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)

2011-08-24

Spreading arrays into arguments in JavaScript

Labels: dev, javascript, jslang

Update 2012-02-01: Complete rewrite of the section on spreading constructor arguments.

Sometimes, one needs to *spread* the elements of an array, to use them as the arguments of a function call. JavaScript allows you to do that via `Function.prototype.apply`, but that does not work for constructor invocations. This post explains spreading and how to make it work in the presence of the new operator.

1. Spreading

Spreading means making a function call, method call or constructor invocation and supplying the arguments via an array. In Python, one calls this *unpacking*. ECMAScript.next will have the spread operator (prefix `...`) for this purpose. In current JavaScript, you can perform this operation via `Function.prototype.apply`.

2. Spreading function arguments

`Math.max()` returns the maximum among its 0 or more numeric parameters. With the spread operator, you can use it for arrays:

```
Math.max(...[13, 7, 30])
```

This is the equivalent of

```
Math.max(13, 7, 30)
```

In current JavaScript, you have to use `apply()`.

```
> Math.max.apply(null, [13, 7, 30])
30
```

Explanation: An `apply` invocation looks as follows:

```
func.apply(thisValue, [param1, param2, ...])
```

which is equivalent to

```
thisValue.func(param1, param2, ...)
```

Note that `func` does not have to be a method of `thisValue` – `apply` temporarily turns it into one.

3. Spreading constructor arguments

The `Date` constructor takes several numeric parameters and produces a date. With the spread operator, you can hand in an array.

```
new Date(...[2011, 11, 24]) // Christmas Eve 2011
```

However, this time we cannot use `apply` to implement spreading, because it does not work with `new`:

Dr. Axel Rauschmayer

Free online books by Axel

Speaking JavaScript [up to ES5]
Exploring ES6

JavaScript training: Ecmanauten

- ▶ 2015 (65)
- ▶ 2014 (55)
- ▶ 2013 (98)
- ▶ 2012 (177)
- ▼ 2011 (380)
  - ▶ December (22)
  - ▶ November (25)
  - ▶ October (27)
  - ▶ September (15)
  - ▼ August (24)
    - Twitter bookmarklet tip: associate a name with a w...
    - How to write and unit-test universal JavaScript mo...
    - Team sizes: webOS versus Android versus OS X
    - Spreading arrays into arguments in JavaScript
    - Chrome Native Client: write C++ apps against HTML5...
    - Firefox's extensions get a security review, Chrome...
    - HP lost its way
    - Google's options for Motorola in light of its Andr...
    - HP will spin off PC division, stop making webOS ha...
    - Firefox: the disappearing version number
    - TV: Alphas
    - An introduction to JSDoc
    - What is the correct media type for JavaScript sour...
    - Press review: Google buys Motorola Mobility
    - Bluetooth low energy – two years on a coin-cell ba...
    - JavaScript's JSON API
    - Patents: Google versus Microsoft

```
> new Date.apply(null, [2011, 11, 24])
TypeError: function apply() { [native code] } is not a constructor
```

`new` expects `Date.apply` to be a constructor function. No matter how you parenthesize the above expression, the fundamental problem remains: `apply` performs a function call, it does not hand arguments to the `new` operator.

3.1. A manual work-around

First step. Let's get the result right first, we'll worry later about handing in an array instead of separate arguments.

```
new (Date.bind(null, 2011, 11, 24))
```

We used `bind()` to produce a function with zero parameters (by providing them ahead of time) and then invoked that nullary function via `new`. `bind` has the following signature:

```
func.bind(thisValue, arg1, arg2, ...)
```

It turns `func` into a fresh function whose implicit `this` parameter is `thisValue` and whose initial arguments are always as given. When one invokes the fresh function, the arguments of such an invocation are appended to what has already been provided via `bind`. MDN has more details. Note that in the previous example, the first argument was `null`, because `bind` turns `Date` into a function that does not need a `thisValue`: It is only invoked as a constructor and `new` overrides the `thisValue` from `bind`.

Second step. We want to hand an array to `bind`. So we again use `apply` to turn an array into arguments for a function call.

```
new (Function.prototype.bind.apply(
    Date, [null].concat([2011, 11, 24])))
```

We invoked `apply` on the function `Function.prototype.bind`, with two arguments:

- 1st argument: `this` has the value `Date`, as in `Date.bind(...)`, above.
- 2nd argument: The arguments for `bind` are created by prepending `null` to the array `[2011, 11, 24]`.

3.2. A library method

Mozilla suggests turning the above work-around into a library method. The following is a slightly edited version of their suggestion:

```
if (!Function.prototype.construct) {
    Function.prototype.construct = function(argArray) {
        if (! Array.isArray(argArray)) {
            throw new TypeError("Argument must be an array");
        }
        var constr = this;
        var nullaryFunc = Function.prototype.bind.apply(
            constr, [null].concat(argArray));
        return new nullaryFunc();
    };
}
```

Interaction:

```
> Date.construct([2011, 11, 24])
Sat Dec 24 2011 00:00:00 GMT+0100 (CET)
```

3.3. A seemingly simpler solution

You can manually perform the actions of the `new` operator. For example:

```
var foo = new Foo("abc");
```

Is equivalent to

```
var foo = Object.create(Foo.prototype);
Foo.call(foo, "abc");
```

With that work-around, we can write a simpler library method (checks are omitted):

```
Function.prototype.construct = function(argArray) {
    var constr = this;
    var inst = Object.create(constr.prototype);
    constr.apply(inst, argArray);
```

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...
14h

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

```
        return inst;
    };
```

Alas, `Date` invoked as a function works the same as `Date` invoked as a constructor: It ignores the first parameter of `call()` and `apply()` and always produces a new instance.

```
    > Date.construct([2011, 11, 24])
    {}
```

You can see that the result of `construct()` has not been set up. Several built-in constructors work the same as `Date`. But the library method will work properly with most non-built-in constructors (handling constructors that actively return values require a bit more work).

Labels

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)
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)

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

Recommend 1 **Share** **Sort by Best**

Join the discussion…

igauravsehrawat • 3 months ago

In the note section developer.mozilla.org/en-US/do... MDN says
```
Note: This non-native Function.construct method will not work with some native constructors (like Date, for example). In these cases you have to use the Function.prototype.bind method (for example, imagine having an array like the following, to be used with Date constructor: [2012, 11, 4]; in this case you have to write something like: new (Function.prototype.bind.apply(Date, [null].concat([2012, 11, 4])))()
)
```
And then adds "anyhow this is not the best way to do things and probably should not be used in any production environment."
Any idea why?

Reply • Share ›

Serge Zarouski • a year ago

Another good use case is to use spread instead of Array.from - results seems to be the same, while code is shorter. What do you think?

Reply • Share ›

Axel Rauschmayer Mod → Serge Zarouski • a year ago

Yes, but note that this is a blog post about ES5, not about ES6.

1 • Reply • Share ›

Vasyl Zuzyak • 4 years ago

Hi!

Sorry for bothering so old post but...

I faced with case where I need to pass values (that are in array) as parameters into function call. That function (and object it belongs to) is quite tricky. Due to this I am not able to use apply() method. I have checked this precisely and its sad but true. That function is not a constructor and I am also limited to JavaScript 1.5/ECMA-262 3rd edition so bind() will not help me either.

Do you know any trick that can help me? Otherwise I am forced to support large block of absolutely stupid code in my app..

Thanks in advance

Reply • Share ›

Robert Kuzelj • 4 years ago

Wouldn't it be nice if javascript supported Function.prototype.construct as callback into "new"

Reply • Share ›

Axel Rauschmayer Mod → Robert Kuzelj • 4 years ago

Can you explain what you mean by that?

Reply • Share ›

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

- 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)
- 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)

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)

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)
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)