Convert Array to Object

Asked 8 years, 9 months ago Active 3 days ago Viewed 630k times

416

134

What is the best way to convert:

```
['a','b','c']
```

to:

```
{
  0: 'a',
  1: 'b',
  2: 'c'
}
```

javascript arrays node.js object

edited Sep 17 '18 at 8:58

Nick
2,414 1 22 44

asked Nov 18 '10 at 14:22

David Hellsing
75.9k 38 150 190

3 Maybe what he needs is for some duck-typing code *not* to think that it is an Array instance – Pointy Nov 18 '10 at 14:26

5 Its worth pointing out that Javascript arrays *are* objects. – Spudley Nov 18 '10 at 14:32

If anyone else is looking for a Lodash solution, consider `_.keyBy` (formerly `_.indexBy`): lodash.com/docs#keyBy – jtheletter Feb 18 '16 at 2:26

This is a bit confusing because arrays are objects already, but I guess the point of the question is converting the array exotic object to an ordinary object. – Oriol Apr 3 '16 at 17:25

1 A simple way to do this with Lodash is `_.toPlainObject`. Ex: `var myObj = _.toPlainObject(myArr)` – Julian Soro Nov 4 '16 at 19:36

43 Answers

1 2 next

338

ECMAScript 6 introduces the easily polyfillable `Object.assign` :

> The `Object.assign()` method is used to copy the values of all enumerable own properties from one or more source objects to a target object. It will return the target object.

```
Object.assign({}, ['a','b','c']); // {0:"a", 1:"b", 2:"c"}
```

The own `length` property of the array is not copied because it isn't enumerable.

Also, you can use ES6 spread syntax to achieve the same result:

```
{ ...['a', 'b', 'c'] }
```

edited Jan 18 at 9:26 — itamar — 2,411 3 29 49

answered Apr 3 '16 at 17:03 — Oriol — 173k 39 298 391

Just want to point out - if you already have an array of sorted properties from the original object, using the **spread** operator is what will turn that array directly into a new object: `{ ...[sortedArray]}` – HappyHands31 Jul 26 at 15:18

459

With a function like this:

```
function toObject(arr) {
  var rv = {};
  for (var i = 0; i < arr.length; ++i)
    rv[i] = arr[i];
  return rv;
}
```

Your array already is more-or-less just an object, but arrays do have some "interesting" and special behavior with respect to integer-

```
function toObject(arr) {
  var rv = {};
  for (var i = 0; i < arr.length; ++i)
    if (arr[i] !== undefined) rv[i] = arr[i];
  return rv;
}
```

In modern JavaScript runtimes, you can use the `.reduce()` method:

```
var obj = arr.reduce(function(acc, cur, i) {
  acc[i] = cur;
  return acc;
}, {});
```

That one also avoids "holes" in the array, because that's how `.reduce()` works.

edited Jan 4 '17 at 21:08

Jordan Frankfurt
129 1 3 13

answered Nov 18 '10 at 14:24

Pointy
331k 47 475 541

2 @m93a it already is an object. In JavaScript, there's really no point creating an Array instance (`[]`) if you're not going to use numeric property keys and the "length" property. – Pointy May 11 '13 at 13:16

1 @m93a All array instances are objects. Properties with string (non-numeric) names like "foo" are merely properties of that object, and as such they're just like properties of any other object. The properties with numeric names (like "12" or "1002") are special in that the runtime system automatically adjusts the "length" property accordingly when values are assigned. Also, `.push()` , `.pop()` , `.slice()` , etc. all only work on numeric properties, and not properties like "foo". – Pointy May 12 '13 at 1:42

13 cleaner way to avoid param reassign `const obj = arr.reduce((obj, cur, i) => { return { ...obj, [i]: cur }; }, {});` – huygn Jan 9 '17 at 8:42

2 Arrow + destructuring syntax w/o explicit return: `const obj = arr.reduce((obj, cur, i) => ({ ...obj, [i]: cur }), {});` – Marc Scheib Mar 27 '18 at 8:23

2 @eugene_sunic indeed, but that approach was not available when I answered this in 2010 :) – Pointy Jul 14 '18 at 15:22

You could use an accumulator aka `reduce` .

```
  return result;
}, {}) //watch out the empty {}, which is passed as "result"
```

Pass an empty object `{}` as a starting point; then "augment" that object incrementally. At the end of the iterations, `result` will be `{"0": "a", "1": "b", "2": "c"}`

If your array is a set of key-value pair objects:

```
[{ a: 1},{ b: 2},{ c: 3}].reduce(function(result, item) {
  var key = Object.keys(item)[0]; //first property: a, b, c
  result[key] = item[key];
  return result;
}, {});
```

will produce: `{a: 1, b: 2, c: 3}`

For the sake of completeness, `reduceRight` allows you to iterate over your array in reverse order:

```
[{ a: 1},{ b: 2},{ c: 3}].reduceRight(/* same implementation as above */)
```

will produce: `{c:3, b:2, a:1}`

Your accumulator can be of any type for you specific purpose. For example in order to swap the key and value of your object in an array, pass `[]` :

```
[{ a: 1},{ b: 2},{ c: 3}].reduce(function(result, item, index) {
  var key = Object.keys(item)[0]; //first property: a, b, c
  var value = item[key];
  var obj = {};
  obj[value] = key;
  result.push(obj);
  return result;
}, []); //an empty array
```

will produce: `[{1: "a"}, {2: "b"}, {3: "c"}]`

Unlike `map` , `reduce` may not be used as a 1-1 mapping. You have full control over the items you want to include or exclude. Therefore `reduce` allows you to achieve what `filter` does, which makes `reduce` very versatile:

```
[{ a: 1},{ b: 2},{ c: 3}].reduce(function(result, item, index) {
  if(index !== 0) { //skip the first item
    result.push(item);
  }
  return result;
}, []); //an empty array
```

will produce: `[{2: "b"}, {3: "c"}]`

Caution: `reduce` and `Object.key` are part of `ECMA 5th edition` ; you should provide a polyfill for browsers that don't support them (notably IE8).

See a default implementation by Mozilla.

edited Sep 1 '17 at 14:18

answered Jan 3 '14 at 18:04

roland

5,780 5 36 57

1 Helpful with maintaining a redux store when you are using a map of objects and you need to drop one of the keys – dhruvpatel Jul 5 at 22:27

99

If you're using jquery:

```
$.extend({}, ['a', 'b', 'c']);
```

answered Dec 17 '11 at 19:53

Max

9,196 4 19 15

3 Weird, if I give in an Array variable it puts every character in a separate object: 0: "a", 1: ",", 2: "b"... So it ignores the quotes, but includes the commas... – TrySpace Jun 20 '12 at 11:22

@Max I think there is no such behavior. The returned object is `{0: 'a', 1: 'b', 2: 'c'}` what is an expected result. – ivkremer Nov 13 '13 at 21:07

3 Is there a way to use $.extend while also informing the keys in a non-numerical manner, ie: making calcultations on the array items? – Davi Lima Nov 13 '14 at 0:21

For completeness, ECMAScript 2015(ES6) spreading. Will require either a transpiler(Babel) or an environment running at least ES6.

53

```
console.log(
   { ...['a', 'b', 'c'] }
)
```

Run code snippet Expand snippet

edited Aug 21 at 8:35

vsync
54.6k 38 176 245

answered Mar 17 '16 at 14:47

mcmhav
753 7 9

1 Actually this isn't natively available in ES2015 either. It is very elegant however. – Aluan Haddad Mar 26 '18 at 3:01

I'd probably write it this way (since very rarely I'll not be having the underscorejs library at hand):

46

```
var _ = require('underscore');

var a = [ 'a', 'b', 'c' ];
var obj = _.extend({}, a);
console.log(obj);
// prints { '0': 'a', '1': 'b', '2': 'c' }
```

edited Jul 19 '16 at 14:47

Prerna Jain
641 9 27

answered Feb 21 '12 at 21:58

Dave Dopson
31.9k 16 79 79

9 you downvote my response, but no comment? My answer is correct and tested. What is the objection? – Dave Dopson Jun 13 '12 at 0:33

14 This question has no underscore tag, and you are also assuming node.js or a require library. – David Hellsing Nov 4 '12 at 9:45

10 underscore is pretty common on both client and server. It's assumed for Backbone.js and is probably THE most common utility library. That being said, I included the require line to make it clear I was using a library. There's no 1-liner for describing "add underscore.js to your page", so some translation is

more appropriate than `_.map` . All in all, this is not a good answer on several levels. – David Hellsing Nov 5 '12 at 13:04

4 The "you must answer without mentioning underscore or lo-dash" people are so annoying. Comparable to saying that you must answer C++ questions without mentioning the standard libraries. If underscore or lo-dash is not an option, the OP should mention that. – Charlie Martin Dec 4 '14 at 0:28

Here is an O(1) ES2015 method just for completeness.

26

```
var arr = [1, 2, 3, 4, 5]; // array, already an object
Object.setPrototypeOf(arr, Object.prototype); // now no longer an array, still an object
```

answered Jun 22 '15 at 22:02

Benjamin Gruenbaum
201k 65 427 452

1 New JavaScript standards!!! this works awesome! – Rick Jul 1 '15 at 18:57

(1) According to MDN, changing the [[Prototype]] of an object is a very slow operation. (2) This does not remove the own `length` property. (3) The object is still an array, `Array.isArray(arr) === true` . (4) Special array behaviors are not removed, e.g. `arr.length = 0` removes all indices. (5) Therefore, I think `Object.assign` is much better. – Oriol Apr 3 '16 at 17:11

1 @Oriol mdn is wrong, at least in V8 (but also other engines) this is a pretty fast operation in this particular case - since objects have a numerical store anyway this is basically changing two pointers. – Benjamin Gruenbaum Jun 24 '16 at 10:17

1 Downvoter - care to explain why? – Benjamin Gruenbaum Jun 17 '18 at 12:19

This also can be used for the fact that it is apparently unique among the other quick solutions, of preserving any non-index ("own") properties on the array (i.e., non-positive-integer properties)... – Brett Zamir Nov 27 '18 at 8:55

Surprised not to see -

23

```
Object.assign({}, your_array)
```

answered Jul 19 '16 at 21:03

Wylliam Judd

18

we can use `Object.assign` and `array.reduce` function to convert an Array to Object.

```
var arr = [{a:{b:1}},{c:{d:2}}]
var newObj = arr.reduce((a, b) => Object.assign(a, b), {})

console.log(newObj)
```

Run code snippet Expand snippet

edited Oct 25 '17 at 15:01

answered Oct 25 '17 at 9:13

KARTHIKEYAN.A

6,430 3 50 60

17

```
Object.assign({}, ['one', 'two']); // {0: 'one', 1: 'two'}
```

Easy way in modern JavaScript is to use `Object.assign()` that does nothing but copying key:value from one object to another. In our case, `Array` donates properties to new `{}` .

edited Mar 20 '17 at 1:46

Paul Rooney

13.5k 7 31 46

answered Nov 14 '16 at 15:57

Appeiron

1,003 7 13

How is this better than any of the already existing answers that gave the very same solution? – Dan Dascalescu Apr 3 at 22:54

Dan Dascalescu at time I've added answer there was only above answer regarding `O.assign` but it lacked explanation. Nowadays destructuring array into object is a way (`{...array}`) – Appeiron Apr 4 at 14:25

I ended up using object spread operator, since it is part of the ECMAScript 2015 (ES6) standard.

Made the following fiddle as an example.

edited Feb 20 '18 at 10:00

answered Sep 23 '17 at 18:02

locropulenton
2,036 1 17 41

1 I'm not sure about the performance of this, but out of so many Array to Object conversion (since i want to use objects for all my codes to standardise it), this is probably the easiest. – Someone Special Apr 28 '18 at 10:27

How is this better than any of the already existing answers that gave the very same solution? – Dan Dascalescu Apr 3 at 22:54

10

Five years later, there's a good way :)

`Object.assign` was introduced in ECMAScript 2015.

```
Object.assign({}, ['a', 'b', 'c'])
// {'0':'a', '1':'b', '2':'c'}
```

answered Aug 24 '16 at 5:08

Paul Draper
45.8k 30 142 217

This answer was already given. – Dan Dascalescu Apr 3 at 22:56

9

you can use spread operator

```
x = [1,2,3,10]
{...x} // {0:1, 1:2, 2:3, 3:10}
```

answered Feb 8 '18 at 5:15

noel zubin

1 This answer was already given several times. – Dan Dascalescu Apr 3 at 22:57

9

If you're using ES6, you can use Object.assign and the spread operator

```
{ ...['a', 'b', 'c'] }
```

If you have nested array like

```
var arr=[[1,2,3,4]]
Object.assign(...arr.map(d => ({[d[0]]: d[1]})))
```

edited Mar 6 '18 at 15:31

answered May 16 '17 at 5:52

murthy naika k
416 5 10

1 Not only are your variable names different, but your example will not work, creating a map from the constructor initially requires 2d key-value Array. `new Map([['key1', 'value1'], ['key2', 'value2']]);` – Shannon Hochkins May 17 '17 at 3:38

9

Using `javascript#forEach` one can do this

```
var result = {},
    attributes = ['a', 'b','c'];

attributes.forEach(function(prop,index) {
  result[index] = prop;
});
```

With ECMA6:

```
attributes.forEach((prop,index)=>result[index] = prop);
```

11.9k 4 17 39

For ES2016, spread operator for objects. Note: This is *after* ES6 and so transpiler will need to be adjusted.

9

```
const arr = ['a', 'b', 'c'];
const obj = {...arr}; // -> {0: "a", 1: "b", 2: "c"}
```

Run code snippet Expand snippet

edited May 17 at 16:06

dman
4,967 12 66 138

answered Sep 20 '18 at 10:46

Oleksii Trekhleb
847 9 16

The spread operator answer was already given 2 years earlier. – Dan Dascalescu Apr 3 at 22:58

Spread operator for objects is *not* ES6! It is ES2016 – dman May 17 at 16:04

FWIW, one another recent approach is to use the new `Object.fromEntries` along with `Object.entries` as follows:

6

```
const arr = ['a','b','c'];
arr[-2] = 'd';
arr.hello = 'e';
arr.length = 17;
const obj = Object.fromEntries(Object.entries(arr));
```

...which allows for avoiding storing sparse array items as `undefined` or `null` and preserves non-index (e.g., non-positive-integer/non-numeric) keys.

One may wish to add `arr.length`, however, as that is not included:

```
obj.length = arr.length;
```

Brett Zamir
10.1k 3 33 58

5

A quick and dirty one:

```
var obj = {},
  arr = ['a','b','c'],
  l = arr.length;

while( l && (obj[--l] = arr.pop() ) ){};
```

edited Aug 31 '15 at 7:54

answered Nov 18 '10 at 14:31

Mic
21.4k 7 49 68

I think you forgot to test that; it produces `{0:"c", 1:"b", 2:"a"}` . You either want `unshift` instead of `pop` or (better) start with `i=arr.length-1` and decrement instead. – Phrogz Nov 18 '10 at 14:38

yeah.. just changed it, but then it become less interesting :/ – Mic Nov 18 '10 at 14:39

Could even do `l = arr.length` , and then `while (l && (obj[--l] = arr.pop())){}` (I realize this is old, but why not simplify it even further). – Qix Aug 24 '15 at 21:14

2 @Qix, Nice shortening – Mic Aug 31 '15 at 7:56

4

Quick and dirty #2:

```
var i = 0
  , s = {}
  , a = ['A', 'B', 'C'];

while( a[i] ) { s[i] = a[i++] };
```

answered Jan 15 '13 at 19:26

Why are you using comma notation? – Conner Ruhl Jan 17 '13 at 1:34

3 Personal preference to have the comma leading on the next line. I find this notation easier to read. – BingeBoy Jun 2 '13 at 3:26

1 The loop will stop if the array contains a falsy value. You should check `i < a.length` instead of `a[i]` . – Oriol Apr 12 '16 at 9:49

4

As of Lodash 3.0.0 you can use _.toPlainObject

```
var obj = _.toPlainObject(['a', 'b', 'c']);
console.log(obj);


<script src="https://cdn.jsdelivr.net/lodash/4.16.4/lodash.min.js"></script>
```

Run code snippet Expand snippet

answered Oct 14 '16 at 17:36

acontell

5,941 1 12 24

:/ Why to load a framework to do simplest things? – Andrés Morales Mar 9 '18 at 20:36

3

This allows you to generate from an array an object with keys you define in the order you want them.

```
Array.prototype.toObject = function(keys){
    var obj = {};
    var tmp = this; // we want the original array intact.
    if(keys.length == this.length){
        var c = this.length-1;
        while( c>=0 ){
            obj[ keys[ c ] ] = tmp[c];
            c--;
```

```
};

result = ["cheese","paint",14,8].toObject([0,"onion",4,99]);
```

`console.log(">>> :" + result.onion);` will output "paint", the function has to have arrays of equal length or you get an empty object.

Here is an updated method

```
Array.prototype.toObject = function(keys){
    var obj = {};
    if( keys.length == this.length)
        while( keys.length )
            obj[ keys.pop() ] = this[ keys.length ];
    return obj;
};
```

edited Jun 7 '14 at 19:43

answered Nov 7 '13 at 23:31

Mark Giblin
669 2 9 16

This destroys both the keys array content and, despite the internal comment, also the values array (its contents). JavaScript works differently than PHP with JavaScript automatically acting by reference on the properties of an object/array. – Brett Zamir Feb 3 '14 at 12:01

No it doesn't, the routine makes copies, the original is not touched. – Mark Giblin Feb 8 '14 at 23:29

Yes, the original is touched. See jsfiddle.net/bRTv7 . The array lengths end up both being 0. – Brett Zamir Feb 8 '14 at 23:37

Ok, the edited version I just changed it with does not destroy the arrays, find that strange that it should have done that. – Mark Giblin Feb 10 '14 at 23:19

`var tmp = this;` had just made a reference, not a copy, as with the keys array you had passed into the function. As mentioned, JavaScript works differently than PHP. I also fixed your current function's while loop condition to be `>=0` instead of `>0` because otherwise you would avoid including the first item in the array (at the 0 index). – Brett Zamir Feb 10 '14 at 23:43

3

Here's a recursive function I just wrote. It's simple and works well.

```
// Convert array to object
```

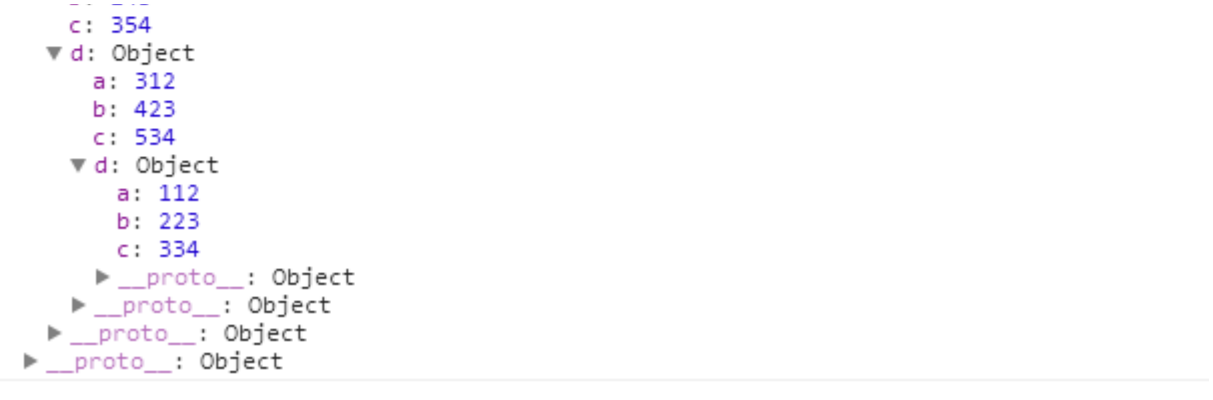
```
        for(var i in array){
            var thisEle = convArrToObj(array[i]);
            thisEleObj[i] = thisEle;
        }
    }else {
        thisEleObj = array;
    }
    return thisEleObj;
}
```

Here's an example (jsFiddle):

```
var array = new Array();
array.a = 123;
array.b = 234;
array.c = 345;
var array2 = new Array();
array2.a = 321;
array2.b = 432;
array2.c = 543;
var array3 = new Array();
array3.a = 132;
array3.b = 243;
array3.c = 354;
var array4 = new Array();
array4.a = 312;
array4.b = 423;
array4.c = 534;
var array5 = new Array();
array5.a = 112;
array5.b = 223;
array5.c = 334;

array.d = array2;
array4.d = array5;
array3.d = array4;
array.e = array3;


console.log(array);

// Convert array to object
var convArrToObj = function(array){
    var thisEleObj = new Object();
```

```
                thisEleObj[i] = thisEle;
            }
        }else {
            thisEleObj = array;
        }
        return thisEleObj;
    }
    console.log(convArrToObj(array));
```

```
▼ [a: 123, b: 234, c: 345, d: Array[0], e: Array[0]]
    a: 123
    b: 234
    c: 345
  ▼ d: Array[0]
      a: 321
      b: 432
      c: 543
      length: 0
    ▶ __proto__: Array[0]
  ▼ e: Array[0]
      a: 132
      b: 243
      c: 354
    ▼ d: Array[0]
        a: 312
        b: 423
        c: 534
      ▼ d: Array[0]
          a: 112
          b: 223
          c: 334
          length: 0
        ▶ __proto__: Array[0]
        length: 0
      ▶ __proto__: Array[0]
      length: 0
    ▶ __proto__: Array[0]
▼ Object {a: 123, b: 234, c: 345, d: Object, e: Object}
    a: 123
    b: 234
    c: 345
  ▼ d: Object
      a: 321
      b: 432
      c: 543
```



Results:

answered Jul 14 '14 at 1:21

JVE999

1,899 5 32 59

Works really well. Thanks – Hanmaslah Jan 26 '17 at 7:33

This should be higher up, if you have `['a' = '1', 'b' = '2', 'c' = '3']` and want it like `{a: 1, b: 2, c: 3}` this works perfect. – Wanjia Jul 23 '17 at 15:17

3

```
.reduce((o,v,i)=>(o[i]=v,o), {})
```

[docs]

or more verbose

```
var trAr2Obj = function (arr) {return arr.reduce((o,v,i)=>(o[i]=v,o), {});}
```

or

```
var transposeAr2Obj = arr=>arr.reduce((o,v,i)=>(o[i]=v,o), {})
```

```
JSON.stringify([["a", "X"], ["b", "Y"]].reduce((o,v,i)=>{return o[i]=v,o}, {}))
=> "{"0":["a","X"],"1":["b","Y"]}"
```

some more complex example

```
[["a", "X"], ["b", "Y"]].reduce((o,v,i)=>{return o[v[0]]=v.slice(1)[0],o}, {})
=> Object {a: "X", b: "Y"}
```

even shorter (by using `function(e) {console.log(e); return e;}` === `(e)=>(console.log(e),e)`)

```
 nodejs
> [[1, 2, 3], [3,4,5]].reduce((o,v,i)=>(o[v[0]]=v.slice(1),o), {})
{ '1': [ 2, 3 ], '3': [ 4, 5 ] }
```

[/docs]

edited Jun 6 '16 at 21:40

answered Jun 6 '16 at 21:34

test30

2,374 23 23

What did you intend with `[/docs]` ? It would be more useful to make the code snippets executable. – Dan Dascalescu Apr 3 at 23:00

3

I would do this simply with `Array.of()` . Array of has the ability to use it's context as a constructor.

> NOTE 2 The of function is an intentionally generic factory method; it does not require that its this value be the Array constructor. Therefore it can be transferred to or inherited by other constructors that may be called with a single numeric argument.

So we may bind `Array.of()` to a function and generate an array like object.

```
function dummy(){};
var thingy = Array.of.apply(dummy,[1,2,3,4]);
console.log(thingy);
```

By utilizing `Array.of()` one can even do array sub-classing.

edited May 23 '17 at 10:31
Community ♦
1 1

answered Oct 14 '16 at 16:37
Redu
13.9k 2 26 40

3

If you can use `Map` or `Object.assign` , it's very easy.

Create an array:

```
const languages = ['css', 'javascript', 'php', 'html'];
```

The below creates an object with index as keys:

```
Object.assign({}, languages)
```

Replicate the same as above with Maps

Converts to an index based object `{0 : 'css'}` etc...

```
const indexMap = new Map(languages.map((name, i) => [i, name] ));
indexMap.get(1) // javascript
```

Convert to an value based object `{css : 'css is great'}` etc...

```
const valueMap = new Map(languages.map(name => [name, `${name} is great!`] ));
valueMap.get('css') // css is great
```

answered May 17 '17 at 3:36
Shannon Hochkins
5,328 11 42 76

2

Using **Array** prototype function **'push'** and **'apply'** you can populate the object with the array elements.

```
var arr = ['a','b','c'];
var obj = new Object();
Array.prototype.push.apply(obj, arr);
console.log(obj);    // { '0': 'a', '1': 'b', '2': 'c', length: 3 }
console.log(obj[2]); // c
```

Run code snippet Expand snippet

answered Jan 4 at 16:13

SridharKritha
2,993 1 25 27

What if I want to convert to `{ name: a, div :b, salary:c}` – Prashant Pimpale Jan 4 at 16:41

1

You could use a function like this:

```
var toObject = function(array) {
    var o = {};
    for (var property in array) {
        if (String(property >>> 0) == property && property >>> 0 != 0xffffffff) {
            o[i] = array[i];
        }
    }
    return o;
};
```

This one should handle sparse arrays more efficiently.

edited Nov 18 '10 at 17:03

answered Nov 18 '10 at 14:37

Pablo Cabrera
4,872 4 19 27

Here's a solution in coffeescript

1

```
arrayToObj = (arr) ->
  obj = {}
  for v,i in arr
    obj[i] = v if v?
  obj
```

answered Jan 31 '13 at 19:43

David
342 2 9

7 What's coffeescript? We are talking about JS! :D – m93a May 11 '13 at 20:42

2 It's a little language that compiles into JavaScript :) – David May 12 '13 at 21:37

6 Hmph, it uses strange notation. I like plain JS more. – m93a May 13 '13 at 14:31

2 @David You could do the entire for loop on 1 line to make the whole thing only 3 lines. :) Example: `obj[i] = v for v,i in arr when v?` – XåppII'-I0llwlg'I - Mar 20 '15 at 3:10

If you are using angularjs you can use angular.extend, the same effect with $.extend of jquery.

1

```
var newObj = {};
angular.extend(newObj, ['a','b','c']);
```

answered Feb 24 '15 at 6:40

Vicruz
296 1 12

It's not directly relevant but I came here searching for a one liner for merging nested objects such as

1

```
const nodes = {
```

```
    node2: {
        interfaces: {if3: {}, if4: {}}
    },
    node3: {
        interfaces: {if5: {}, if6: {}}
    },
}
```

The solution is to use a combination of reduce and object spread:

```
const allInterfaces = nodes => Object.keys(nodes).reduce((res, key) => ({...res,
...nodes[key].interfaces}), {})
```

answered Jul 15 '17 at 21:46

Adnan Y
1,872 1 14 24