eBook Gratuit

APPRENEZ lodash

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#lodash

Table des matières

À propos

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

Chapitre 1: Commencer avec lodash

Remarques

Lodash est une bibliothèque d'utilitaires pour manipuler et examiner des objets et des tableaux.

Versions

Version	Date de sortie
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	Date de sortie
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	Date de sortie
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	Date de sortie
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

Installer

Lodash fonctionne aussi bien sur les deux serveurs (comme node.js) que sur les navigateurs.

node.js avec npm

Télécharger avec npm depuis la CLI:

```
npm install lodash
```

Ensuite, dans vos scripts de noeud:

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

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

Téléchargez votre propre copie pour les clients sur le site Web (c.-à-d. Dans le navigateur)

1. Téléchargez lodash ou utilisez un gestionnaire de paquets comme npm, jspm ou bower.
2. Incluez la référence de script dans votre page avec `<script src="lodash.js"></script>` . (Corrigez le chemin si ce n'est pas dans le même répertoire que la page Web.)

Utiliser lodash dans un navigateur depuis un CDN

Accédez à un site CDN et sélectionnez la version que vous souhaitez utiliser. Copiez le lien et utilisez-le pour la référence du script dans votre page.

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

4.13.1 est la version actuelle au moment de la rédaction

Lire Commencer avec lodash en ligne: https://riptutorial.com/fr/lodash/topic/2522/commencer-avec-lodash

Chapitre 2: Chaînage

Remarques

Le chaînage implicite avec `_(arr1)` et le chaînage explicite avec `_.chain(arr1)` fonctionnent de la même manière. Les exemples ci-dessous montrent comment ils diffèrent légèrement.

Chaînage explicite avec `_.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
```

Chaînage implicite avec `_(...)`

```
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
```

Les deux se comportent différemment lors de la fin de la chaîne avec une opération qui renvoie une valeur unique: Avec le chaînage implicite, le "déroulage" de la valeur unique est implicite. (Donc pas besoin d'appeler `.value()` .)

(Lorsque la chaîne implicite se termine par une valeur de collection, vous devez toujours `.value()` le résultat avec `.value()` .)

Examples

Chaînage

Toute méthode de collecte de lodash a deux syntaxes.

Sans chaînage:

```
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]
```

Avec chaînage:

```
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.
```

La version de chaînage est en réalité plus efficace, car aucun résultat intermédiaire n'est créé. Les expressions sont évaluées paresseusement par l'appel à `.values()` à la fin de la chaîne.

Lire Chaînage en ligne: https://riptutorial.com/fr/lodash/topic/2846/chainage

Chapitre 3: Travailler avec des listes et des tableaux

Syntaxe

- _.map (collection, Fonction) => newCollection
- _.filter (collection, prédicat) => newCollection
- _.some (collection, Predicate) => true ou false
- _.reduce (collection, BiFunction, seed) => valeur accumulée

Paramètres

Paramètre	Sens
Collection	Un groupe d'itération d'éléments. Cela peut être un tableau ou un objet.
Fonction	Une fonction qui prend 1 entrée et renvoie une sortie.
BiFonction	Une fonction qui prend 2 entrées et renvoie une sortie.
Prédicat	Une fonction qui prend 1 entrée et renvoie une valeur booléenne.
la graine	La valeur initiale pour une opération de réduction. Lorsque cette option est omise, le premier élément de la collection est utilisé à la place.

Examples

Utilisez _.map pour transformer une liste

`_.map` est utile pour changer une liste dans une liste différente de manière purement déclarative. Plutôt que d'utiliser des techniques impératives comme un `while` ou `for` boucle en javascript, vous pouvez simplement spécifier comment vous voulez manipuler un élément d'une liste et

Utilisez `_.map` pour transformer une nouvelle liste en fonction de la fonction que vous fournissez.

Disons que nous voulons mettre tous les nombres dans une liste. Nous allons d'abord créer une liste en utilisant la fonction `_.range` :

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

Nous allons maintenant créer une liste de carrés en utilisant `_.map` :

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

_.filtre

Filtrer une liste pour ne sélectionner que les éléments que nous voulons faire. Lodash fournit une fonction appelée `_.filter` filtres éléments en fonction de la fonction de prédicat que vous fournissez. Un prédicat est une fonction qui prend des données et renvoie true ou false.

Regardons comment nous obtiendrions juste les nombres pairs d'une liste des nombres 0 à 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 ]
```

_.certains

Nous pouvons affirmer un prédicat sur une collection en utilisant `_.some` pour vérifier si au moins un membre d'une collection répond à certains critères. C'est génial lors de l'écriture de la logique métier pour affirmer certaines conditions sur un groupe d'objets. Par exemple, supposons que vous vouliez vous assurer qu'au moins une personne dans un groupe avait un permis de conduire avant que ce groupe ne puisse partir en voyage. Nous ne garantissons cependant pas la satisfaction d'un groupe qui sera à la fin du voyage.

```
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
```

_.réduire

Réduire une liste à une seule valeur est facile lorsque vous avez `_.reduce` . Disons que nous voulions voir si un groupe de personnes pouvait se permettre un trajet en taxi. Nous voudrions examiner l'ensemble de l'argent qu'ils ont ensemble en tant que groupe, ce qui signifie que nous voudrions réduire une liste d'objets à une valeur unique, en l'occurrence la somme de leur argent.

```
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
```

Lire Travailler avec des listes et des tableaux en ligne:
https://riptutorial.com/fr/lodash/topic/3160/travailler-avec-des-listes-et-des-tableaux

Chapitre 4: Travailler avec des objets

Examples

.a

Déterminez si un objet a (ou contient) une clé. Si la clé à rechercher est exprimée sous la forme d'un chemin (avec notation par points), elle traversera les structures d'objet imbriquées pour déterminer si la clé existe.

```
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
```

Les tableaux peuvent être utilisés pour séparer des parties du chemin plutôt que des chaînes

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

Remarque

`_.has` ne regardera que les propriétés directes (propriétés appartenant à alias) de l'objet.

Lire Travailler avec des objets en ligne: https://riptutorial.com/fr/lodash/topic/5504/travailler-avec-des-objets

Chapitre 5: Utils

Examples

_.identité

Cette méthode renvoie simplement le premier argument.

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

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

Que signifie _.*identity* dans la documentation lodash?

Cette méthode est utilisée dans la documentation *lodash* au lieu de la `function(x){return x;}` (ou l'équivalent ES6 `x => x`).

Cela signifie généralement soit "pas de transformation", soit utilisé comme prédicat: la véracité de la valeur.

Exemple de _.*identité* dans la documentation de _.*times*

La fonction `_.times` prend deux arguments. Cela s'exprime comme ceci dans la documentation: *var res = _.times (n, [iteratee = _. Identity])*

L' *iterate* est utilisé pour transformer les valeurs au fur et à mesure de leur itération.

La documentation montre que le paramètre *iteratee* est facultatif et, s'il est omis, il aura la valeur par défaut de `_.identity` , **ce qui signifie dans ce cas "aucune 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);
```

Exemple de _.*identity* dans la documentation de _.*findKey* et

_.findLastKey

Les fonctions `_.findKey` et `_.findLastKey` prennent deux arguments. Cela est exprimé comme ceci dans la documentation: *_.findKey (objet, [predicate = _. Identity])* et *_.findLastKey (objet, [predicate = _. Identity])*

Cela signifie à nouveau que le second paramètre est facultatif et, s'il est omis, il aura la valeur par défaut de `_.identity` , **ce qui dans ce cas signifie le premier (ou le dernier) "tout ce qui est véridique"**

```
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);
```

Lire Utils en ligne: https://riptutorial.com/fr/lodash/topic/3192/utils

Crédits

S. No	Chapitres	Contributeurs
1	Commencer avec lodash	4444, Arjan Einbu, Community, Conrad.Dean
2	Chaînage	4castle, Arjan Einbu, canac
3	Travailler avec des listes et des tableaux	4castle, Arjan Einbu, Conrad.Dean
4	Travailler avec des objets	Arjan Einbu
5	Utils	Arjan Einbu