ECMAScript 6

The Future of JavaScript is Now!

Tom Van Cutsem

JOIN ‘15

@tvcutsem

My involvement in JavaScript

- PhD on programming language technology

- 2010: Visiting Faculty at Google, Caja team

- Joined ECMA TC39
- Actively contributed to ECMAScript 6 spec

Talk Outline

- Part I: JavaScript's origins, and the long road to ECMAScript 6
- Part II: a brief tour of ECMAScript 6
- Part III: using ECMAScript 6 today
- Wrap-up

Part I
JavaScript's origins, and the road to ECMAScript 6

JavaScript's origins

- Invented by Brendan Eich in 1995, then an intern at Netscape, to support client-side scripting in Netscape navigator
- First called *LiveScript*, then *JavaScript*, then standardized as *ECMAScript*
- Microsoft "copied" JavaScript in IE JScript, "warts and all"

Brendan Eich,
Inventor of JavaScript

The world's most misunderstood language

Douglas Crockford,
Inventor of JSON

See also: "JavaScript: The World's Most Misunderstood Programming Language" by Doug Crockford at http://www.crockford.com/javascript/javascript.html

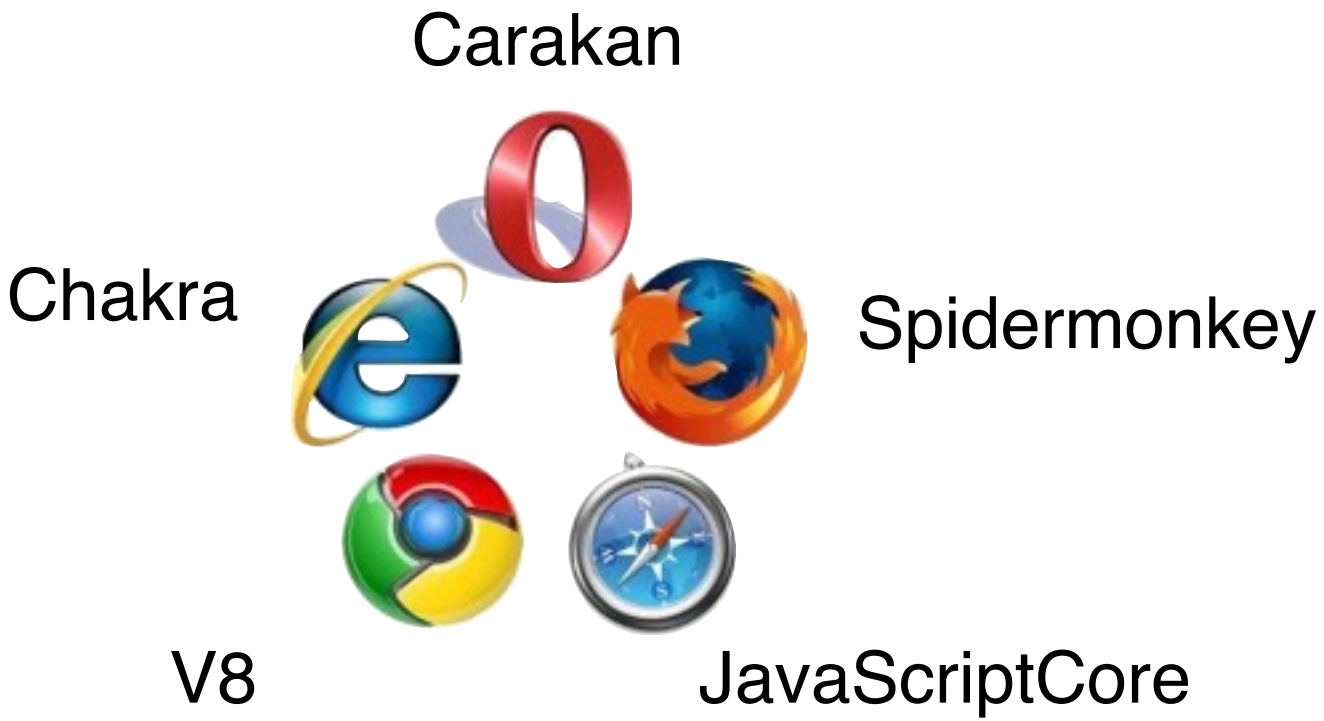
The Good Parts

- Functions as first-class objects
- Dynamic objects with prototype-based inheritance
- Object literals
- Array literals

The Bad Parts

- Global variables (no modules)
- `with` statement
- Implicit type coercion
- Var hoisting (no block scope)
- …

ECMAScript: “Standard” JavaScript



ECMAScript: “Standard” JavaScript

~~Carakan~~ V8

Chakra

Spidermonkey

V8

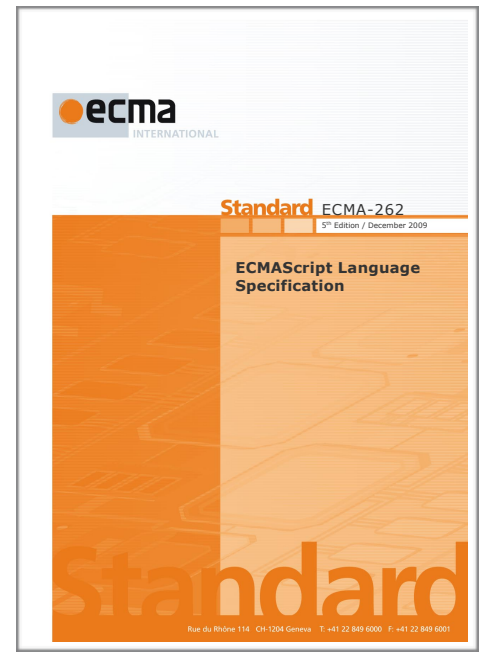
JavaScriptCore

(170.000+ npm packages!)

TC39: the JavaScript “standardisation committee”

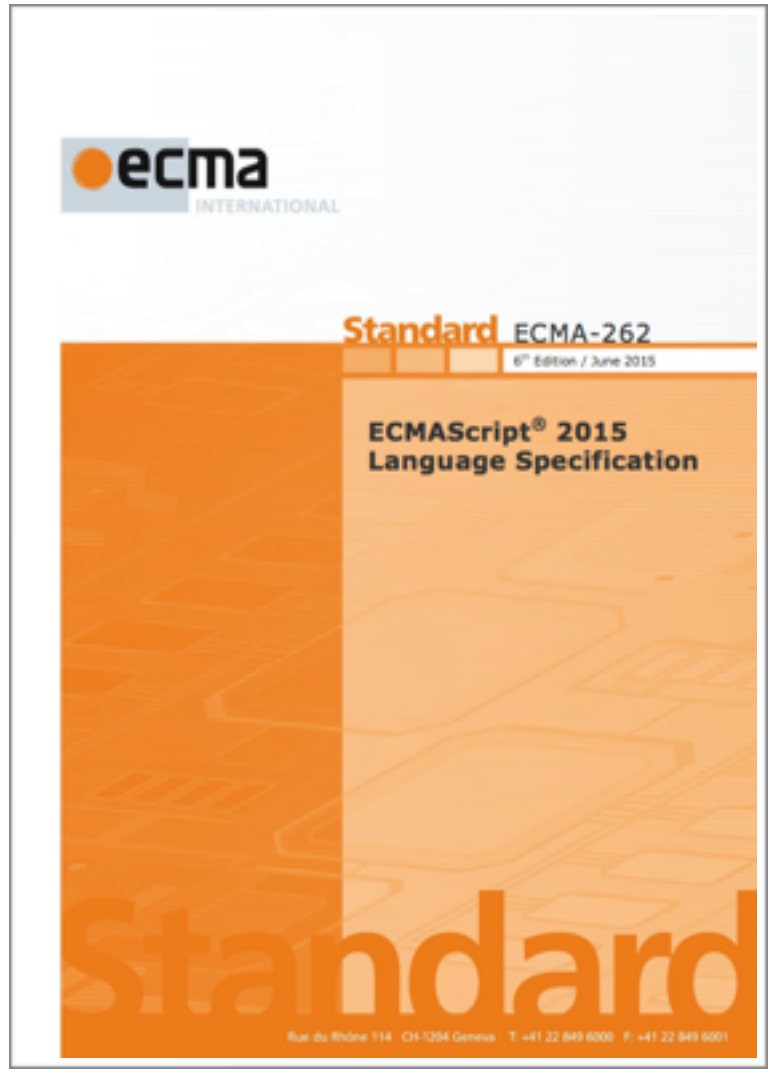
- Representatives from major Internet companies, browser vendors, web organisations, popular JS libraries and academia
- Maintains the ECMA-262 specification.
- The spec is a handbook mainly intended for language implementors. Extremely detailed to minimize incompatibilities.

Allen Wirfs-Brock,
ECMA-262 technical editor



ECMAScript specification: history

- 1st ed. 1997
- 2nd ed. 1998
- 3rd ed. 1999
- ~~4th ed.~~
- 5th ed. 2009
  (introduced “strict mode”)
- 6th ed. June 2015
  (aka ECMAScript 2015)



TC39

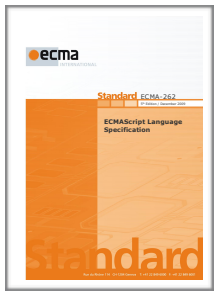
- Meets bi-monthly, mostly in the SF bay area. **Meetings** are technical, not political in nature
- **Discussions** held in the open on es-discuss@mozilla.org
- Committee very much aware of the dangers of “design-by-committee”.
  - **Champion** model to combat this (each feature led by handful of experts)
- Important **decisions** made by global consensus

Part II
A brief tour of ECMAScript 6

ECMAScript 6

- Major update: many new features (too many to list here)
- Point-in-case:

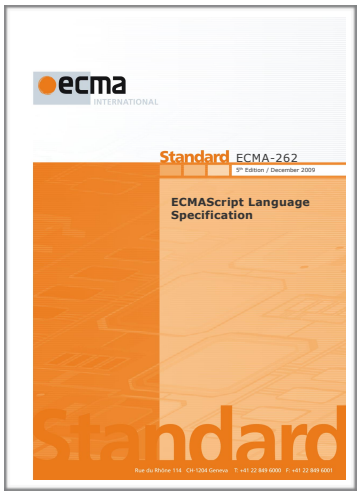
ES5.1



258-page pdf

x 2.2

ES6



566-page pdf

ECMAScript 6: shortlist

- Arrow functions
- Classes
- Control flow Goodness:
  - Iterators
  - Generators
  - Promises
  - async/await [tentative, ECMAScript 7 sneak peak]

ECMAScript 6: shortlist

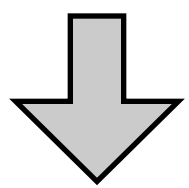
- **Arrow functions**
- Classes
- Control flow Goodness:
  - Iterators
  - Generators
  - Promises
  - async/await [tentative, ECMAScript 7 sneak peak]

ECMAScript 6: arrow functions

- Shorter, and also automatically captures current value of `this`
  No more `var that = this;`

ES5

```
function sum(array) {
  return array.reduce(
    function(x, y) { return x + y; }, 0);
}
```



ES6

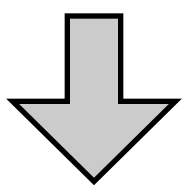
```
function sum(array) {
  return array.reduce((x, y) => x + y, 0);
}
```

ECMAScript 6: arrow functions

- Shorter, and also automatically captures current value of `this`
  No more `var that = this;`

ES5

```
function sum(array) {
  return array.reduce(
    function(x, y) { return x + y; }, 0);
}
```



ES6

```
function sum(array) {
  return array.reduce((x, y) => x + y, 0);
}
```

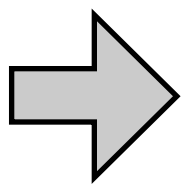
ECMAScript 6: shortlist

- Arrow functions
- **Classes**
- Control flow Goodness:
  - Iterators
  - Generators
  - Promises
  - async/await [tentative, ECMAScript 7 sneak peak]

ECMAScript 6: classes

- All code inside a class is implicitly opted into strict mode!

```
function Point(x, y) {
  this.x = x;
  this.y = y;
}

Point.prototype = {
  toString: function() {
    return "[Point...]";
  }
}

var p = new Point(1,2);
p.x;
p.toString();
```

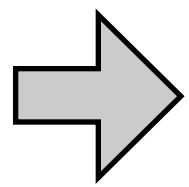


```
class Point {
  constructor(x, y) {
    this.x = x;
    this.y = y;
  }

  toString() {
    return "[Point...]";
  }
}

var p = new Point(1,2);
p.x;
p.toString();
```

ECMAScript 6: classes

- All code inside a class is implicitly opted into strict mode!

```
function Point(x, y) {
  this.x = x;
  this.y = y;
}

Point.prototype = {
  toString: function() {
    return "[Point...]";
  }
}

var p = new Point(1,2);
p.x;
p.toString();
```



```
class Point {
  constructor(x, y) {
    this.x = x;
    this.y = y;
  }

  toString() {
    return "[Point...]";
  }
}

var p = new Point(1,2);
p.x;
p.toString();
```

ECMAScript 6: classes

- Single-inheritance, super-calls, static members

```
class Point3D extends Point {
  constructor(x, y, z) {
    super(x,y);
    this.z = z;
  }

  static getOrigin() {
    return new Point3D(0,0,0);
  }
}
```

ECMAScript 6: shortlist

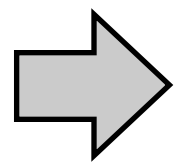
- Arrow functions
- Classes
- **Control flow Goodness:**
  - **Iterators**
  - Generators
  - Promises
  - async/await [tentative, ECMAScript 7 sneak peak]

ECMAScript 6 Iterators

```
function fibonacci() {
  var pre = 0, cur = 1;
  return {
    next: function() {
      var temp = pre;
      pre = cur;
      cur = cur + temp;
      return { done: false, value: cur }
    }
  }
}
```

ES5

```
var iter = fibonacci();
var nxt = iter.next();
while (!nxt.done) {
  var n = nxt.value;
  if (n > 100)
    break;
  print(n);
  nxt = iter.next();
}
```



ES6

```
for (var n of fibonacci()) {
  if (n > 100)
    break;
  print(n);
}
```

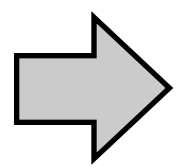
// generates 1, 1, 2, 3, 5, 8, 13, 21, …

ECMAScript 6 Iterators

```
function fibonacci() {
  var pre = 0, cur = 1;
  return {
    next: function() {
      var temp = pre;
      pre = cur;
      cur = cur + temp;
      return { done: false, value: cur }
    }
  }
}
```

ES5

```
var iter = fibonacci();
var nxt = iter.next();
while (!nxt.done) {
  var n = nxt.value;
  if (n > 100)
    break;
  print(n);
  nxt = iter.next();
}
```



ES6

```
for (var n of fibonacci()) {
  if (n > 100)
    break;
  print(n);
}
```

// generates 1, 1, 2, 3, 5, 8, 13, 21, …

ECMAScript 6: shortlist

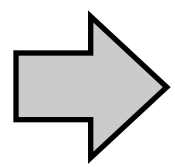
- Arrow functions
- Classes
- **Control flow Goodness:**
  - Iterators
  - **Generators**
  - Promises
  - async/await [tentative, ECMAScript 7 sneak peak]

ECMAScript 6 Generators

- A generator function implicitly creates and returns an iterator

ES5

```
function fibonacci() {
  var pre = 0, cur = 1;
  return {
    next: function() {
      var tmp = pre;
      pre = cur;
      cur = cur + tmp;
      return { done: false, value: cur }
    }
  }
}
```



ES6

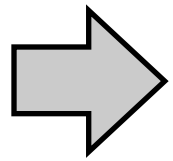
```
function* fibonacci() {
  var pre = 0, cur = 1;
  for (;;) {
    var tmp = pre;
    pre = cur;
    cur = cur + tmp;
    yield cur;
  }
}
```

ECMAScript 6 Generators

- A generator function implicitly creates and returns an iterator

ES5

```
function fibonacci() {
  var pre = 0, cur = 1;
  return {
    next: function() {
      var tmp = pre;
      pre = cur;
      cur = cur + tmp;
      return { done: false, value: cur }
    }
  }
}
```



ES6

```
function* fibonacci() {
  var pre = 0, cur = 1;
  for (;;) {
    var tmp = pre;
    pre = cur;
    cur = cur + tmp;
    yield cur;
  }
}
```

ECMAScript 6: shortlist

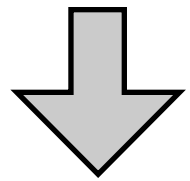
- Arrow functions
- Classes
- **Control flow Goodness:**
  - Iterators
  - Generators
  - **Promises**
  - async/await [tentative, ECMAScript 7 sneak peak]

ECMAScript 6 Promises

- A promise is a placeholder for a value that may only be available in the future

ES5

```
readFile("hello.txt", function (err, content) {
  if (err) {
    // handle error
  } else {
    // use content
  }
})
```



ES6

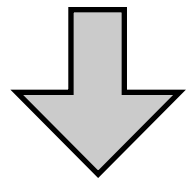
```
var pContent = readFile("hello.txt");
pContent.then(function (content) {
  // use content
}, function (err) {
  // handle error
});
```

ECMAScript 6 Promises

- A promise is a placeholder for a value that may only be available in the future

ES5

```
readFile("hello.txt", function (err, content) {
  if (err) {
    // handle error
  } else {
    // use content
  }
})
```



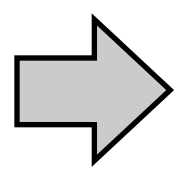
ES6

```
var pContent = readFile("hello.txt");
var p2 = pContent.then(function (content) {
  // use content
}, function (err) {
  // handle error
});
```

ECMAScript 6 Promises

- Promises can be *chained* to avoid callback hell

```
// step2(value, callback) -> void

step1(function (value1) {
    step2(value1, function(value2) {
        step3(value2, function(value3) {
            step4(value3, function(value4) {
                // do something with value4
            });
        });
    });
});
```



```
// promisedStep2(value) -> promise

Q.fcall(promisedStep1)
.then(promisedStep2)
.then(promisedStep3)
.then(promisedStep4)
.then(function (value4) {
    // do something with value4
})
.catch(function (error) {
    // handle any error here
})
.done();
```

Example adapted from https://github.com/kriskowal/q

ECMAScript 6 Promises

- Promises already exist as a library in ES5
- Personal favorite on npm: q (https://github.com/kriskowal/q)
- Then why standardize?
  - Wide disagreement on a single Promise API. ES6 settled on an API called “Promises/A+”. See promisesaplus.com
  - Standard API allows platform APIs to use Promises as well
  - W3C’s latest DOM APIs already use promises

ECMAScript 6: shortlist

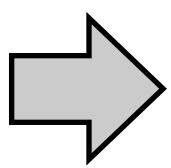
- Arrow functions
- Classes
- **Control flow Goodness:**
  - Iterators
  - Generators
  - Promises
  - **async/await [tentative, ECMAScript 7 sneak peak]**

ECMAScript **7**: async/await

- async/await is a C# 5.0 feature that enables asynchronous programming using “direct style” control flow (i.e. no callbacks)

ES6

```
// promisedStep2(value) -> promise

Q.fcall(promisedStep1)
.then(promisedStep2)
.then(promisedStep3)
.then(promisedStep4)
.then(function (value4) {
    // do something with value4
})
.catch(function (error) {
    // handle any error here
})
.done();
```



ES7

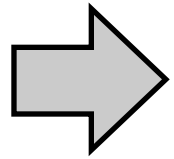
```
// step2(value) -> promise

(async function() {
  try {
    var value1 = await step1();
    var value2 = await step2(value1);
    var value3 = await step3(value2);
    var value4 = await step4(value3);
    // do something with value4
  } catch (error) {
    // handle any error here
  }
}())
```

async/await in ECMAScript **6**

- Generators can be used as async functions, with some tinkering
- co npm library in iojs (no flags) or node (>= 0.11.x, `--harmony` flag)

ES7

```
(async function() {
  try {
    var value1 = await step1();
    var value2 = await step2(value1);
    var value3 = await step3(value2);
    var value4 = await step4(value3);
    // do something with value4
  } catch (error) {
    // handle any error here
  }
}())
```



ES6

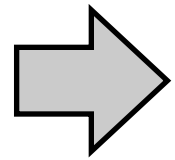
```
co(function*() {
  try {
    var value1 = yield step1();
    var value2 = yield step2(value1);
    var value3 = yield step3(value2);
    var value4 = yield step4(value3);
    // do something with value4
  } catch (error) {
    // handle any error here
  }
})
```

async/await in ECMAScript **6**

- Generators can be used as async functions, with some tinkering
- co npm library in iojs (no flags) or node (>= 0.11.x, `--harmony` flag)

ES7

```
(async function() {
  try {
    var value1 = await step1();
    var value2 = await step2(value1);
    var value3 = await step3(value2);
    var value4 = await step4(value3);
    // do something with value4
  } catch (error) {
    // handle any error here
  }
}())
```



ES6

```
co(function*() {
  try {
    var value1 = yield step1();
    var value2 = yield step2(value1);
    var value3 = yield step3(value2);
    var value4 = yield step4(value3);
    // do something with value4
  } catch (error) {
    // handle any error here
  }
})
```

ECMAScript 6: shortlist

- Arrow functions
- Classes
- Control flow Goodness:
  - Iterators
  - Generators
  - Promises
  - async/await [tentative, ECMAScript 7 sneak peak]

ECMAScript 6: longlist

- Luke Hoban has an excellent overview of all ES6 features at

  git.io/es6features

Luke Hoban,
Microsoft representative on TC39

- arrows
- classes
- enhanced object literals
- template strings
- destructuring
- default + rest + spread
- let + const
- iterators + for..of
- generators
- unicode
- modules
- module loaders
- map + set + weakmap + weakset
- proxies
- symbols
- subclassable built-ins
- promises
- math + number + string + array + object APIs
- binary and octal literals
- reflect api
- tail calls

Part III
Using ECMAScript 6 today

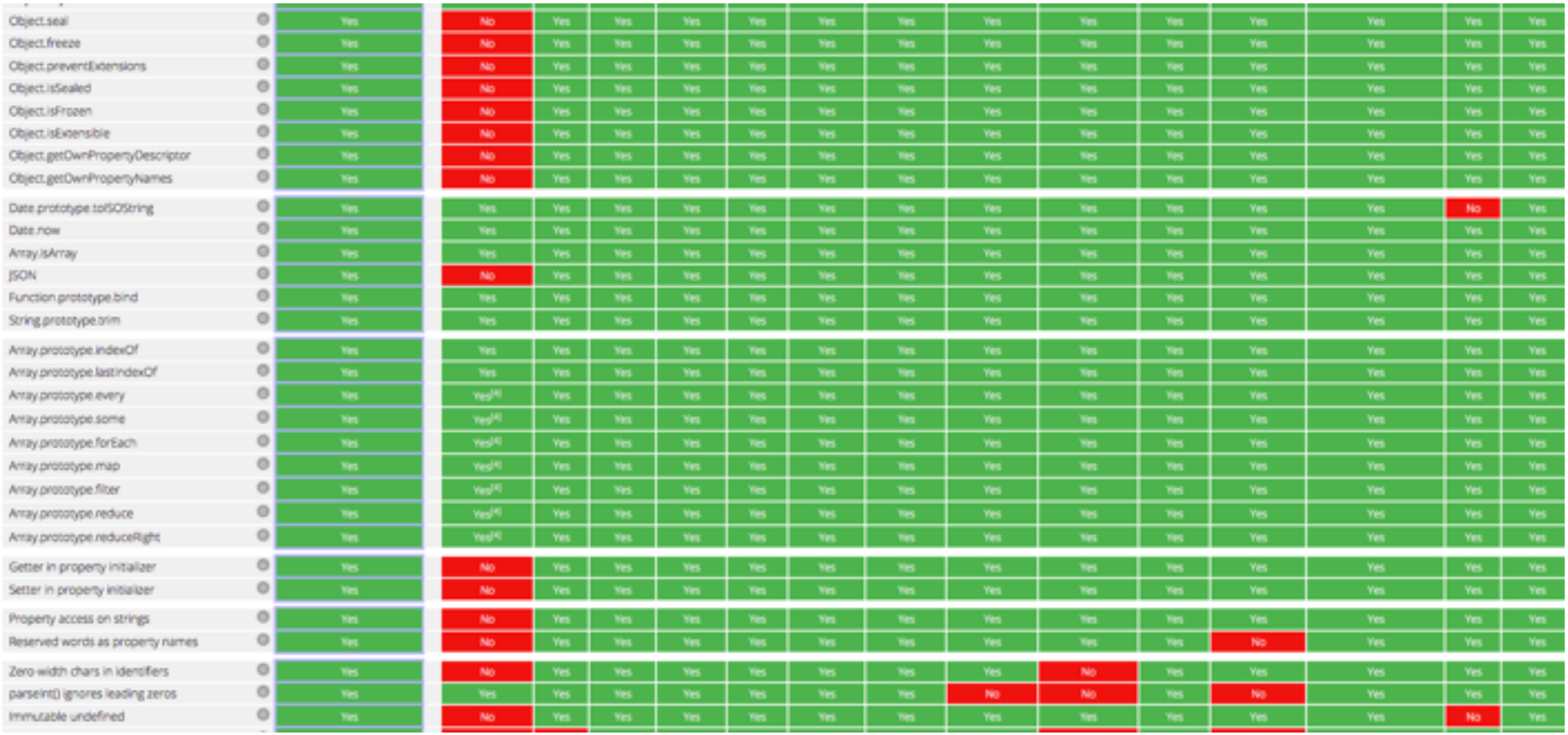
ECMAScript 6: timeline

- ECMAScript 6 spec was officially ratified in June 2015
- However: browsers do not support full ES6 overnight
- Parts of ES6 already supported on some browsers today

ECMAScript 6 support (july 2015)

See Juriy Zaytsev's (a.k.a. kangax) excellent compatibility tables http://kangax.github.io/es5-compat-table/es6/ for current status

ECMAScript 5 support (july 2015)



Object.seal	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Object.freeze	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Object.preventExtensions	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Object.isSealed	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Object.isFrozen	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Object.isExtensible	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Object.getOwnPropertyDescriptor	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Object.getOwnPropertyNames	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Date.prototype.toISOString	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
Date.now	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Array.isArray	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
JSON	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Function.prototype.bind	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
String.prototype.trim	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Array.prototype.indexOf	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Array.prototype.lastIndexOf	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Array.prototype.every	Yes	Yes[4]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Array.prototype.some	Yes	Yes[4]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Array.prototype.forEach	Yes	Yes[4]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Array.prototype.map	Yes	Yes[4]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Array.prototype.filter	Yes	Yes[4]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Array.prototype.reduce	Yes	Yes[4]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Array.prototype.reduceRight	Yes	Yes[4]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Getter in property initializer	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Setter in property initializer	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Property access on strings	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Reserved words as property names	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
Zero-width chars in identifiers	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
parseInt() ignores leading zeros	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	No	Yes	Yes	Yes	Yes
Immutable undefined	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes

ECMAScript 6 compilers

- Compile ECMAScript 6 to ECMAScript 5
- **Babel**: focus on producing readable (as-if hand-written) ES5 code. Supports JSX as well.

- Microsoft **TypeScript**: technically not ES6 but roughly a superset of ES6. Bonus: type inference and optional static typing.

async/await in ECMAScript **5** (!)

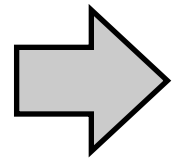
- BabelJS async functions plug-in based on Facebook Regenerator
- Works in browsers too!

facebook.github.io/regenerator

github.com/lukehoban/ecmascript-asyncawait

ES7

```
(async function() {
  try {
    var value1 = await step1();
    var value2 = await step2(value1);
    var value3 = await step3(value2);
    var value4 = await step4(value3);
    // do something with value4
  } catch (error) {
    // handle any error here
  }
}())
```



ES5

```
(function callee$0$0() {
  var value1, value2, value3, value4;
  return regeneratorRuntime.async(function callee$0$0$(context$1$0) {
    while (1) switch (context$1$0.prev = context$1$0.next) {
      case 0:
        context$1$0.prev = 0;
        context$1$0.next = 3;
        return regeneratorRuntime.awrap(step1());

      case 3:
        value1 = context$1$0.sent;
        context$1$0.next = 6;
        return regeneratorRuntime.awrap(step2(value1));

      case 6:
        value2 = context$1$0.sent;
        context$1$0.next = 9;
        return regeneratorRuntime.awrap(step3(value2));

      case 9:
        value3 = context$1$0.sent;
        context$1$0.next = 12;
        return regeneratorRuntime.awrap(step4(value3));

      case 12:
        value4 = context$1$0.sent;
        context$1$0.next = 17;
        break;

      case 15:
        context$1$0.prev = 15;
        context$1$0.t0 = context$1$0["catch"](0);

      case 17:
      case "end":
        return context$1$0.stop();
    }
  }, null, this, [[0, 15]]);
})();
```

ECMAScript 6 server-side support

- node --harmony

- **iojs**: node.js fork that more closely tracks latest developments in V8
  - supports major features out-of-the-box, such as classes, collections, generators, promises
  - the future of node: developed by core node contributors, community-driven

ECMAScript & the JVM ecosystem

- **Nashorn** on JDK8. ES6 is on their roadmap.
- **RingoJS**: what node.js is to v8, RingoJS aims to be to Rhino. Developer server-side JS apps on the JVM (Jetty-based I/O)
- **Rhino** development itself appears largely abandoned (Rhino 1.7 has partial ES5 support, no ES6 roadmap)

Image courtesy of dzone.com

Wrap-up

Take-home messages

ES6 is a *major* upgrade

Expect engines to upgrade gradually

Take-home messages

ES6 is a *major* upgrade

Expect engines to upgrade gradually

BABEL

TypeScript

Use ES6 to ES5 compilers to bridge the gap

ES6

You can start using ES6 today!

Thanks for listening!

ECMAScript 6

The Future of JavaScript is Now!

Tom Van Cutsem

JOIN ‘15

@tvcutsem