Most popular (last 30 days)

ECMAScript 2017: the final feature set
ECMAScript 6 modules: the final syntax
ES proposal: import() – dynamically importing ES modules
Making transpiled ES modules more spec-compliant
ES proposal: Shared memory and atomics
Classes in ECMAScript 6 (final semantics)
Communicating between Web Workers via MessageChannel

Most popular (all time)

ECMAScript 6 modules: the final syntax
Classes in ECMAScript 6 (final semantics)
Iterating over arrays and objects in JavaScript
ECMAScript 6's new array methods
The final feature set of ECMAScript 2016 (ES7)
WebAssembly: a binary format for the web
Basic JavaScript for the impatient programmer
Google Dart to "ultimately ... replace JavaScript"
Six nifty ES6 tricks
Google's Polymer and the future of web UI frameworks

Blog archive

► 2017 (4)
► 2016 (38)

2014-05-08

ECMAScript 6's new array methods

Labels: dev, esnext, javascript

Check out my book (free online): "Exploring ES6". Updated version of this blog post: chapter "New Array features".

This blog post explains what new array methods ECMAScript 6 will bring and how to use them in current browsers.

Note: I'm using the terms *constructor* and *class* interchangeably.

1. Class methods

`Array` has gained methods of its own.

1.1. Array.from(arrayLike, mapFunc?, thisArg?)

`Array.from()`'s basic functionality is to convert two kinds of objects to arrays:

- Array-like objects, which have a property `length` and indexed elements. Examples include the results of DOM operations such as `document.getElementsByClassName()`.
- Iterable objects, whose contents can be retrieved one element at a time. Arrays are iterable, as are ECMAScript's new data structures `Map` and `Set`.

The following code is an example of converting an array-like object to an array:

```
let lis = document.querySelectorAll('ul.fancy li');
Array.from(lis).forEach(function (li) {
    console.log(node);
});
```

The result of `querySelectorAll()` is not an array and does not have a `forEach()` method, which is why we need to convert it to an array before we can use that method.

Mapping via Array.from()

`Array.from()` is also a convenient alternative to using `map()` generically:

```
let spans = document.querySelectorAll('span.name');

// map(), generically:
let names1 = Array.prototype.map.call(spans, s => s.textContent);

// Array.from():
let names2 = Array.from(spans, s => s.textContent);
```

The second parameter of both methods is an arrow function.

In this example, the result of `document.querySelectorAll()` is again an array-like object, not an array, which is why we couldn't invoke `map()` on it. Previously, we converted the array-like object to an array in order to call `forEach()`. Here, we skipped

Dr. Axel Rauschmayer

► 2015 (65)
▼ 2014 (55)
► December (4)
► November (1)
► October (2)
► September (4)
► August (3)
► July (7)
► June (7)
▼ May (8)
Checking whether a value is an integer in JavaScri...
Implementing a raffle in JavaScript
Understanding the four layers of JavaScript OOP
We're doing syntax highlighting wrong
ECMAScript 6's new array methods
Meet the site of the Web Components community
Identifying the current <script> element
JavaScript's "this": how it works, where it can tr...
► April (6)
► March (1)
► February (5)
► January (7)
► 2013 (98)
► 2012 (177)
► 2011 (380)
► 2010 (174)
► 2009 (68)
► 2008 (46)
► 2007 (12)
► 2006 (1)
► 2005 (2)

Labels

dev (592)
javascript (400)
computers (316)
life (194)
jslang (179)
esnext (156)
apple (107)

that intermediate step via a generic method call and via the two-parameter version of `Array.from()`.

Holes

`Array.from()` ignores holes [3] in arrays, it treats them as if they were `undefined` elements:

```
> Array.from([0,,2])
[ 0, undefined, 2 ]
```

That means that you can use `Array.from()` to create and fill an array:

```
> Array.from(new Array(5), () => 'a')
[ 'a', 'a', 'a', 'a', 'a' ]
> Array.from(new Array(5), (x,i) => i)
[ 0, 1, 2, 3, 4 ]
```

If you want to fill an array with a fixed value (first one of the previous two examples) then `Array.prototype.fill()` (see below) is a better choice.

from() in subclasses of Array

Another use case for `Array.from()` is to convert an array-like or iterable object to an instance of a subclass of `Array`. For example, if you create a subclass `MyArray` of `Array` (subclassing arrays is explained in [1]) and want to convert such an object to an instance of `MyArray`, you simply use `MyArray.from()`. The reason that works is because constructors inherit from each other in ECMAScript 6 (a super-constructor is the prototype of its sub-constructors).

```
class MyArray extends Array {
    ...
}
let instanceOfMyArray = MyArray.from(anIterable);
```

You can also combine this functionality with mapping, to get a map operation where you control the result's constructor:

```
// from() – determine the result's constructor via the receiver
// (in this case, MyArray)
let instanceOfMyArray = MyArray.from([1, 2, 3], x => x * x);

// map(): the result is always an instance of Array
let instanceOfArray   = [1, 2, 3].map(x => x * x);
```

1.2. Array.of(...items)

If you want to turn several values into an array, you should always use an array literal, especially since the array constructor doesn't work properly if there is a single value that is a number (more information on this quirk):

```
> new Array(3, 11, 8)
[ 3, 11, 8 ]
> new Array(3)
[ , , ,]
> new Array(3.1)
RangeError: Invalid array length
```

But how are you supposed to turn values into an instance of a sub-constructor of `Array` then? This is where `Array.of()` helps (remember that sub-constructors of `Array` inherit all of `Array`'s methods, including `of()`).

```
class MyArray extends Array {
    ...
}
console.log(MyArray.of(3, 11, 8) instanceof MyArray); // true
console.log(MyArray.of(3).length === 1); // true
```

Tweets by @rauschma

Axel Rauschm... @rauschma
Enjoying "Troll Hunters":
– "Let's call him 'Gnome Chomsky'"
– "Juliet dies in this? Nooo!"
10h

Axel Rauschmayer Retweeted
Jonathan Creamer @jcreamer898
A nice little shout out to @rauschma... infoworld.com/article/316483 ... #es2017 #async
JavaScr...
ECMASc...
infoworld...
13h

Axel Rauschmayer @rauschma
Not a physics book! twitter.com/ManningBooks/s ...
02 Feb

Axel Rauschmayer Retweeted
Jordan Harband @ljharb
Making our React components forbid extra props has caught SO many bugs. I highly recommend it.

npmjs.com/airbnb-prop-ty...
npm: air...
Custom ...
npmjs.com
02 Feb

Axel Rauschmayer Retweeted
Seth Petry-Johnson @spetryjohnson
Open strong. Be bold. Tell a story. Do something to get me interested. Opening w/ the "obligatory 'about me' slide" puts me to sleep :)
02 Feb

Google

webdev (95)
mobile (83)
scitech (50)
hack (49)
mac (47)
google (39)
java (37)
ios (33)
business (32)
video (32)
clientjs (31)
hci (27)
entertainment (26)
nodejs (26)
society (26)
browser (25)
firefox (25)
html5 (24)
ipad (24)
movie (23)
psychology (22)
2ality (18)
tv (18)
android (17)
social (17)
chrome (16)
fun (16)
jsmodules (16)
tablet (16)
humor (15)
politics (15)
web (15)
cloud (14)
hardware (14)
microsoft (14)
software engineering (13)
blogging (12)
gaming (12)
eclipse (11)
gwt (11)
numbers (11)
programming languages (11)
app store (10)
media (10)
nature (10)
security (10)
semantic web (10)
software (10)
twitter (10)
webos (10)
12quirks (9)
education (9)
jstools (9)

`Array.of()` is also handy as a function that doesn't have `Array()`'s quirk related to wrapping values in arrays. However, be careful about an `Array.prototype.map()` pecularity that can trip you up here:

```
> ['a', 'b'].map(Array.of)
[ [ 'a', 0, [ 'a', 'b' ] ],
  [ 'b', 1, [ 'a', 'b' ] ] ]
> ['a', 'b'].map(x => Array.of(x)) // better
[ [ 'a' ], [ 'b' ] ]
> ['a', 'b'].map(x => [x]) // best (in this case)
[ [ 'a' ], [ 'b' ] ]
```

As you can see above, `map()` passes three parameters to its callback, the last two are simply often ignored (details).

2. Prototype methods

Several new methods are available for array instances.

2.1. Iterating over arrays

The following methods help with iterating over arrays:

- `Array.prototype.entries()`
- `Array.prototype.keys()`
- `Array.prototype.values()`

The result of each of the aforementioned methods is a sequence of values, but they are not returned as an array; they are revealed one by one, via an iterator. Let's look at an example (I'm using `Array.from()` to put the iterators' contents into arrays):

```
> Array.from([ 'a', 'b' ].keys())
[ 0, 1 ]
> Array.from([ 'a', 'b' ].values())
[ 'a', 'b' ]
> Array.from([ 'a', 'b' ].entries())
[ [ 0, 'a' ],
  [ 1, 'b' ] ]
```

You can combine `entries()` with ECMAScript 6's `for-of` loop [2] and destructuring to conveniently iterate over (index, element) pairs:

```
for (let [index, elem] of ['a', 'b'].entries()) {
    console.log(index, elem);
}
```

Note: this code already works in the current Firefox.

2.2. Searching for array elements

`Array.prototype.find(predicate, thisArg?)`
returns the first array element for which the callback `predicate` returns `true`. If there is no such element, it returns `undefined`. Example:

```
> [6, -5, 8].find(x => x < 0)
-5
> [6, 5, 8].find(x => x < 0)
undefined
```

`Array.prototype.findIndex(predicate, thisArg?)`
returns the index of the first element for which the callback `predicate` returns `true`. If there is no such element, it returns `-1`. Example:

```
> [6, -5, 8].findIndex(x => x < 0)
1
> [6, 5, 8].findIndex(x => x < 0)
-1
```

Both `find*` methods ignore holes [3]. The full signature of the callback `predicate` is:

photo (9)
webcomponents (9)
windows 8 (9)
async (8)
idea (8)
iphone (8)
itunes (8)
scifi-fantasy (8)
app (7)
babel (7)
bookmarklet (7)
chromeos (7)
english (7)
es proposal (7)
fringe (7)
html (7)
jsint (7)
jsshell (7)
thunderbolt (7)
webapp (7)
advancedjs (6)
blogger (6)
crowdsourcing (6)
latex (6)
lion (6)
promises (6)
ted (6)
book (5)
environment (5)
gadget (5)
googleio (5)
intel (5)
jsarrays (5)
jshistory (5)
layout (5)
light peak (5)
michael j. fox (5)
music (5)
pdf (5)
polymer (5)
shell (5)
tc39 (5)
template literals (5)
underscorejs (5)
vlc (5)
__proto__ (4)
coffeescript (4)
concurrency (4)
dart (4)
facebook (4)
gimp (4)
googleplus (4)
health (4)
howto (4)

```
predicate(element, index, array)
```

Finding `NaN` via `findIndex()`

A well-known limitation of `Array.prototype.indexOf()` is that it can't find `NaN`, because it searches for elements via `===`:

```
> [NaN].indexOf(NaN)
-1
```

With `findIndex()`, you can use `Object.is()` [4] and will have no such problem:

```
> [NaN].findIndex(y => Object.is(NaN, y))
0
```

You can also adopt a more general approach, by creating a helper function `elemIs()`:

```
> function elemIs(x) { return Object.is.bind(Object, x) }
> [NaN].findIndex(elemIs(NaN))
0
```

2.3. Array.prototype.fill(value, start?, end?)

Fills an array with the given `value`:

```
> ['a', 'b', 'c'].fill(7)
[ 7, 7, 7 ]
```

Holes [3] get no special treatment:

```
> new Array(3).fill(7)
[ 7, 7, 7 ]
```

Optionally, you can restrict where the filling starts and ends:

```
> ['a', 'b', 'c'].fill(7, 1, 2)
[ 'a', 7, 'c' ]
```

3. When can I use the new array methods?

- Some of them are already available in browsers. As usual, check kangax's ECMAScript 6 compatibility table.
- Paul Miller's es6-shim library has backported them to ECMAScript 5.

4. References

[1]: Subclassing builtins in ECMAScript 6 [2]: Iterators and generators in ECMAScript 6 [3]: "Holes in Arrays" (Speaking JavaScript) [4]: Stricter equality in JavaScript

11 Comments **The 2ality blog** 1 **Login**

Recommend 6 **Share** **Sort by Best**

Join the discussion…

Šime Vidas • 3 years ago
Good overview!

1. Instead of "constructor methods", how about the term "static Array functions"? It's a good complement to the term "Array methods", I think.

2. In section 1.2., could you first explicitly define what Array.of() does? That section is a bit confusing without the definition.

4 • Reply • Share ›

hp (4)
javafx (4)
kindle (4)
leopard (4)
macbook (4)
motorola (4)
münchen (4)
occupy (4)
pl fundamentals (4)
presenting (4)
publishing (4)
series (4)
textbook (4)
web design (4)
amazon (3)
asmjs (3)
back to the future (3)
bitwise_ops (3)
css (3)
es2016 (3)
flattr (3)
fluentconf (3)
food (3)
foreign languages (3)
house (3)
icloud (3)
info mgmt (3)
jsfuture (3)
jsstyle (3)
linux (3)
mozilla (3)
python (3)
regexp (3)
samsung (3)
tizen (3)
traffic (3)
typedjs (3)
unix (3)
adobe (2)
angry birds (2)
angularjs (2)
astronomy (2)
audio (2)
comic (2)
design (2)
dom (2)
ecommerce (2)
eval (2)
exploring es6 (2)
facebook flow (2)
facets (2)
flash (2)
free (2)
futurama (2)

Šime Vidas ➦ Šime Vidas • 3 years ago
Correction: "Static Array methods" is best, I think.
2 ^ | ˅ • Reply • Share ›

Axel Rauschmayer Mod ➦ Šime Vidas • 3 years ago
Good point. My current favorite is “class methods”. I changed the term in the post.
^ | ˅ • Reply • Share ›

jcrben • a year ago
"But how are you supposed to turn values into an instance of a sub-constructor of Array then?"

Could you elaborate on where this is helpful?
^ | ˅ • Reply • Share ›

Florian F. • 2 years ago
No mention of Array#copyWithin :) ?
^ | ˅ • Reply • Share ›

Axel Rauschmayer Mod ➦ Florian F. • a year ago
I missed in when I wrote this blog post (maybe it wasn’t in the spec yet, back then). But it is described in “Exploring ES6”: http://exploringjs.com/es6/ch_...
^ | ˅ • Reply • Share ›

vrunoa • 2 years ago
+1
^ | ˅ • Reply • Share ›

Colin May • 3 years ago
Thanks, as always, for the clear and useful article. Note that the footnote markers within the article appear as links (e.g. change style on mouseover) but do nothing.
^ | ˅ • Reply • Share ›

Henrique Silvério • 3 years ago
Thanks for this nice article.
Exciting ES6 news! =]
^ | ˅ • Reply • Share ›

azu • 3 years ago
s/es5-shim/es6-shim/
^ | ˅ • Reply • Share ›

Axel Rauschmayer Mod ➦ azu • 3 years ago
Indeed. Fixed, thanks!
^ | ˅ • Reply • Share ›

Subscribe **Add Disqus to your site Add Disqus Add** **Privacy**

Newer Post   Home   Older Post

Subscribe to: Post Comments (Atom)

guide (2)
history (2)
hyena (2)
internet explorer (2)
iteration (2)
journalism (2)
jquery (2)
jsengine (2)
jslib (2)
law (2)
lightning (2)
markdown (2)
math (2)
meego (2)
month (2)
nike (2)
nokia (2)
npm (2)
programming (2)
raffle (2)
repl (2)
servo (2)
sponsor (2)
steve jobs (2)
travel (2)
typescript (2)
usb (2)
winphone (2)
wwdc (2)
airbender (1)
amdefine (1)
aol (1)
app urls (1)
architecture (1)
atscript (1)
basic income (1)
biology (1)
blink (1)
bluetooth (1)
canada (1)
clip (1)
coding (1)
community (1)
cross-platform (1)
deutsch (1)
diaspora (1)
distributed-social-network (1)
dsl (1)
dvd (1)
dzone (1)
emacs (1)
emberjs (1)
energy (1)
esnext news (1)

esprop (1)
example (1)
facetator (1)
feedback (1)
firefly (1)
firefoxos (1)
fritzbox (1)
german (1)
git (1)
guest (1)
guice (1)
h.264 (1)
home entertainment (1)
hosting (1)
htc (1)
ical (1)
jsdom (1)
jsmyth (1)
library (1)
location (1)
marketing (1)
mars (1)
meta-data (1)
middle east (1)
mpaa (1)
msl (1)
mssurface (1)
netflix (1)
nsa (1)
obama (1)
openoffice (1)
opinion (1)
oracle (1)
organizing (1)
philosophy (1)
pixar (1)
pnacl (1)
prism (1)
privacy (1)
proxies (1)
puzzle (1)
raspberry pi (1)
read (1)
rodney (1)
rust (1)
safari (1)
sponsoring (1)
star trek (1)
static generation (1)
talk (1)
technique (1)
theora (1)
thunderbird (1)
typography (1)

unicode (1)
v8 (1)
voice control (1)
webassembly (1)
webkit (1)
webm (1)
webpack (1)
yahoo (1)