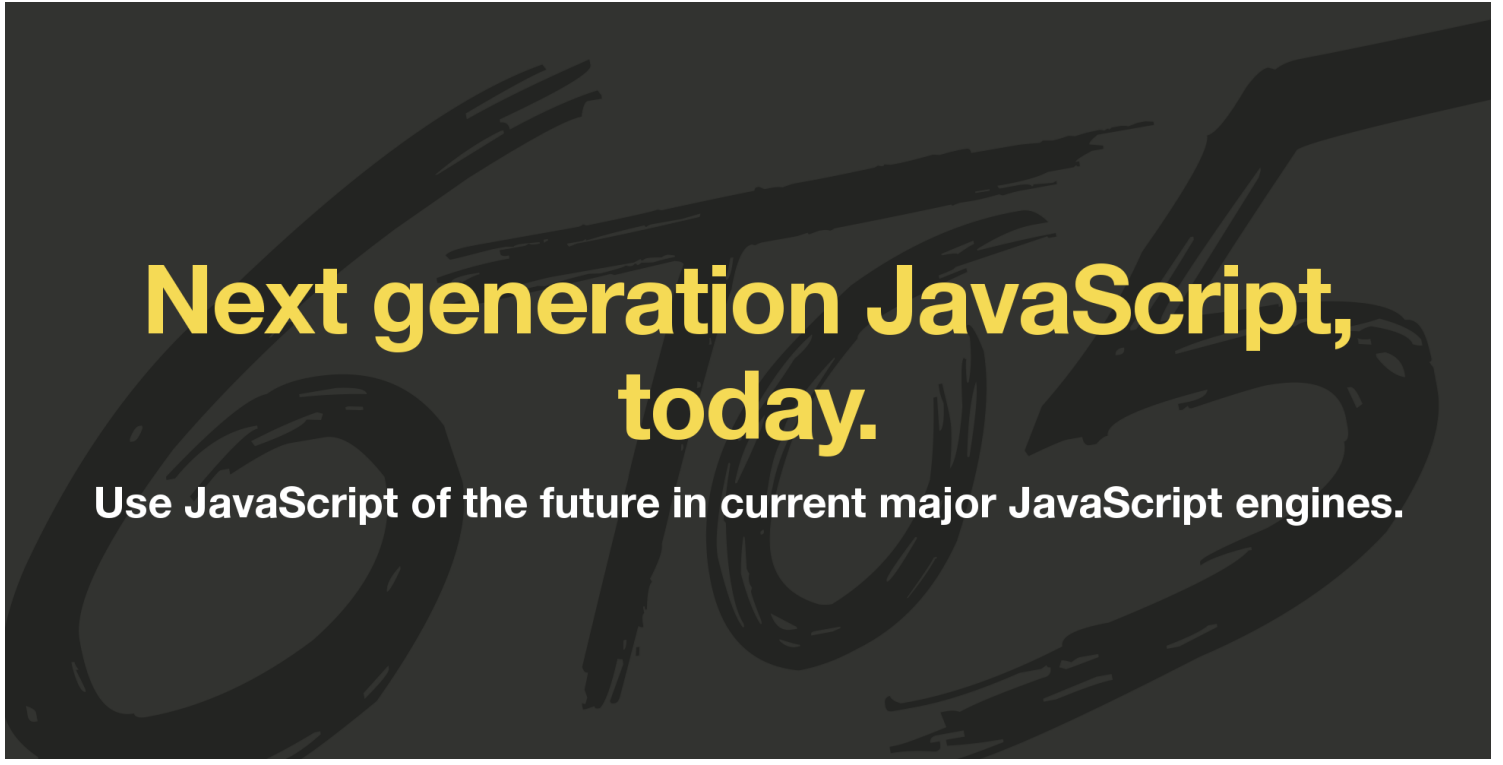
ES6 Now: The Good, The Bad, and The Hype

A technical walkthrough exploring
modern VS long standing patterns
with the JavaScript language

by Andrea Giammarchi
@WebReflection

How ES6 Has Been Promoted

How ES6 Has Been Promoted



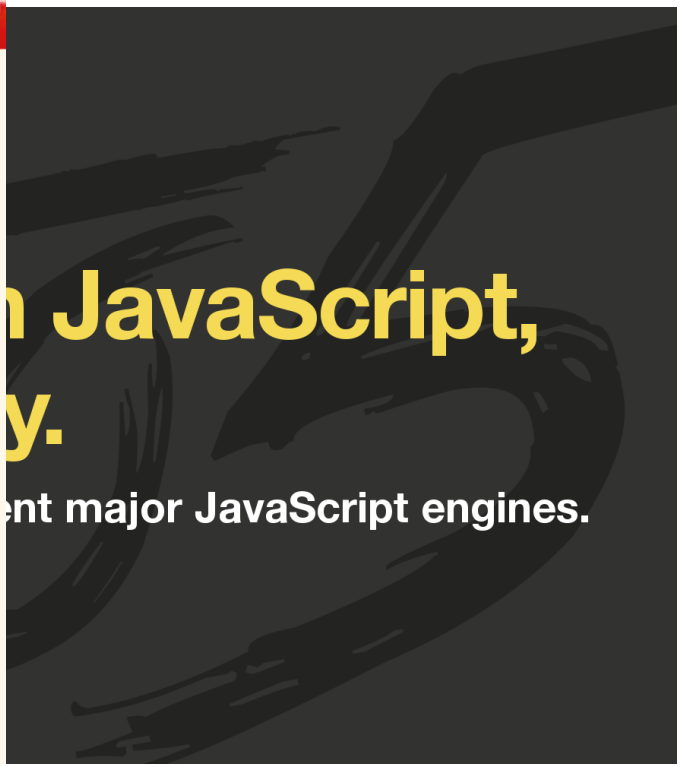
How ES6 Has Been Promoted

Tue May 27 2014 02:18:47 GMT-0300 (BRT)

ES6 - THE FUTURE IS HERE

*A talk by **Sebastiano Armeli** showing some of the ES6 features like scoping, generators, collections, modules and proxies.*

talks



How ES6 Has Been Promoted

Tue May 27 2014 02:18:

ES6 - THE F
IS HERE

*A talk by **Sebastian***
some of the ES6 feat
generators, collectio
proxies.

talks

2014-08-01

Using ECMAScript 6 today

Labels: dev, esnext, javascript, jslang

Updates:

- [2014-11-05] Restructured the post and added more content (terminology, let, promises, ES7+).
- [2014-09-02] Rewrote Sect. "More material on ECMAScript 6"; mentioned iterators, generators and promises.

ECMAScript 6 (ES6) still sounds like something from a far-away future. After all, it will only become a standard by mid 2015. However, its features are continually appearing in browsers and there are compilers that translate ES6 code to ES5 code. The latter is already a compelling solution, because the ECMAScript 6 feature set is already frozen.

This blog post gives a brief overview of ECMAScript 6 features and describes tools that enable you to use them today.

How ES6 Has Been Promoted

Tue May 27 2014 02:18:

ES6 - THE F
IS H

A talk
some
gener
proxie

talks

2014-08-01

Using ECMAScript 6 today

Labels: dev, esnext, javascript, jslang

Updates:

ogy, let,

ed

r all, it will only
earing in
e latter is already
zen.

This blog post gives a brief overview of ECMAScript 6 features and describes tools that enable you to use them today.

NEWS

Use ECMAScript 6 Today

by Sayanee Basu 7 May 2013 32 Comments

Today, ECMAScript 6 is in the process of being finalized. ECMAScript is the

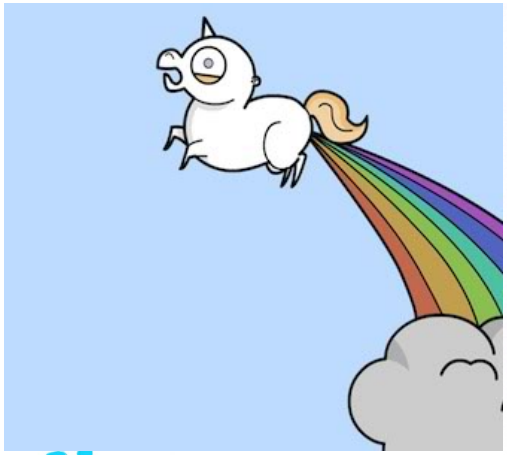
How ES6 Has Been Promoted

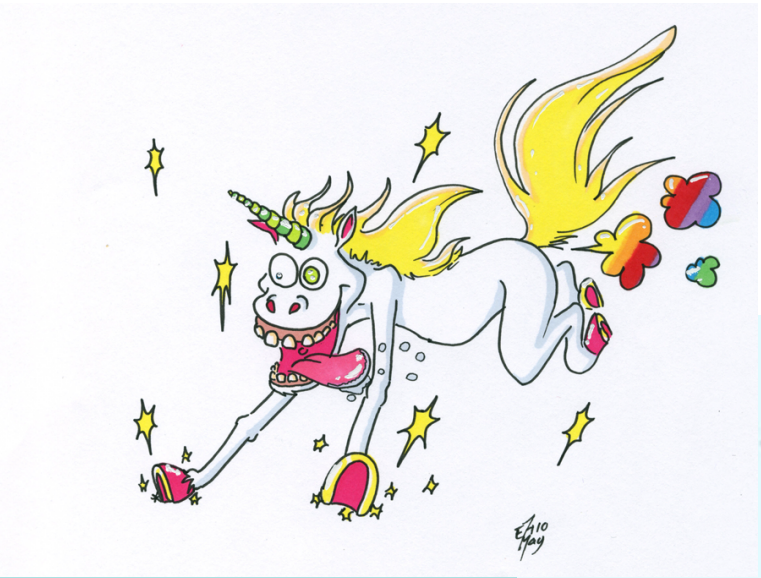
It's time to make the switch.

Babel is available wherever you use JavaScript - so start using it today!

How Developers Reacted

How Developers Reacted





What's on the ES6 menu?

What's on the ES6 menu?

fat arrow, classes, enhanced object literals, template strings, destructuring, default/rest/spread arguments, let, const, iterators via for..of, generators, comprehension, unicode, modules, loaders, map, set, weakmap, weakset, proxies, symbols, subclassable built-ins, promises, math/number/string/object APIs, binary and octal literals, reflect API, tail calls

...and what was possible already?

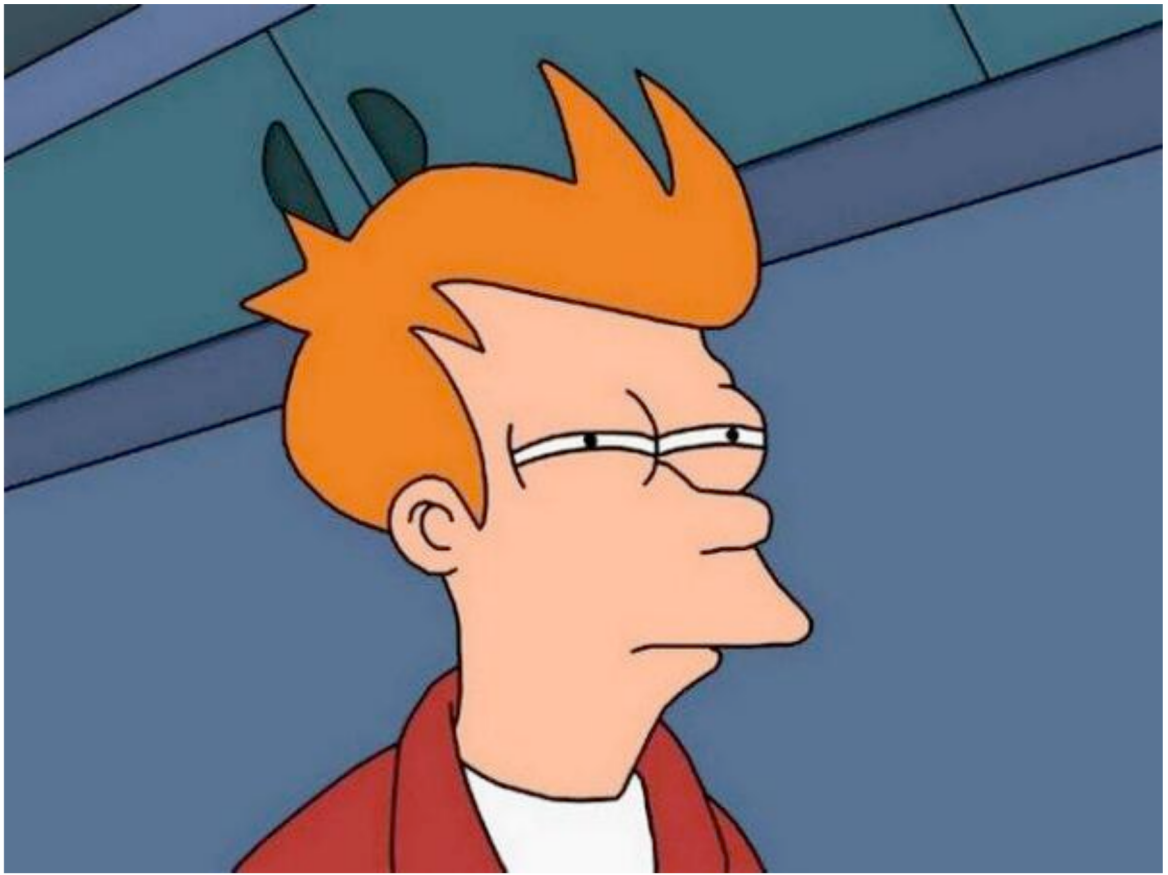
...and what was possible already?

take TypeScript, as example

...and what was possible already?

```
class Greeter {
  greeting:String;
  constructor(message:String) {
    this.greeting = message;
  }
  greet(Void):String {
    return "Hello, " + this.greeting;
  }
}
var greeter:Greeter = new Greeter("world");
```

...and what was possible already?



...and what was possible already?

```
class Greeter {
  var greeting:String;
  function Greeter(message:String) {
    this.greeting = message;
  }
  function greet(Void):String {
    return "Hello, " + this.greeting;
  }
}
var greeter:Greeter = new Greeter("world");
```

...and what was possible already?

yep, latter one is ActionScript 2 from 2003

“only” 9 years before via ES4

a matter of Point of View

a matter of Point of View

joe (3 days ago)

Heh, I remember I thought many ES6 features were oddly designed, or eve
said features (and had decided to target ES6), I dutifully added 'em to my c

Now I think *I* was crazy. When I was implementing the module import/exp
I can't imagine living without it; I love it. I thought the concept of Symbol w
"__" gets hard on the eyes after a while).

When I first started by project (all-shape.com), my code looked like this:

```
function Class() {
    Object.defineProperty(this, "prop", {
        get : function() {
        }
    })
}
Class.prototype.MethodA = function() {
}
Class.prototype.MethodB = function() {
}
```

a matter of Point of View

You have no idea how much it meant to me when I implemented classes and started seething this instead

```
class Class {
    MethodA() {
    }
    MethodB() {
    }
    get prop() {
    }
}
```

Believe it or not, I implemented super() late. Oh man. When I started seeing suff like:

```
Bleh.prototype.bleh.call(this)
```

Replaced with:

```
super.bleh();
```

I thought I would jump for joy.

```
}
```

Recap:

Recap:

How was he writing ?

```
function MyClass() {
    Object.defineProperty(this, "prop", {
        get : function() {
        }
    })
}
MyClass.prototype.MethodA = function() {
}
MyClass.prototype.MethodB = function() {
}
```

Recap:

How was he writing ?

```
function MyClass() {
    Object.defineProperty(this, "prop", {
        get : function() {
        }
    })
}
MyClass.prototype.MethodA = function() {
}
MyClass.prototype.MethodB = function() {
}
```

How could have been instead ?

```
var MyClass = Class({
  methodA: function () {},
  methodB: function () {},
  get prop() {}
});
```

Recap:

What was he expecting ?

```
class MyClass {
    MethodA() {
    }
    MethodB() {
    }
    get prop() {
    }
}
```

How could have been instead ?

```
var MyClass = Class({
  methodA: function () {},
  methodB: function () {},
  get prop() {}
});
```

Recap:

What was he expecting ?

```
class MyClass {
    MethodA() {
    }
    MethodB() {
    }
    get prop() {
    }
}
```

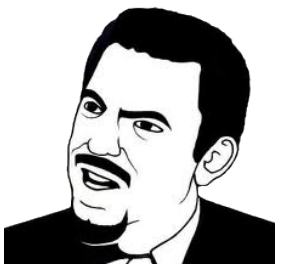
What could I write already ?

```
var MyClass = Class({
  methodA() {},
  methodB() {},
  get prop() {}
});
```

Recap:

What was he expecting ?

```
class MyClass {
    MethodA() {
    }
    MethodB() {
    }
    get prop() {
    }
}
```



What could I write already ?

```
var MyClass = Class({
  MethodA() {
  },
  MethodB() {
  },
  get prop() {
  }
});
```

Recap:

What was he expecting ?

```
class MyClass {
    MethodA() {
    }
    MethodB() {
    }
    get prop() {
    }
}
```

What could I write already ?

```
var MyClass = Class({
  methodA() {},
  methodB() {},
  get prop() {}
});
```

babel –whitelist=es6.properties.shorthand

Recap:

What was he expecting ?

```
class MyClass {
    MethodA() {
    }
    MethodB() {
    }
    get prop() {
    }
}
```

What could I write already ?

```
var MyClass = Class({
  methodA() {},
  methodB() {},
  get prop() {}
});
```

babel –whitelist=es6.properties.shorthand

https://github.com/WebReflection/es-class

ES6 Classes

./class demo

ES6 Classes – The Good

- after all these years we have a standard for classes
- getters/setters easily on the prototype
- getters/setters and methods are **not enumerable**
- easy to call super
- easy to subclass user classes and natives

ES6 Classes – The Bad

- classes are not features complete
- get/set & null objects can break on older browsers if transpiled
- properties as defaults can't be defined
- `super`, if present, can only be invoked before `this`
  ('cause `this` is undefined before `super`)
- native classes subclassing is basically nowhere

ES6 Classes – The Hype

- better options and with more features are ignored (ie es-class)

class

10247 results for ‘class’

class `deadlyicon`
A simple yet powerful Ruby-like Class inheritance system
★ 1 v0.1.4
class, Class, Constructor, prototype, inheritance, class inheritance

class-extender `zlatkofedor`
Simple class inheritance
★ 0 v1.0.11
class, extend, inheritance, amd

findhit-class `cuss`
javascript class framework
★ 0 v0.1.1
class, framework, extender, extend, nodejs

class-emit `zlatkofedor`
Simple class emit
★ 0 v1.0.12
class, emit, on, amd

backbone-class `damassi`
Backbone Class is the missing 'Backbone.Class' in Backbone.js which provides clean JavaScript class inheritence via the Backbone.extend pattern

10247 results for ‘class’

class `deadlyicon`
A simple yet powerful Ruby-like Class inheritance system
★ 1 v0.1.4
class, Class, Constructor, prototype, inheritance, class inheritance

class-extender `zlatkofedor`
Simple class inheritance
★ 0 v1.0.11
class, extend, inheritance, amd

findhit-class `cuss`
javascript class framework
★ 0 v0.1.1
class, framework, extender, extend, nodejs

class-emit `zlatkofedor`
Simple class emit
★ 0 v1.0.12
class, emit, on, amd

backbone-class `damassi`
Backbone Class is the missing 'Backbone.Class' in Backbone.js which provides clean JavaScript class inheritence via the Backbone.extend pattern

ES6 Classes – The Hype

- better options and with more features are ignored (ie es-class)
- getters/setters are suddenly OK to use
- transpiled inconsistency? who cares! (ie node 0.10+__proto__)
- composition sacrified due lack of traits
- ain't nobody gonna need native subclassing

ES6 String Template

ES6 String Template

How it is in ES6

```
`test1 ${1 + 2} test2 ${3 + 4}`


function greetings(name) {
  return `Hello ${name}!`;
}
console.log(greetings('Andrea'));


(function (name) {
return (
`Hello there,
  this is ${name}!`
)}('Andrea'));



html`<a href="${href}">hello</a>`;
```

ES6 String Template

How it is in ES6

```
`test1 ${1 + 2} test2 ${3 + 4}`
```

```
function greetings(name) {
  return `Hello ${name}!`;
}
console.log(greetings('Andrea'));
```

```
(function (name) {
return (
`Hello there,
  this is ${name}!`
)}('Andrea'));
```

```
html`<a href="${href}">hello</a>`;
```

How it could have been since about ever

```
'test1 ${1 + 2} test2 ${3 + 4}'.template();
```

ES6 String Template

How it is in ES6

```
`test1 ${1 + 2} test2 ${3 + 4}`
```

```
function greetings(name) {
  return `Hello ${name}!`;
}
console.log(greetings('Andrea'));
```

```
(function (name) {
return (
`Hello there,
  this is ${name}!`
)}('Andrea'));
```

```
html`<a href="${href}">hello</a>`;
```

How it could have been since about ever

```
'test1 ${1 + 2} test2 ${3 + 4}'.template();
```

```
function greetings(user) {
  return 'Hello ${name}!'.template(user);
}
console.log(greetings({name: 'Andrea'}));
```

ES6 String Template

How it is in ES6

```
`test1 ${1 + 2} test2 ${3 + 4}`
```

```
function greetings(name) {
  return `Hello ${name}!`;
}
console.log(greetings('Andrea'));
```

```
(function (name) {
return (
`Hello there,
  this is ${name}!`
)}('Andrea'));
```

```
html`<a href="${href}">hello</a>`;
```

How it could have been since about ever

```
'test1 ${1 + 2} test2 ${3 + 4}'.template();
```

```
function greetings(user) {
  return 'Hello ${name}!'.template(user);
}
console.log(greetings({name: 'Andrea'}));
```

```
(function () {/*
Hello there,
  this is ${name}!
*/}).template({
  name: 'Andrea'
});
```

ES6 String Template

How it is in ES6

```
`test1 ${1 + 2} test2 ${3 + 4}`
```

```
function greetings(name) {
  return `Hello ${name}!`;
}
console.log(greetings('Andrea'));
```

```
(function (name) {
return (
`Hello there,
  this is ${name}!`
)}('Andrea'));
```

```
html`<a href="${href}">hello</a>`;
```

How it could have been since about ever

```
'test1 ${1 + 2} test2 ${3 + 4}'.template();
```

```
function greetings(user) {
  return 'Hello ${name}!'.template(user);
}
console.log(greetings({name: 'Andrea'}));
```

```
(function () {/*
Hello there,
  this is ${name}!
*/}).template({
  name: 'Andrea'
});
```

```
'<a href="${href}">hello</a>'.template(
  html, {href: href}
);
```

ES6 String Template

./strings demo

https://gist.github.com/WebReflection/8f227532143e63649804

ES6 String Template – The Good

- finally a standard way to have multiline strings
- can simplify ugly and error prone long strings concatenation
- it's a fast and lightweight eval, open doors to new patterns
- excellent shortcut for tests and debugging or HTML segments

ES6 String Template – The Bad

- multiline will bring in unnecessary white spaces + no \r\n
- despite the name, actually not good for templating
- it's a fast and lightweight eval, open doors to new footguns
- not suitable by default for i18n strings. This will hit developers

ES6 String Template – The Hype

- using tools to merge templates was already possible
- composing from external micro templates is better than having everything within the JS file
- testability, isolation, i18n won't work if not evaluated within the template string once passed around
- suddenly using evaluation is OK

ES6 WeakMap

ES6 WeakMap

- It's a great way to relate objects without modofying them
- also a great way to relate DOM nodes to JS Objects without needing to use an expando property (usually considered a dirty approach that could also be problematic)

ES6 WeakMap

- It's a great way to relate objects without modofying them
- also a great way to relate DOM nodes to JS Objects without needing to use an expando property (usually considered a dirty approach that could also be problematic)

Considering a generic link used as counter

```
var a = document.body.appendChild(
  document.createElement('a')
);
a.textContent = 'click me';
a.href = '#mocked-href';
```

ES6 WeakMap

What can we do in ES6 today

```
var wm = new WeakMap;

a.addEventListener(
  'click', setDataObject
);

function setDataObject(e) {
  var el = e.currentTarget;
  e.preventDefault();
  if (!wm.has(el)) {
    wm.set(el, {counter: 0});
  }
  console.log(++wm.get(el).counter);
}
```

ES6 WeakMap

What can we do in ES6 today

```
var wm = new WeakMap;

a.addEventListener(
  'click', setDataObject
);

function setDataObject(e) {
  var el = e.currentTarget;
  e.preventDefault();
  if (!wm.has(el)) {
    wm.set(el, {counter: 0});
  }
  console.log(++wm.get(el).counter);
}
```

What we could have always done

```
a.addEventListener('click', {
  handleEvent: function (e) {
    e.preventDefault();
    console.log(++this.counter);
  },
  counter: 0
});
```

ES6 WeakMap

What can we do in ES6 today

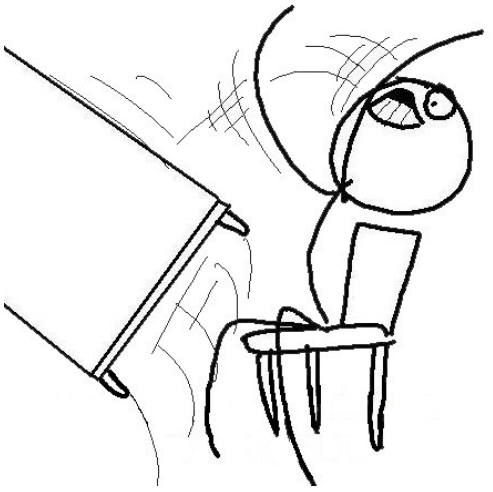
```
var wm = new WeakMap;

a.addEventListener(
  'click', setDataObject
);

function setDataObject(e) {
  var el = e.currentTarget;
  e.preventDefault();
  if (!wm.has(el)) {
    wm.set(el, {counter: 0});
  }
  console.log(++wm.get(el).counter);
}
```

What we could have always done

```
a.addEventListener('click', {
  handleEvent: function (e) {
    e.preventDefault();
    console.log(++this.counter);
  },
  counter: 0
});
```



ES6 WeakMap

What can we do in ES6 today

```
var wm = new WeakMap;

a.addEventListener(
  'click', setDataObject
);

function setDataObject(e) {
  var el = e.currentTarget;
  e.preventDefault();
  if (!wm.has(el)) {
    wm.set(el, {counter: 0});
  }
  console.log(++wm.get(el).counter);
}
```

W3C Recommendation

What we could have always done

IDL Definition

```
// Introduced in DOM Level 2:
interface EventListener {
  void              handleEvent(in Event evt);
};
```

Methods

handleEvent

This method is called whenever an event occurs of the

Parameters

evt of type Event

The Event contains contextual information abo the event's flow and default action.

No Return Value

No Exceptions

ES6 WeakMap

What can we do in ES6 today

```
var wm = new WeakMap;

a.addEventListener(
  'click', setDataObject
);

function setDataObject(e) {
  var el = e.currentTarget;
  e.preventDefault();
  if (!wm.has(el)) {
    wm.set(el, {counter: 0});
  }
  console.log(++wm.get(el).counter);
}
```

What we could have always done

```
a.addEventListener('click', {
  handleEvent: function (e) {
    e.preventDefault();
    console.log(++this.counter);
  },
  counter: 0
});
```

ES6 WeakMap

What can we do in ES6 today

```
var wm = new WeakMap;

a.addEventListener(
  'click', setDataObject
);

function setDataObject(e) {
  var el = e.currentTarget;
  e.preventDefault();
  if (!wm.has(el)) {
    wm.set(el, {counter: 0});
  }
  console.log(++wm.get(el).counter);
}
```

What we could have always done

```
a.addEventListener(
  'click',
  new Counter
);
```

ES6 WeakMap

What can we do in ES6 today

```
var wm = new WeakMap;

a.addEventListener(
  'click', setDataObject
);

function setDataObject(e) {
  var el = e.currentTarget;
  e.preventDefault();
  if (!wm.has(el)) {
    wm.set(el, {counter: 0});
  }
  console.log(++wm.get(el).counter);
}
```

What we could have always done

```
a.addEventListener(
  'click',
  new Counter
);


var Counter = Class({
  handleEvent: function (e) {
    e.preventDefault();
    console.log(++this.i);
  },
  i: 0
});
```

ES6 WeakMap

./weakmap demo

ES6 WeakMap – The Good

- a powerful way to relate objects in an unobtrusive way
- a possible solution to obtain private properties behavior
- a good pattern to help avoid leaks

ES6 WeakMap – The Bad

- apparently very heavy in terms of GC pressure
- impossible to polyfill without being obtrusive or without leaking
- if abused can suffer both leaks and GC pressure

ES6 WeakMap – The Hype

- already possible in the standard DOM world but ignored
- instantly abused within constructors and private scopes
- unaware of how weak polyfills work
- unaware of failures and surprises behind polyfills

Symbols in a nutshell

Symbols in a nutshell

- **Good**: the way to have unique Ids without abusing `Math.random()` and `new Date` + the way to mean protected properties and methods

Symbols in a nutshell

- **Good**: the way to have unique Ids without abusing `Math.random()` and `new Date` + the way to mean protected properties and methods
- **Bad**: do not scale with immutable variables + all polyfills are full of caveats

Symbols in a nutshell

- **Good**: the way to have unique Ids without abusing `Math.random()` and `new Date` + the way to mean protected properties and methods
- **Bad**: do not scale with immutable variables + all polyfills are full of caveats
- **Hype**: adoption without fully understanding, i.e. `Object.assign`, `Object.getOwnPropertySymbols` and `Reflect.ownKeys`

Symbols in a nutshell

- **Good**: the way to have unique Ids without abusing `Math.random()` and `new Date` + the way to mean protected properties and methods
- **Bad**: do not scale with immutable variables + all polyfills are full of caveats
- **Hype**: adoption without fully understanding, i.e. `Object.assign`, `Object.getOwnPropertySymbols` and `Reflect.ownKeys`

https://github.com/WebReflection/get-own-property-symbols

Destructuring

Destructuring Arrays

ES6

```
var [
  ,
  firstName,
  lastName
] = "John Doe".match(
  /^(\w+) (\w+)$/
);
```

Destructuring Arrays

ES6

```
var [
  ,
  firstName,
  lastName
] = "John Doe".match(
  /^(\w+) (\w+)$/
);
```

"This makes it really neat for example to pull values out of regular expression matches:"

https://strongloop.com/strongblog/getting-started-with-javascript-es6-destructuring/

Destructuring Arrays

ES6

```
var [
  ,
  firstName,
  lastName
] = "John Doe".match(
  /^(\w+) (\w+)$/
);
```

ES3

```
if (/^(\w+) (\w+)$/.test("John Doe"))
with ({
  firstName: RegExp.$1,
  lastName: RegExp.$2
}) {
  firstName, lastName;
}
```

"This makes it really neat for example to pull values out of regular expression matches:"

https://strongloop.com/strongblog/getting-started-with-javascript-es6-destructuring/

Destructuring Objects

ES6

```
var person = {
  firstName: "John",
  lastName: "Doe"
};
var {firstName, lastName} = person;
```

Destructuring Objects

ES6

```
var person = {
  firstName: "John",
  lastName: "Doe"
};
var {firstName, lastName} = person;
```

ES3

```
var person = {
  firstName: "John",
  lastName: "Doe"
};
with(person){ firstName, lastName }
```

Destructuring Objects

ES6

```
var person = {
  dateOfBirth: [1, 1, 1980]
};
var {
  dateOfBirth: [day, month, year]
} = person;
```

Destructuring Objects

ES6

```
var person = {
  dateOfBirth: [1, 1, 1980]
};
var {
  dateOfBirth: [day, month, year]
} = person;
```

which format?
DD/MM/YYYY or MM/DD/YYYY ?

Destructuring Objects

ES6

```
var person = {
  dateOfBirth: [1, 1, 1980]
};
var {
  dateOfBirth: [day, month, year]
} = person;
```

which format?
DD/MM/YYYY or MM/DD/YYYY ?

ES3

no need to mess up like that

Destructuring – The Good

- it simplifies named variables declaration ([x,y,z] = pointArray)
- it makes syntax more compact
- ...rest (and ...spread) ability is amazing !

Destructuring – The Bad

- it might mislead assignment without defaults-ability
- it brings in less predictable variables/properties greppability

Destructuring – The Hype

- it has been similarily around for ages unnoticed

ES6 – The Good

ES6 – The Good

- tons of new features developers have been asking for
- tooling works already and is the most advanced we've ever had
- there are new possibilities capable to unlock new JS potentials
- new comers will probably find themselves more comfortable

ES6 – The Bad

- few features are incomplete, might be a blocker for many
- tooling might not fix problems in every target engine
- debugging after tooling AND minification is a mess
- new comers should learn “older” JS anyway to better deal with it

ES6 – The Hype

- developers crave for the latest even if unnecessary or slower
- incomplete, not fully cross-platform compatibility is suddenly OK
- “*works in Chrome? Good to go!*” is unethical and unproductive

ES6 – A Better Approch

- polyfills and transpilers might not work in your targets: have you test them ****after**** transpiling and minifying your code?

ES6 – A Better Approch

- polyfills and transpilers might not work in your targets: have you test them ****after**** transpiling and minifying your code?
- older browsers/engines become slower and slower because of wrapped, transpile, polyfilled, polluted environment: are you targeting these too?

ES6 – A Better Approch

- polyfills and transpilers might not work in your targets: have you test them ****after**** transpiling and minifying your code?
- older browsers/engines become slower and slower because of wrapped, transpile, polyfilled, polluted environment: are you targeting these too?
- learn JavaScript, like “*All The Things*” !!! After that, feel free to choose your patterns accordingly. Don't stop with a superset or just classes, don't stop with a framework: there's more!

ES6 – A Better Approch

- polyfills and transpilers might not work in your targets: have you test them ****after**** transpiling and minifying your code?
- older browsers/engines become slower and slower because of wrapped, transpile, polyfilled, polluted environment: are you targeting these too?
- learn JavaScript, like "*All The Things*" !!! After that, feel free to choose your patterns accordingly. Don't stop with a superset or just classes, don't stop with a framework: there's more!

- Finally, **feel free to go ES6 when appropriate**!

ES6 – Last Quick Note

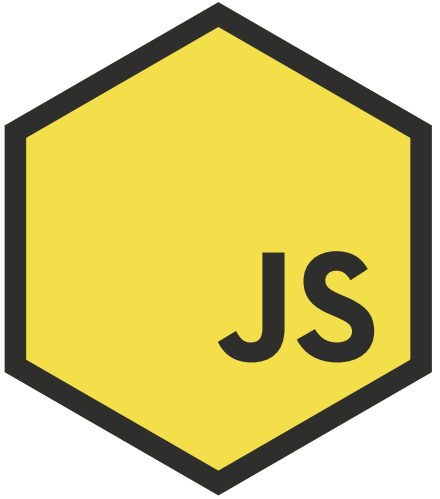
if you are writing “Universal JS”, please transpile before publishing to npm (or others)

https://medium.com/@mjackson/universal-javascript-4761051b7ae9

ES6 – Last Quick Note

if you are writing “Universal JS”, please transpile before publishing to npm (or others)

https://medium.com/@mjackson/universal-javascript-4761051b7ae9



ES6 Now: The Good, The Bad, and The Hype

Thank You!

by Andrea Giammarchi
@WebReflection