Lodash Library

What is that and what does it consist of?

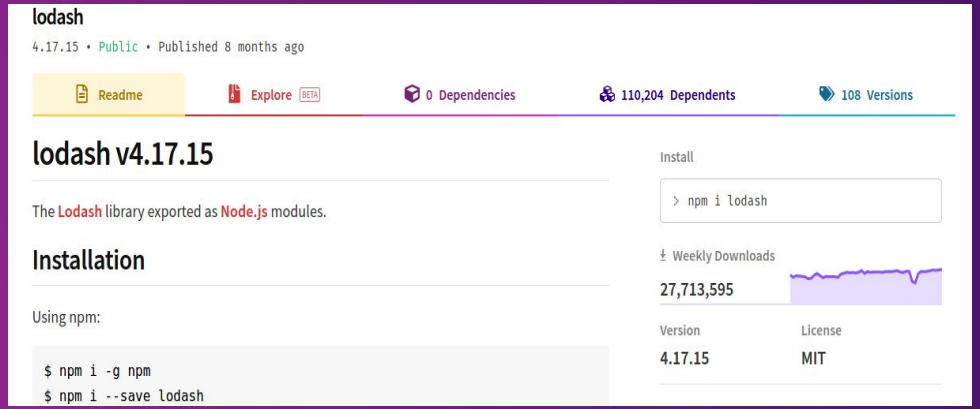
Tunisian Javascript Community
Fourth Meetup

FARES KARBIA

Tunisian JS Community

What is a Lodash Library?

Lodash is a superset of Underscore. What is Underscore? Underscore is “[…] a whole mess of useful functional programming helpers without extending any built-in objects.” In short, Lodash is a JS helper library for arrays, strings and objects. Documentation is here: https://lodash.com/docs



_.map()

_.filter()

_.times()

var _ = require('lodash')

_.chain()

_.find()

_.debounce()

Traditional way VS Lodash

_.map()

To extract some property from an array of objects:

```
let arr = [{ n: 1 }, { n: 2 }]
// ES6
arr.map((obj) => obj.n)
// Lodash
_.map(arr, 'n')
```

This can be more helpful when it comes to complex objects:

```
let arr = [
  { a: [ { n: 1 } ]},
  { b: [ { n: 1 } ]}
]
// ES6
arr.map((obj) => obj.a[0].n) // TypeError: property 'a' is not defined in arr[1]
// Lodash
_.map(arr, 'a[0].n') // => [1, undefined]
```

_.filter()

For _.filter, there's also predicate shorthand.

```
let users = [
  { 'user': 'barney', 'age': 36, 'active': true },
  { 'user': 'fred',   'age': 40, 'active': false }
];
// ES6
users.filter((o) => o.active)
// Lodash
_.filter(users, 'active')
_.filter(users, ['active', true])
_.filter(users, {'active': true, 'age': 36})
```

_.chain()

```
let lines = `
an apple orange the grape
banana an apple melon
an orange banana apple
`.split('\n')

_.chain(lines)
  .flatMap(line => line.split(/\s+/))
  .filter(word => word.length > 3)
  .groupBy(_.identity)
  .mapValues(_.size)
  .forEach((count, word) => { console.log(word, count) })

// apple 3
// orange 2
// grape 1
// banana 2
// melon 1
```

Processing collections with chaining, lazy evaluation, along with short, easy-to-test functions, is quite popular these days. Most Lodash functions regarding collections can be chained easily.

_.debounce()

_.debounce will invoke a function after a certain amount of time since the last time it was invoked.

```
function validateEmail() {
  // Validate email here and show error message if not valid
}

var emailInput = document.getElementById("email-field");
emailInput.addEventListener("keyup", _.debounce(validateEmail, 500));
```

_.find()

Instead iterating through an array with a loop to find a specific object, we can simply use _.find. That's nice, but this is not the only thing you can do with _.find. You can also find an object using multiple properties with a single line of code. Take a look!

```
var users = [
  { firstName: "John", lastName: "Doe", age: 28, gender: "male" },
  { firstName: "Jane", lastName: "Doe", age: 5, gender: "female" },
  { firstName: "Jim", lastName: "Carrey", age: 54, gender: "male" },
  { firstName: "Kate", lastName: "Winslet", age: 40, gender: "female" }
];

var user = _.find(users, { lastName: "Doe", gender: "male" });
// user -> { firstName: "John", lastName: "Doe", age: 28, gender: "male" }

var underAgeUser = _.find(users, function(user) {
  return user.age < 18;
});
// underAgeUser -> { firstName: "Jane", lastName: "Doe", age: 5, gender: "female" }
```

_.times()

_.times receives as arguments the number of iterations and a function to execute n times and returns an array of the results. Very useful when creating dynamic test data.

```
function getRandomInteger() {
  return Math.round(Math.random() * 100);
}

var result = _.times(5, getRandomNumber);
// result => [64, 70, 29, 10, 23]
```

ES6 introduces some useful syntaxes like destructuring, spread and arrow function, which can be used to replace a lot of Lodash functions.

```
// Lodash
_.head([1, 2, 3]) // => 1
_.tail([1, 2, 3]) // => [2, 3]
// ES6 destructuring syntax
const [head, ...tail] = [1, 2, 3]

// Lodash
let say = _.rest((who, fruits) => who + ' likes ' + fruits.join(','))
say('Jerry', 'apple', 'grape')
// ES6 spread syntax
say = (who, ...fruits) => who + ' likes ' + fruits.join(',')
say('Mary', 'banana', 'orange')

// Lodash
_.constant(1)() // => 1
_.identity(2) // => 2
// ES6
(x => (() => x))(1)() // => 1
(x => x)(2) // => 2

// Partial application
let add = (a, b) => a + b
// Lodash
let add1 = _.partial(add, 1)
// ES6
add1 = b => add(1, b)

// Curry
// Lodash
let curriedAdd = _.curry(add)
let add1 = curriedAdd(1)
// ES6
curriedAdd = a => b => a + b
add1 = curriedAdd(1)
```

Conclusion

Lodash adds great power to JavaScript language. One can write concise and efficient codes with minor efforts. Besides, Lodash is fully modularized. Though some of its functions will eventually deprecate, but I believe it'll still bring many benefits to developers, while pushing the development of JS language as well.

THANKS

Does anyone have any questions?

fareswardeni@gmail.com

mobelite.fr