FREE eBook

LEARNING lodash

Free unaffiliated eBook created from
Stack Overflow contributors.

#lodash

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: lodash

It is an unofficial and free lodash ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official lodash.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with lodash

Remarks

Lodash is a library of utilities for manipulating and examining objects and arrays.

Versions

Version	Release Date
v0.1.0	2012-04-23
v0.2.0	2012-05-21
v0.2.1	2012-05-24
v0.2.2	2012-05-30
v0.3.0	2012-06-06
v0.3.1	2012-06-10
v0.3.2	2012-06-14
v0.4.0	2012-07-11
v0.4.1	2012-07-11
v0.4.2	2012-07-16
v0.5.0	2012-08-17
v0.5.1	2012-08-18
v0.5.2	2012-08-21
v0.6.0	2012-08-28
v0.6.1	2012-08-29
v0.7.0	2012-09-11
v0.8.0	2012-10-01
v0.8.1	2012-10-04
v0.8.2	2012-10-10

Version	Release Date
v0.9.0	2012-10-24
v0.9.1	2012-10-31
v0.9.2	2012-11-09
v0.10.0	2012-11-17
v1.0.0	2013-02-14
v1.0.1	2013-02-18
v1.0.2	2013-02-18
v1.1.0	2013-03-26
v1.1.1	2013-03-27
v1.2.0	2013-04-16
v1.2.1	2013-04-29
v1.3.0	2013-06-11
v1.3.1	2013-06-12
v2.0.0	2013-09-13
v2.1.0	2013-09-22
v2.2.0	2013-09-28
v2.2.1	2013-10-03
v2.3.0	2013-11-10
v2.4.0	2013-11-25
v2.4.1	2013-12-02
v2.4.2	2015-04-26
v3.0.0	2015-01-26
v3.0.1	2015-01-30
v3.1.0	2015-02-03
v3.2.0	2015-02-12

Version	Release Date
v3.3.0	2015-02-20
v3.3.1	2015-02-24
v3.4.0	2015-03-06
v3.5.0	2015-03-08
v3.6.0	2015-03-25
v3.7.0	2015-04-15
v3.8.0	2015-05-01
v3.9.0	2015-05-19
v3.9.2	2015-05-24
v3.9.3	2015-05-26
v3.10.0	2015-06-30
v3.10.1	2015-08-04
v4.0.0	2016-01-12
v4.0.1	2016-01-25
v4.1.0	2016-01-29
v4.2.0	2016-02-02
v4.2.1	2016-02-03
v4.3.0	2016-02-07
v4.4.0	2016-02-15
v4.5.0	2016-02-17
v4.5.1	2016-02-21
v4.6.0	2016-02-29
v4.6.1	2016-03-01
v4.7.0	2016-03-31
v4.8.0	2016-04-04

Version	Release Date
v4.8.1	2016-04-04
v4.8.2	2016-04-04
v4.9.0	2016-04-08
v4.10.0	2016-04-11
v4.11.0	2016-04-13
v4.11.1	2016-04-14
v4.11.2	2016-04-21
v4.12.0	2016-05-08
v4.13.0	2016-05-22
v4.13.1	2016-05-23

Examples

Setup

Lodash works equally well on both servers (like node.js) and browsers.

node.js with npm

Download with npm from the CLI:

```
npm install lodash
```

Then in your node scripts:

```
var _ = require("lodash");

// use lodash in your program...
```

Download own copy for clientside in website (ie. in the browser)

1. Download lodash or use a package manager like npm, jspm or bower.
2. Include the script reference in your page with `<script src="lodash.js"></script>`. (Fix the

path if it's not in same directory as the webpage.)

Use lodash in a browser from a CDN

Go to a CDN site and select the version you want to use. Copy the link and use it for the script reference in your page.

```
<script src="https://cdn.jsdelivr.net/lodash/4.13.1/lodash.min.js"></script>
```

4.13.1 is current version at time of writing

Read Getting started with lodash online: https://riptutorial.com/lodash/topic/2522/getting-started-with-lodash

Chapter 2: Chaining

Remarks

Implicit chaining with `_(arr1)` and explicit chaining with `_.chain(arr1)` work in similar ways. The examples below show how they differ slighlty.

Explicit chaining with `_.chain(...)`

```
var arr1 = [10, 15, 20, 25, 30, 15, 25, 35];

var sumOfUniqueValues = _.chain(arr1)
    .uniq()
    .sum()       // sum returns a single value
    .value();    //   which must be unwrapped manually with explicit chaining

// sumOfUniqueValues is now 135
```

Implicit chaining with `_(...)`

```
var arr1 = [10, 15, 20, 25, 30, 15, 25, 35];

var sumOfUniqueValues = _(arr1)
    .uniq()
    .sum();      // sum returns a single value and is automatically unwrapped
                 //   with implicit chaining

// sumOfUniqueValues is now 135
```

The two behave differently when ending the chain with an operation that returns a single value: With implicit chaining, the "unwrapping" of the single value is implied. (Thus no need to call `.value()`.)

(When the implicit chain ends with a collection value, you'll still need to unwrap the result with `.value()`.)

Examples

Chaining

Any lodash collection method has two syntaxes.

Without chaining:

```
var arr1 = [10, 15, 20, 25, 30, 15, 25, 35];

var arr2 = _.filter(arr1, function(item){ return item % 10 === 5 });
```

```
// arr2 now contains [15, 25, 15, 25, 35]

var arr3 = _.uniq(arr2);
// arr3 now contains [15, 25, 35]

var arr4 = _.map(arr3, function(item){ return item + 1 });
// arr4 now contains [16, 26, 36]
```

With chaining:

```
var arr1 = [10, 15, 20, 25, 30, 15, 25, 35];

var arr4 = _(arr1)
    .filter(function(item){ return item % 10 === 5 })
    .uniq()
    .map(function(item){ return item + 1 })
    .value();
// arr4 now contains [16, 26, 36] without creating the intermediate results.
```

The chaining version of this is actually more efficient, since no intermediate results are created. The expressions are evaluated lazily by the call to `.values()` at the end of the chain.

Read Chaining online: https://riptutorial.com/lodash/topic/2846/chaining

Chapter 3: Utils

Examples

_.identity

This method just returns the first argument.

```
var res1 = _.identity(10, 20);
// res1 now is 10

var res2 = _.identity("hello", "world");
// res2 now is "hello"
```

What does *_.identity* mean in lodash documentation?

This method is used throughout *lodash* documentation instead of `function(x){return x;}` (or ES6 equivalent `x => x`).

It usually means either "no transformation" or when used as a predicate: the truthiness of the value.

Example of *_.identity* in documentation of *_.times*

The `_.times` function takes two arguments. Its expressed like this in the documentation: *var res = _.times(n, [iteratee=_.identity])*

The *iteratee* is used to transform the values as they are iterated over.

The documentation shows that the *iteratee* parameter is optional, and if it is omitted it will have the default value of `_.identity`, **which in this case means "no transformation"**

```
var res = _.times(5);      // returns [0, 1, 2, 3, 4]

// means the same as:
var res = _.times(5, _.identity);

// which again does the same as:
var res = _.times(5, function(x){ return x; });

// or with the ES6 arrow syntax:
var res = _.times(5, x => x);
```

Example of *_.identity* in documentation of *_.findKey* and

_.findLastKey

The `_.findKey` and `_.findLastKey` functions takes two arguments. Its expressed like this in the documentation: *_.findKey(object, [predicate=_.identity])* and *_.findLastKey(object, [predicate=_.identity])*

This again means that the second parameter is optional, and if it is omitted it will have the default value of `_.identity`, **which in this case means the first (or last) of "anything that is truthy"**

```
var p = {
    name: "Peter Pan",
    age: "Not yet a grownup",
    location: "Neverland"
};

var res1 = _.findKey(p);        // returns "name"
var res2 = _.findLastKey(p);    // returns "location"

// means the same as:
var res1 = _.findKey(p, _.identity);
var res2 = _.findLastKey(p, _.identity);

// which again means the same as:
var res1 = _.findKey(p, function(x) { return x; }
var res2 = _.findLastKey(p, function(x) { return x; }

// or with ES6 arrow syntax:
var res1 = _.findKey(p, x => x);
var res2 = _.findKey(p, x => x);
```

Read Utils online: https://riptutorial.com/lodash/topic/3192/utils

Chapter 4: Working with Lists and Arrays

Syntax

- _.map(collection, Function) => newCollection
- _.filter(collection, Predicate) => newCollection
- _.some(collection, Predicate) => true or false
- _.reduce(collection, BiFunction, seed) => accumulated value

Parameters

Parameter	Meaning
Collection	An iterable group of elements. This can be an array or an object.
Function	A function that takes 1 input, and returns one output.
BiFunction	A function that takes 2 inputs, and returns one output.
Predicate	A function that takes 1 input, and returns a boolean value.
seed	The initial value for a reduction operation. When this is left out, the first element of the collection is used instead.

Examples

Use _.map to Transform a List

`_.map` is useful for changing a list into a different list in a purely declarative way. Rather than using imperative techniques like a `while` or `for` loop in javascript, you can just specify how you want to manipulate an element of a list and

Use `_.map` to make a new list transformed by the function you provide.

Let's say we want to square all the numbers in a list. First we'll create a list using the `_.range` function:

```
var a = _.range(10);       // [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 ]
```

Now we'll create a list of squares by using `_.map`:

```
var b = _.map(a, function(e){ return e * e;} );
// b is now [ 0, 1, 4, 9, 16, 25, 36, 49, 64, 81 ]
```

_.filter

Filtering a list down to only the elements we want to do. Lodash provides a function called `_.filter` filters elements based on the predicate function you provide. A predicate is a function that takes data in and returns either true or false.

Let's look at how we'd get just the even numbers from a list of the numbers 0 through 9:

```
var numbers = _.range(0,10);      // [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

var evenNumbers = _.filter(numbers, function(e){ return e % 2 == 0; });
// evenNumbers is now [ 0, 2, 4, 6, 8 ]
```

_.some

We can assert some predicate over a collection using `_.some` to check if there is at least one member of a collection that meets some criteria. This is great when writing business logic to assert certain conditions on a group of objects. For example, let's say you wanted to make sure at least one person in a group had a driver's license before it was possible for that group to go on a road trip. We make no guarantees on how happy of a group that'll be at the end of the road trip, however.

```
var friends = [
    {
        'name': 'Fred',
        'hasLicense': false
    },
    {
        'name': 'Steve',
        'hasGuitar': true
    },
    {
        'name': 'Mary',
        'hasLicense': true
    },
]

function canGroupDrive(arr){
    return _.some(arr, function(e){ return e.hasLicense; });
}

canGroupDrive(friends);    // returns true
```

_.reduce

Reducing a list to a single value is easy when you have `_.reduce`. Let's say we wanted to see if a group of people could afford a cab ride. We'd want to look at all the money they have together as a group, which means we'd want to reduce a list of objects to a single value, in this case the sum of the money they have.

```
var friends = [
    {
```

```
        'name': 'Alice',
        'money': 10
    },
    {
        'name': 'Bob',
        'money': 3
    },
    {
        'name': 'Clyde',
        'money': 8
    },
]

var totalMoney = function(arr){
    return _.reduce(
        arr,
        function(accumulated, e){
            return accumulated + e.money;
        },
        0
    );
}

function canAffordCab(arr){
    return 18 < totalMoney(arr);
}

canAffordCab(friends);    // returns true
```

Read Working with Lists and Arrays online: https://riptutorial.com/lodash/topic/3160/working-with-lists-and-arrays

Chapter 5: Working with objects

Examples

.has

Determine if an object has (or contains) a key. If the key to search for is expressed as a path (with dot notation) it will traverse nested object structures to determine if the key exists.

```
var obj = {
  a: 2,
  b: 3,
  c: {
    dd:40,
    ee:{
      fff:500
    }
  }
};

var res1 = _.has(obj, "a");          // true
var res2 = _.has(obj, "a.b");        // false
var res3 = _.has(obj, "c");          // true
var res4 = _.has(obj, "c.ee");       // true
var res5 = _.has(obj, "c.fff");      // false
var res6 = _.has(obj, "c.dd.fff");   // false
var res7 = _.has(obj, "c.ee.fff");   // true
```

Arrays can be used to split up parts of the path instead of strings

```
var res8 = _.has(obj, ["a", "b"]);   // false, same as res2
var res9 = _.has(obj, ["c", "ee"]);  // true, same as res4
```

Note

`_.has` will only look at the direct properties (aka. owned properties) of the object.

Read Working with objects online: https://riptutorial.com/lodash/topic/5504/working-with-objects

Credits

S. No	Chapters	Contributors
1	Getting started with lodash	4444, Arjan Einbu, Community, Conrad.Dean
2	Chaining	4castle, Arjan Einbu, canac
3	Utils	Arjan Einbu
4	Working with Lists and Arrays	4castle, Arjan Einbu, Conrad.Dean
5	Working with objects	Arjan Einbu