

.join(" , ")

.value();

console.log(names);

个人比较喜欢lodash延迟计算的现实value，以及JavaScript的函数式风格。在这里首先将users对象包装成为lodash对象，再map获取所有用户的名称，并最后利用join将用户名称以”，“连接在一起。注意这里只是一串方法链，如果你没有显样的调用value方法，使其立即执行的化，你将会得到如下的LodashWrapper延迟表达式：

LodashWrapper {\_\_wrapped\_\_: LazyWrapper, \_\_actions\_\_: Array[1], \_\_chain\_\_: true, constructor: function, after: function…}

因为延迟表达式的存在，因此我们可以多次增加方法链，但这并不会被执行，所以不会存在性能的问题，最后知道我们需要使用的时候，使用value显式立即执行即可。

#### 2. 获取最年轻的用户

var youngest = \_.chain(users)

.min(function(user){

return user.age;

})

.value();

console.log(youngest);

这里利用了lodash提供的min函数可以轻易的解决。

在这里博主还希望用另外一个方式解释lodash方法链的优化,上面的方法可以等价为下面的方式，以age排序的第一个user：

var youngest2 = \_.chain(users)

.sortBy("age")

.map(function(user){

console.log("map", user);

return user;

})

.first()

.value();

console.log(youngest2);

在这里博主多加了一个map作为log输出，如果你执行这行代码的时候，你会惊奇的看见这里只会有一个user的输出，这点可以证明在立即执行的时候lodash为我们的方法链做了可靠的优化；如果我们去掉first函数你则会看见有3个user对象的输出。

#### 3. 获取最年长的用户

var oldest = \_.chain(users)

.max(function(user){

return user.age;

})

.value();

console.log(oldest );

这里则使用lodash的max函数。

#### 4. 用户数组到用户Map的转换

在开发中我们经常会有把一堆素组形式的数据转换为Object形式的数组，便于根据属性key值查找，下面将以user对象来演示：

var userObj = \_.chain(users)

.map(function(user){

return [user.user, user.age];

})

.zipObject()

.value();

console.log(userObj);

利用lodash首先将user数组map为[key, value]的数组集合，最后利用zipObject将结果转换为Object对象，zipObject会利用结果集的第一项作为key，第二项作为value生产Object。

## 结尾

我们在这里展示知识lodash中很小一部分的API，正如随笔开始所说：lodash是为了提供更多“一致的跨浏览器行为……，并改善性能”API。所有的lodash API你可以在这里[https://lodash.com/docs#matches](https://lodash.com/docs" \l "matches)查找。

本文的所演示的demo，你也可以在jsbin <http://jsbin.com/xocixubaru/1/edit?html,js,output>演示。

# lodash学习笔记之Array方法

今天周末在家无聊学习一下lodash. lodash目前的中文资料很少。而且api好像还被墙了。下面说一下lodash的arrary相关的方法。

1. chunk   英 [tʃʌŋk]    顾名思义，是对数组进行分块的方法

* n. 大块；矮胖的人或物

    用法： \_.chunk(array,number)  根据number对array进行均等的分块，如果array不能被number平分，则会留下一个余下的块。

[复制代码](javascript:void(0);)

\_.chunk(['a','b','c','d'],-1);

//当 size<=1的时候，都是按1等分

> ['a','b','c','d']

//size=2

>[['a','b'],['c','d']]

//size=3

>[['a','b','c'],['d']]

//size>=4

>['a','b','c','d']

//不能这么传参数

\_.chunk('a', 'b', 'c', 'd', 2)

>['a']

[复制代码](javascript:void(0);)

2. compact     去除假值

* adj. 紧凑的，紧密的；简洁的
* vt. 使简洁；使紧密结合

    api: Creates an array with all falsey values removed. The values false, null, 0, "", undefined, and NaN are falsey.

    用法：\_.compact(array)

[复制代码](javascript:void(0);)

//很明显第一个参数被处理了，剩下的参数都被忽视了。

\_.compact('a','b','');

>["a"]

\_.compact(['a','b','']);

>["a", "b"]

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

>[1, 2, 3]

[复制代码](javascript:void(0);)

3. difference  从数组中过滤元素

### 用法：\_.difference(array,[values])

   参数说明： array:要被检查/过滤的数组。

                  values:要被在array中剔除的值的集合

[复制代码](javascript:void(0);)

//注意两个参数都应该是数组类型  
\_.difference([1,2,4],2)

[1, 2, 4]

\_.difference([1,2,4],[2])

[1, 4]

\_.difference([1,2,4],[-1])

[1, 2, 4]  
\_.difference([1,2,4],[1,2,4])  
[]

[复制代码](javascript:void(0);)

 4. drop  数组元素删除

     用法：类似于原生js方法中的slice   \_.drop(array,number)

     从头开始删除number个数组元素。number不传的话默认按1处理

[复制代码](javascript:void(0);)

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

// → [2, 3]

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

// → [3]

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

// → []

\_.drop([1, 2, 3], 0);

// → [1, 2, 3]

[复制代码](javascript:void(0);)

5. dropRight    数组元素删除

    用法几乎和drop一样，不同的是从数组末尾开始删除。

6. dropRightWhile   数组元素过滤

    用法 \_.dropRightWhile(array,[predicate=\_.identity],[thisArg])

   -- Creates a slice of array excluding elements dropped from the end.

   -- Elements are dropped until predicate returns false

   -- The predicate is bound to thisArg and invoked with three arguments: (value, index, array).

    参数1：待处理的数组

    参数2：可以是(Function|Object|string)，会对数组的每个元素调用 。

    参数3：判断是否删除的谓词。

\_.dropRightWhile([1, 2, 3], function(n) {

return n > 1;

});

// → [1]

[复制代码](javascript:void(0);)

var users = [

{ 'user': 'barney', 'active': true },

{ 'user': 'fred', 'active': false },

{ 'user': 'pebbles', 'active': false }

];

// using the `\_.matches` callback shorthand

\_.pluck(\_.dropRightWhile(users, { 'user': 'pebbles', 'active': false }), 'user');

// → ['barney', 'fred']

// using the `\_.matchesProperty` callback shorthand

\_.pluck(\_.dropRightWhile(users, 'active', false), 'user');

// → ['barney']

// using the `\_.property` callback shorthand 此处的解释应该是要drop不存在active属性的对象。

\_.pluck(\_.dropRightWhile(users, 'active'), 'user');

// → ['barney', 'fred', 'pebbles']

[复制代码](javascript:void(0);)

 刚开始看的时候对第三条有点迷糊。怎么会一个都没有过滤掉呢？  查看了一下api.

 参数predicate实际上是有几种可能的值类型的，根据参数predicate的值类型的不同，会有如下几种不同的处理：

 1. function  此种情况下, 如果函数返回true,会把这一项drop掉。这种情况下函数一般只有两个参数：array和function

 2. string    如果参数predicate是一个属性名（string类型）的话，则返回值将会是此次遍历此属性的value。然后根据value进行drop。

  并且如果参数3 thisArg也有值的话，则会比较thisArg和predicate的返回值的不同。根据比较的值来进行drop。

  API:  If a property name is provided for predicate  the created \_.property style callback returns the property value of the given element.

  If a value is also provided for thisArg the created \_.matchesProperty  style callback returns true for elements that have a matching property value, else false.

 3. object  此种情况下。如果array中的某一项的属性和object中的属性一致，则返回true,否则就返回false.

  API: If an object is provided for predicate  the created \_.matches style callback returns true for elements that match the properties of the given object, else false.

  在测试的过程中，发现一个奇怪的例子：

var obj=[{'a':0,'b':'sa'},{'a':2,'b':'sadff'},{'a':3,'b':21}]; \_.pluck(\_.dropRightWhile(obj,'a',0),'a');

[0, 2, 3]

 7. dropWhile  数组元素过滤

    和dropRightWhile是基本一致的，不同点是从头到尾来进行计算的。

 8. fill  数组元素填充

     用法：  \_.fill(array, value, [start=0], [end=array.length])

     从开始参数到结束参数，用value来替代或者填补数组元素。因为数组的下标是从0开始的，所以填充的范围是个左闭右开区间-填充的index范围包括start而不包括end.

     注意：此方法直接改变array，而不是返回一个数组。

[复制代码](javascript:void(0);)

var array = [1, 2, 3];

\_.fill(array, 'a');

console.log(array);

// → ['a', 'a', 'a']

\_.fill(Array(3), 2);

// → [2, 2, 2]

\_.fill([4, 6, 8], '\*', 1, 2);

// → [4, '\*', 8]

[复制代码](javascript:void(0);)

9. findIndex  查询元素序号，遍历数组，如果查询到了符合要求的第一个元素则返回序号，如果没查询到符合要求的元素则返回-1.

    用法： \_.findIndex(array, [predicate=\_.identity], [thisArg])   \_.identity()方法返回传给它的第一个参数。

[复制代码](javascript:void(0);)

var users = [

{ 'user': 'barney', 'active': false },

{ 'user': 'fred', 'active': false },

{ 'user': 'pebbles', 'active': true }

];

\_.findIndex(users, function(chr) {

return chr.user == 'barney';

});

// → 0

// using the `\_.matches` callback shorthand

\_.findIndex(users, { 'user': 'fred', 'active': false });

// → 1

// using the `\_.matchesProperty` callback shorthand

\_.findIndex(users, 'active', false);

// → 0

// using the `\_.property` callback shorthand

\_.findIndex(users, 'active');

// → 2

[复制代码](javascript:void(0);)

10. findLastIndex 类似于findIndex,只不过其返回的序列号是符合要求的最后一个。

      用法：\_.findLastIndex(array, [predicate=\_.identity], [thisArg])

11. first  返回数组第一个元素.

     用法：\_.first(array)

     没什么好说的，如果数组为[]则返回undefined。

12. flatten 抹平嵌套数组

### 用法：\_.flatten(array, [isDeep])

    isDeep为空或者false的情况下，只抹平第一层嵌套。为true的情况下，递归的进行抹平。

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

// → [1, 2, 3, [4]]

// using `isDeep`

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

// → [1, 2, 3, 4]

13. flattenDeep 递归的抹平嵌套数组

### 用法：\_.flattenDeep(array)

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

// → [1, 2, 3, 4]

14. indexOf

用法：\_.indexOf(array, value, [fromIndex=0])  从数组array中查询value的序号，参数3如果是true的话，执行二分查找。

[复制代码](javascript:void(0);)

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

// → 1

// using `fromIndex`

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

// → 3

// performing a binary search

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

// → 2

[复制代码](javascript:void(0);)

15.initial  返回除了末尾元素的数组

用法：\_.initial(array)

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

// → [1, 2]

16. intersection 返回新数组，其值就是数组参数的交集

### 用法：\_.intersection([arrays])

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

// → [2]

17. last 返回参数数组的末尾元素

### 用法：\_.last(array)

18. lastIndexOf  类似于indexOf,搜索方向为从末尾到开头

### 用法：\_.lastIndexOf(array, value, [fromIndex=array.length-1])

[复制代码](javascript:void(0);)

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

// → 3

// using `fromIndex`

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

// → 1

// performing a binary search

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

// → 3

[复制代码](javascript:void(0);)

19.pull 移除值，直接在原数组上进行操作

用法：\_.pull(array, [values])

var array = [1, 2, 3, 1, 2, 3];

\_.pull(array, 2, 3);

console.log(array);

// → [1, 1]

20. pullAt 按序号移除值,直接操作原数组并且返回移除的值组成的数组。

### 用法：\_.pullAt(array, [indexes])

[复制代码](javascript:void(0);)

var array = [5, 10, 15, 20];

var evens = \_.pullAt(array, 1, 3);

console.log(array);

// → [5, 15]

console.log(evens);

// → [10, 20]

[复制代码](javascript:void(0);)

可以看出来，移除1,3位置的元素从逻辑上来说是同时移除的。避免了数组越界的问题。

21.remove 移除元素,对原数组进行操作，并且返回移除元素的集合。

用法：\_.remove(array, [predicate=\_.identity], [thisArg])

从参数可以看出来，参数的处理逻辑是类似于前面的dropRightWhile方法的。

API:Removes all elements from array that predicate returns truthy for and returns an array of the removed elements. The predicate is bound to thisArg and invoked with three arguments: (value, index, array).

[复制代码](javascript:void(0);)

var array = [1, 2, 3, 4];

var evens = \_.remove(array, function(n) {

return n % 2 == 0;

});

console.log(array);

// → [1, 3]

console.log(evens);

// → [2, 4]

[复制代码](javascript:void(0);)

22. rest 移除数组首元素 和initial相反

用法：\_.rest(array)

23.slice 数组截取

### 用法：\_.slice(array, [start=0], [end=array.length])

那么和原生的slice有什么不同呢？

API:This method is used instead of ***Array#slice*** to support node lists in IE < 9 and to ensure dense arrays are returned.

24.sortedIndex 在对一个有序数组array进行插入的时候，返回value应该插入的位置。从左向右计算。

### 用法：\_.sortedIndex(array, value, [iteratee=\_.identity], [thisArg])

 API:Uses a binary search to determine the lowest index at which ***value***should be inserted into ***array***in order to maintain its sort order. If an iteratee function is provided it is invoked for ***value***and each element of ***array***to compute their sort ranking. The iteratee is bound to ***thisArg***and invoked with one argument; (value).

[复制代码](javascript:void(0);)

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

// → 1

\_.sortedIndex([4, 4, 5, 5], 5);

// → 2

var dict = { 'data': { 'thirty': 30, 'forty': 40, 'fifty': 50 } };

// using an iteratee function

\_.sortedIndex(['thirty', 'fifty'], 'forty', function(word) {

return this.data[word];

}, dict);

// → 1

// using the `\_.property` callback shorthand

\_.sortedIndex([{ 'x': 30 }, { 'x': 50 }], { 'x': 40 }, 'x');

// → 1

[复制代码](javascript:void(0);)

25. sortedLastIndex 用法类似于sortedindex,不同的是从右至左计算插入的位置

用法：\_.sortedLastIndex(array, value, [iteratee=\_.identity], [thisArg])

\_.sortedLastIndex([4, 4, 5, 5], 5);

// → 4

26. take 数组切片

### 用法：\_.take(array, [n=1])

API：Creates a slice of ***array***with ***n***elements taken from the beginning.

[复制代码](javascript:void(0);)

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

// → [1]

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

// → [1, 2]

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

// → [1, 2, 3]

\_.take([1, 2, 3], 0);

// → []

[复制代码](javascript:void(0);)

27. takeRight 类似于take方法，执行方向不同。

### 用法：\_.takeRight(array, [n=1])

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

// → [3]

28. takeRightWhile

### 用法：\_.takeRightWhile(array, [predicate=\_.identity], [thisArg])

*API：Creates a slice of****array****with elements taken from the end. Elements are taken until****predicate****returns falsey. The predicate is bound to****thisArg****and invoked with three arguments: (value, index, array).*

[复制代码](javascript:void(0);)

\_.takeRightWhile([1, 2, 3], function(n) {

return n > 1;

});

// → [2, 3]

var users = [

{ 'user': 'barney', 'active': true },

{ 'user': 'fred', 'active': false },

{ 'user': 'pebbles', 'active': false }

];

// using the `\_.matches` callback shorthand

\_.pluck(\_.takeRightWhile(users, { 'user': 'pebbles', 'active': false }), 'user');

// → ['pebbles']

// using the `\_.matchesProperty` callback shorthand

\_.pluck(\_.takeRightWhile(users, 'active', false), 'user');

// → ['fred', 'pebbles']

// using the `\_.property` callback shorthand

\_.pluck(\_.takeRightWhile(users, 'active'), 'user');

// → []

[复制代码](javascript:void(0);)

29. takeWhile 类似于takeRightWhile 执行顺序相反

### 用法：\_.takeWhile(array, [predicate=\_.identity], [thisArg])

30.union 数组合并，去除重复值

用法：\_.union([arrays])

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

// → [1, 2, 4]

31.uniq/unique 数组去重

### 用法：\_.uniq(array, [isSorted], [iteratee], [thisArg])

*API:Creates a duplicate-free version of an array, using****SameValueZero****for equality comparisons, in which only the first occurence of each element is kept. Providing****true****for****isSorted****performs a faster search algorithm for sorted arrays. If an iteratee function is provided it is invoked for each element in the array to generate the criterion by which uniqueness is computed. The****iteratee****is bound to****thisArg****and invoked with three arguments: (value, index, array).*

[复制代码](javascript:void(0);)

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

// → [2, 1]

// using `isSorted`

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

// → [1, 2]

// using an iteratee function

\_.uniq([1, 2.5, 1.5, 2], function(n) {

return this.floor(n);

}, Math);

// → [1, 2.5]

// using the `\_.property` callback shorthand

\_.uniq([{ 'x': 1 }, { 'x': 2 }, { 'x': 1 }], 'x');

// → [{ 'x': 1 }, { 'x': 2 }]

[复制代码](javascript:void(0);)

32. unzip  zip的逆运算，还原zip后的数组

### 用法：\_.unzip(array)

[复制代码](javascript:void(0);)

var zipped = \_.zip(['fred', 'barney'], [30, 40], [true, false]);

// → [['fred', 30, true], ['barney', 40, false]]

\_.unzip(zipped);

// → [['fred', 'barney'], [30, 40], [true, false]]  
  
var zipped2=\_.zip(['fre','shike'],[30,40,50],[true,false]);  
// → [['fred', 30, true], ['barney', 40, false],[undefined,50,undefined]]

[复制代码](javascript:void(0);)

33. unzipWith  在数组重组的时候同时进行某些操作

### 用法：\_.unzipWith(array, [iteratee], [thisArg])

### *[iteratee]* (Function): The function to combine regrouped values.

var zipped = \_.zip([1, 2], [10, 20], [100, 200]);

// → [[1, 10, 100], [2, 20, 200]]

\_.unzipWith(zipped, \_.add);

// → [3, 30, 300]

可以看出来，在unZip之后，因为iteratee是\_.add函数，因此将unZip的结果进行了相加。

34. without  从数组中去除某些值

### 用法：\_.without(array, [values])

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

// → [3]

不同于difference方法。其values参数可以不是一个数组，而是接在array参数之后的零散参数。

35.xor  对称消除重复值

API:Creates an array that is the symmetric difference of the provided arrays.

symmetric:对称的，均匀的。

### 用法：\_.xor([arrays])

[复制代码](javascript:void(0);)

\_.xor([1, 2], [4, 2]);  
// [1, 4]

  \_.xor([1,2],[3,4])  
  // [1, 2, 3, 4]  
  \_.xor([1,2],[3,4,1])  
  // [2, 3, 4]  
  \_.xor([1,2],[1,2])  
  // []

[复制代码](javascript:void(0);)

如果更进一步探究：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | \_.xor([1,2],[1,2],[1])  [1]  \_.xor([1,2],[1,2],[3,4])  [3, 4]  \_.xor([1,2],[1,2],[1,4])  [1, 4]  \_.xor([1,2],[1,2],[1,4],[1,4])  []  \_.xor([1,2],[1,2],[3,4,1])  [3, 4, 1] |

是不是很有趣？ 推测如果前面的数组参数两两消除了，后面的数组即使有重复的元素，还是会保留。

.xor([1,2],[1,2],[1,2])

[1, 2]

所以说,xor这个函数应该是参数两个两个进行重复值消除的。

\_.xor([1,2],[1,2,3],[1,2])

如果n和n+1还有未消除的非重复值，那么会和n+2和n+3消除后保留下来的数组进行合并。

36. zip  数组分组

API:Creates an array of grouped elements, the first of which contains the first elements of the given arrays, the second of which contains the second elements of the given arrays, and so on.

### 用法：\_.zip([arrays])

\_.zip(['fred', 'barney'], [30, 40], [true, false]);

// → [['fred', 30, true], ['barney', 40, false]]

如果zip的数组长度不一，则会这样

\_.zip(['fred', 'barney'], [30, 40,50], [true, false]);

// → [['fred', 30, true], ['barney', 40, false],[undefined,50,undefined]]

37. zipObject  数组转对象   .\_pair的反操作，

### 用法：\_.zipObject(props, [values=[]])

API:The inverse of [*\_.pairs*](https://lodash.com/docs#pairs); this method returns an object composed from arrays of property names and values. Provide either a single two dimensional array, e.g. *[[key1, value1], [key2,value2]]* or two arrays, one of property names and one of corresponding values.

#### Arguments

1. props (Array): The property names.
2. [values=[]] (Array): The property values.

\_.zipObject([['fred', 30], ['barney', 40]]);

// → { 'fred': 30, 'barney': 40 }

\_.zipObject(['fred', 'barney'], [30, 40]);

// → { 'fred': 30, 'barney': 40 }

可以看出来，当只有一个数组参数的时候，最底层的数组被解读为[key,value]

当有两个数组参数的时候，这两个数组分别被解释为name和key的集合。 然后被组装为object返回。

38.  zipWith  类似于unzipWith函数

### 用法：\_.zipWith([arrays], [iteratee], [thisArg])

\_.zipWith([1, 2], [10, 20], [100, 200], \_.add);

// → [111, 222]

以上就是lodash v3.8的所有数组方法，对比ECMAScript 5中的数组方法。确实在一些特殊操作上简化了我们的一些特殊处理。

分类: [javascript](http://www.cnblogs.com/shikelong/category/593651.html), [lodash](http://www.cnblogs.com/shikelong/category/688054.html)

# lodash v4.6.1

## A modern JavaScript utility library delivering [modularity](https://www.npmjs.com/browse/keyword/lodash-modularized), performance, & extras.

\_.assign({ 'a'**:** 1 }**,** { 'b'**:** 2 }**,** { 'c'**:** 3 })**;**  
// → { 'a': 1, 'b': 2, 'c': 3 }  
\_.map([1**,** 2**,** 3]**,** function(n) { return n **\*** 3**;** })**;**  
// → [3, 6, 9]

## Documentation

* [API Documentation](https://lodash.com/docs)
* [FP Guide](https://github.com/lodash/lodash/wiki/FP-Guide)

## Download

Lodash is released under the [MIT license](https://raw.githubusercontent.com/lodash/lodash/4.6.1/LICENSE) & supports [modern environments](https://lodash.com/#support).  
Review the [build differences](https://github.com/lodash/lodash/wiki/build-differences) & pick one that’s right for you.

* [Core build](https://raw.githubusercontent.com/lodash/lodash/4.6.1/dist/lodash.core.js) ([~4 kB gzipped](https://raw.githubusercontent.com/lodash/lodash/4.6.1/dist/lodash.core.min.js))
* [Full build](https://raw.githubusercontent.com/lodash/lodash/4.6.1/dist/lodash.js) ([~21 kB gzipped](https://raw.githubusercontent.com/lodash/lodash/4.6.1/dist/lodash.min.js))
* [CDN copies](https://www.jsdelivr.com/projects/lodash)

## Installation

In a browser:

**<**script src**=**"lodash.js"**></**script**>**

Using npm:

$ {sudo -H} npm i -g npm  
$ npm i --save lodash

In Node.js:

// Load the full build.  
var \_ **=** require('lodash')**;**  
// Load the core build.  
var \_ **=** require('lodash/core')**;**  
// Load the fp build for immutable auto-curried iteratee-first data-last methods.  
var \_ **=** require('lodash/fp')**;**  
  
// Load a method category.  
var array **=** require('lodash/array')**;**  
var object **=** require('lodash/fp/object')**;**  
  
// Load a single method for smaller builds with browserify/rollup/webpack.  
var chunk **=** require('lodash/chunk')**;**  
var extend **=** require('lodash/fp/extend')**;**

**Note:**  
Don’t assign values to the [special variable](https://nodejs.org/api/repl.html#repl_repl_features) \_ when in the REPL.  
Install [n\_](https://www.npmjs.com/package/n_) for a REPL that includes lodash by default.

## Why Lodash?

Lodash makes JavaScript easier by taking the hassle out of working with arrays,  
numbers, objects, strings, etc. Lodash’s modular methods are great for:

* Iterating arrays, objects, & strings
* Manipulating & testing values
* Creating composite functions

## Module Formats

Lodash is available in a [variety of builds](https://lodash.com/custom-builds) & module formats.

* [lodash](https://www.npmjs.com/package/lodash) & [per method packages](https://www.npmjs.com/browse/keyword/lodash-modularized)
* [lodash-amd](https://www.npmjs.com/package/lodash-amd)
* [lodash-es](https://www.npmjs.com/package/lodash-es) & [babel-plugin-lodash](https://www.npmjs.com/package/babel-plugin-lodash)
* [lodash/fp](https://github.com/lodash/lodash/tree/4.6.1-npm/fp)

## Further Reading

* [Contributing](https://github.com/lodash/lodash/blob/4.6.1/.github/CONTRIBUTING.md)
* [Release Notes](https://github.com/lodash/lodash/releases/tag/4.0.0)
* [Wiki (Changelog, Roadmap, etc.)](https://github.com/lodash/lodash/wiki)

## Support

Tested in Chrome 48-49, Firefox 44-45, IE 9-11, Edge 13, Safari 8-9, Node.js 0.10, 0.12, 4, & 5, & PhantomJS 1.9.8.

Automated [browser](https://saucelabs.com/u/lodash) & [CI](https://travis-ci.org/lodash/) test runs are available.

# Lodash，你正在使用的JavaScript库

JavaScript工具库[lodash](https://lodash.com/)发布了3.5版，成为了[npm包仓库](https://www.npmjs.com/browse/depended)中依赖最多的库。它正在摆脱屌丝身份，成为开发者的不二之选。

lodash一开始是[Underscore.js](http://underscorejs.org/)库的一个fork，因为和其他(Underscore.js的)贡献者意见相左。[John-David Dalton](https://twitter.com/jdalton)的最初目标，是提供更多“一致的跨浏览器行为……，并改善性能”。之后，该项目在现有成功的基础之上取得了更大的成果，并于一月份发布了3.0版本。

与其前任Underscore一样，lodash的名字也是源于所有函数前面的那个字符。就像jQuery在全部函数前加全局的$一样，lodash使用全局的\_来提供对工具的快速访问。例如，要对数组的所有元素执行某个行为，我们可以：

\_.each([1, 2], function(n) { console.log(n); });

3.0版包含了新增的47个新方法（如\_.camelCase和\_.flattenDeep）在内的许多更改。更重要的是对链式方法进行延迟计算：

var users = [

{ 'user': 'barney', 'age': 36 },

{ 'user': 'fred', 'age': 40 },

{ 'user': 'pebbles', 'age': 1 }

];

var youngest = \_.chain(users)

.sortBy('age')

.map(function(chr) {

return chr.user + ' is ' + chr.age;

})

.first()

.value();

// → 'pebbles is 1'

在InfoQ的一次采访中，Dalton表示在这些方法中使用延迟计算改进了lodash的性能：

延迟计算意味着在显示或隐式的value()调用之前不会执行链式方法。由于执行被延后了，因此lodash可以进行shortcut fusion这样的优化，通过合并链式iteratee大大降低迭代的次数。

Filip Zawada[描述了这一改变](http://filimanjaro.com/blog/2014/introducing-lazy-evaluation/)对于性能的提升。

最初，lodash只是Underscore的一个复制品，但3.0之后，不会再有针对于Underscore的构建。“尽管我们仍然会运行Underscore/Backbone的单元测试，但lodash 3.0将不再支持单独的Underscore/Backbone构建”，Dalton说。

在过去的一年中，我们看到Underscore根据lodash的API进行了很多调整，因此为Underscore进行单独构建的需求减少了。如果开发者仍然需要兼容某些边缘情况，建议在使用Underscore的同时，用lodash 3.0版中的模块进行补充。

一些主要的npm包都依赖于lodash，如JavaScript转译器[Babel](https://babeljs.io/)、博客平台[Ghost](https://ghost.org/)，和项目脚手架工具[Yeoman](http://yeoman.io/)。其中Ghost是从Underscore迁移到了lodash。在被InfoQ问及在Ghost中引入lodash时，Ghost的创始人John O'Nolan说到，“这是一个非常明智的选择，它几乎完全是由我们开源开发社区推动的。

我们发现lodash包含更多的功能，更好的性能、恰到好处地使用了semver，并且在Node.js社区（以及其他依赖）中越来越抢眼。

lodash不但深受node.js开发者欢迎，还广泛应用于基于浏览器的项目中。Web开发者可以使用实验构建工具来挑选项目中需要的方法，而不必下载整个库。对于node.js项目，在不需要整个包的时候，可以只include单个方法。

# FP Guide

jdalton edited this page 10 hours ago · [1 revision](https://github.com/lodash/lodash/wiki/FP-Guide/_history)

### Pages 8

* [**Home**](https://github.com/lodash/lodash/wiki)
* [**Build Differences**](https://github.com/lodash/lodash/wiki/Build-Differences)
* [**Changelog**](https://github.com/lodash/lodash/wiki/Changelog)
* [**Deprecations**](https://github.com/lodash/lodash/wiki/Deprecations)
* [**FP Guide**](https://github.com/lodash/lodash/wiki/FP-Guide)
* [**Migrating**](https://github.com/lodash/lodash/wiki/Migrating)
* [**Resources**](https://github.com/lodash/lodash/wiki/Resources)
* [**Roadmap**](https://github.com/lodash/lodash/wiki/Roadmap)

##### Clone this wiki locally



[**Clone in Desktop**](github-windows://openRepo/https:/github.com/lodash/lodash.wiki)

## lodash/fp

The lodash/fp module is an instance of lodash with its methods wrapped to produce immutable auto-curried iteratee-first data-last methods.

## Installation

In a browser:

<script src='path/to/lodash.js'></script>

<script src='path/to/lodash.fp.js'></script>

In Node.js:

// Load the fp build.

var \_ = require('lodash/fp');

// Load a method category.

var object = require('lodash/fp/object');

// Load a single method for smaller builds with browserify/rollup/webpack.

var extend = require('lodash/fp/extend');

## Convert

Although lodash/fp & its method modules come pre-converted, there are times when you may want to convert another lodash package or create a customized conversion. That’s when theconvert module comes in handy.

var convert = require('lodash/fp/convert');

// Convert by name.

var assign = convert('assign', require('lodash.assign'));

// Convert by object.

var fp = convert({

'assign': require('lodash.assign'),

'chunk': require('lodash.chunk')

});

// Convert by `lodash` instance.

var fp = convert(lodash.runInContext());

It’s even customizable.

// Every option is `true` by default.

var filter = convert('filter', \_.filter, {

// Specify capping iteratee arguments.

'cap': true,

// Specify currying.

'curry': true,

// Specify fixed arity.

'fixed': true,

// Specify immutable operations.

'immutable': true,

// Specify rearranging arguments.

'rearg': true

});

// Specify `cap` of `false` to create a function that doesn’t cap iteratee arguments.

var filter = convert('filter', \_.filter, { 'cap': false });

filter(function(value, index) {

return index % 2 == 0;

})(['a', 'b', 'c']);

// => ['a', 'c']

## Mapping

Immutable auto-curried iteratee-first data-last methods sound great, but what does that really mean for each method? Below is a breakdown of the mapping used to convert each method.

#### Capped Iteratee Arguments

Methods that cap iteratees to one argument:  
cloneDeepWith, cloneWith, dropRightWhile, dropWhile, every, filter, find, findIndex,findKey, findLast, findLastIndex, findLastKey, flatMap, forEach, forEachRight, forIn,forInRight, forOwn, forOwnRight, map, mapKeys, mapValues, partition, reject, remove,some, takeRightWhile, takeWhile, & times

Methods that cap iteratees to two arguments:  
assignInWith, assignWith, isEqualWith, isMatchWith, reduce, reduceRight, & transform

The iteratee of mapKeys is invoked with one argument: (key)

#### Fixed Arity

Methods with a fixed arity of one:  
attempt, castArray, ceil, create, curry, curryRight, floor, fromPairs, invert, iteratee,memoize, method, methodOf, mixin, over, overEvery, overSome, rest, reverse, round,runInContext, spread, template, trim, trimEnd, trimStart, uniqueId, & words

Methods with a fixed arity of two:  
add, after, ary, assign, assignIn, at, before, bind, bindKey, chunk, cloneDeepWith,cloneWith, concat, countBy, curryN, curryRightN, debounce, defaults, defaultsDeep, delay,difference, drop, dropRight, dropRightWhile, dropWhile, endsWith, eq, every, filter,find, find, findIndex, findKey, findLast, findLastIndex, findLastKey, flatMap,flattenDepth, forEach, forEachRight, forIn, forInRight, forOwn, forOwnRight, get,groupBy, gt, gte, has, hasIn, includes, indexOf, intersection, invertBy, invoke,invokeMap, isEqual, isMatch, join, keyBy, lastIndexOf, lt, lte, map, mapKeys, mapValues,matchesProperty, maxBy, merge, minBy, omit, omitBy, overArgs, pad, padEnd, padStart,parseInt, partial, partialRight, partition, pick, pickBy, pull, pullAll, pullAt, random,range, rangeRight, rearg, reject, remove, repeat, result, sampleSize, some, sortBy,sortedIndex, sortedIndexOf, sortedLastIndex, sortedLastIndexOf, sortedUniqBy, split,startsWith, subtract, sumBy, take, takeRight, takeRightWhile, takeWhile, tap, throttle,thru, times, trimChars, trimCharsEnd, trimCharsStart, truncate, union, uniqBy, uniqWith,unset, unzipWith, without, wrap, xor, zip, zipObject, & zipObjectDeep

Methods with a fixed arity of three:  
assignInWith, assignWith, clamp, differenceBy, differenceWith, getOr, inRange,intersectionBy, intersectionWith, isEqualWith, isMatchWith, mergeWith, orderBy, pullAllBy,pullAllWith, reduce, reduceRight, replace, set, slice, sortedIndexBy, sortedLastIndexBy,transform, unionBy, unionWith, update, xorBy, xorWith, & zipWith

Methods with a fixed arity of four:  
fill, setWith, & updateWith

#### Rearranged Arguments

Methods with a fixed arity of two have an argument order of:  
(b, a)

Methods with a fixed arity of three have an argument order of:  
(b, c, a)

Methods with a fixed arity of four have an argument order of:  
(c, d, b, a)

Methods with custom argument orders:

* \_.assignInWith has an order of (c, a, b)
* \_.assignWith has an order of (c, a, b)
* \_.getOr has an order of (c, b, a)
* \_.isMatchWith has an order of (c, b, a)
* \_.mergeWith has an order of (c, a, b)
* \_.pullAllBy has an order of (c, b, a)
* \_.pullAllWith has an order of (c, b, a)
* \_.setWith has an order of (d, b, c, a)
* \_.sortedIndexBy has an order of (c, b, a)
* \_.sortedLastIndexBy has an order of (c, b, a)
* \_.updateWith has an order of (d, b, c, a)
* \_.zipWith has an order of (c, a, b)

Methods with unchanged argument orders:  
add, assign, assignIn, concat, difference, gt, gte, lt, lte, matchesProperty, merge,partial, partialRight, random, range, rangeRight, subtract, zip, & zipObject

The methods partial & partialRight accept an array of arguments to partially apply as their second parameter.

#### New Methods

Methods created to accommodate Lodash’s variadic methods:  
curryN, curryRightN, getOr, trimChars, trimCharsEnd, & trimCharsStart

#### Aliases

There are 34 method aliases:

* \_.\_\_ is an alias of \_.placeholder
* \_.all is an alias of \_.some
* \_.allPass is an alias of \_.overEvery
* \_.apply is an alias of \_.spread
* \_.assoc is an alias of \_.set
* \_.assocPath is an alias of \_.set
* \_.compose is an alias of \_.flowRight
* \_.contains is an alias of \_.includes
* \_.dissoc is an alias of \_.unset
* \_.dissocPath is an alias of \_.unset
* \_.each is an alias of \_.forEach
* \_.eachRight is an alias of \_.forEachRight
* \_.equals is an alias of \_.isEqual
* \_.extend is an alias of \_.assignIn
* \_.extendWith is an alias of \_.assignInWith
* \_.first is an alias of \_.head
* \_.init is an alias of \_.initial
* \_.mapObj is an alias of \_.mapValues
* \_.omitAll is an alias of \_.omit
* \_.nAry is an alias of \_.ary
* \_.path is an alias of \_.get
* \_.pathEq is an alias of \_.matchesProperty
* \_.pathOr is an alias of \_.getOr
* \_.pickAll is an alias of \_.pick
* \_.pipe is an alias of \_.flow
* \_.prop is an alias of \_.get
* \_.propOf is an alias of \_.propertyOf
* \_.propOr is an alias of \_.getOr
* \_.somePass is an alias of \_.overSome
* \_.unapply is an alias of \_.rest
* \_.unnest is an alias of \_.flatten
* \_.useWith is an alias of \_.overArgs
* \_.whereEq is an alias of \_.filter
* \_.zipObj is an alias of \_.zipObject