Lodash : return first key of object whose value(i.e Array) has a given element (i.e string) in it

Asked 3 years, 5 months ago Active 3 years ago Viewed 33k times

21

2

I have an object like:

```
var obj = {
  "01": ["a","b"],
  "03": ["c","d"],
  "04": ["e","c"]
};
```

and I know an array element (say "c") of the object key's value then **How to find first key value i.e "03" using lodash without using if else?**

I tried like this using lodash and if else:

```
var rId = "";
_.forOwn(obj, function (array, id) {
     if (_.indexOf(array, "c") >= 0) {
          rId = id;
          return false;
     }
});

console.log(rId); // "03"
```

Expected Result: first key i.e "03" if element matches else "".

After seeing comments: Now I'm also curious to know about

Does I need to go with native javascript(hard to read program in the cases if we use more than 2 if blocks) or lodash way(easily readable program solution in one line)?

javascript lodash

edited Sep 28 '16 at 18:09

asked Apr 30 '16 at 19:17

Sandeep Sharma

992 2 7 28

please add the wanted result. – Nina Scholz Apr 30 '16 at 19:24

how about native javascript solution? – RomanPerekhrest Apr 30 '16 at 19:25

Without lodash: jsfiddle.net/rayon_1990/5pd8sq5g – Rayon Apr 30 '16 at 19:27

4 Answers

34

Since you just want a way to be able to find a key using a simple Lodash command, the following should work:

```
_.findKey(obj, function(item) { return item.indexOf("c") !== -1; });
```

or, using ES6 syntax,

```
_.findKey(obj, (item) => (item.indexOf("c") !== -1));
```

This returns "03" for your example.

The predicate function - the second argument to `findKey()` - has automatic access to the value of the key. If nothing is found matching the predicate function, `undefined` is returned.

Documentation for `findKey()` is here.

Examples taken from the documentation:

```
var users = {
  'barney':  { 'age': 36, 'active': true },
  'fred':    { 'age': 40, 'active': false },
  'pebbles': { 'age': 1,  'active': true }
};

_.findKey(users, function(o) { return o.age < 40; });
// → 'barney' (iteration order is not guaranteed)
```

```
// The `_.matches` iteratee shorthand.
_.findKey(users, { 'age': 1, 'active': true });
// → 'pebbles'

// The `_.matchesProperty` iteratee shorthand.
_.findKey(users, ['active', false]);
// → 'fred'

// The `_.property` iteratee shorthand.
_.findKey(users, 'active');
// → 'barney'
```

edited Apr 30 '16 at 20:55

answered Apr 30 '16 at 19:31

Akshat Mahajan

6,432 2 23 35

I'm doing node.js programming. Now I'm also curious to know about Does I need to go with native javascript(hard to read program) or lodash way(easily readable program)? – Sandeep Sharma Apr 30 '16 at 19:48

@SandeepSharma Use what you feel is best for your needs and requirements. If you already have Lodash, I think it's worth sticking with it. Lodash's API is what Javascript's semantics should have been. Since it's server-side, there are no *real* arguments against using Lodash - on the client, it's a bulky library, but who cares about library size on the server? No one. There is a *tiny* performance hit, but not enough to really make a difference. – Akshat Mahajan Apr 30 '16 at 19:50

@SandeepSharma Also, Javascript programs don't have to be hard to read. If your Javascript is hard to read, it's because you're making it needlessly complicated. JS can be as simple and clean to read as Lodash if you take the time to subscribe to proper patterns. Lodash *is* Javascript, after all. – Akshat Mahajan Apr 30 '16 at 19:55

@SandeepSharma Anyway, if this answer correctly answers your original question, you should mark it as accepted answer. – Akshat Mahajan Apr 30 '16 at 19:59

you have fixed the item[0] to zeroth index. – Sandeep Sharma Apr 30 '16 at 20:46

7

The irony is it is not any harder to implement without any libs.

```
Object.keys(obj).filter(x => obj[x].includes("c"))[0]
```

answered Apr 30 '16 at 21:45

Роман Парадеев


3,357 10 28

2 Good point! You could even use `.find(...)` instead of `.filter(...)[0]` : `Object.keys(obj).find(x => obj[x].includes("c"))` – ContinuousLoad Aug 16 '17 at 19:40

1

Here comes a single liner answer from the future. Currently only works in Firefox 47 on. Part of ES7 proposal.

```
var obj = {
  "01": ["a","b"],
  "03": ["c","d"],
  "04": ["e","c"]
},
    res = Object.entries(obj).find(e => e[1].includes("c"))[0];
document.write(res);
```

Run code snippet | Copy snippet to answer | Expand snippet

edited Apr 30 '16 at 22:39

answered Apr 30 '16 at 19:48


Redu
14.2k 3 28 41

I believe is not on a part of ES7 but rather ES8. – Роман Парадеев Apr 30 '16 at 21:50

You probably shoud update `find` condition, since now you are not searching through the whole array, but only the first item. – Роман Парадеев Apr 30 '16 at 21:52

@Роман Парадеев Yes you are right i have misunderstood the quotestion. I will correct accordingly. By the way your solution is perfect and deserves a +. – Redu Apr 30 '16 at 22:37

0

As an alternative solution: consider native Javascript approach using `Object.keys` and `Array.some` functions:

```
var obj = {"01": ["a","b"],"03": ["c","d"],"04": ["e","c"]},
        search_str = "c", key = "";
```

```
Object.keys(obj).some(function(k) { return obj[k].indexOf(search_str) !== -1 && (key =
k); });
// the same with ES6 syntax:
// Object.keys(obj).some((k) => obj[k].indexOf(search_str) !== -1 && (key = k));

console.log(key);  // "03"
```

edited Apr 30 '16 at 19:55

answered Apr 30 '16 at 19:33

RomanPerekhrest

66.1k 4 22 58

Using ES6 syntax would make reading this slightly easier. – Akshat Mahajan Apr 30 '16 at 19:39

@AkshatMahajan, added as comment – RomanPerekhrest Apr 30 '16 at 19:55