ES6 is Nigh

A PRESENTATION ON THE FUTURE OF JAVASCRIPT

http://es6isnigh.com

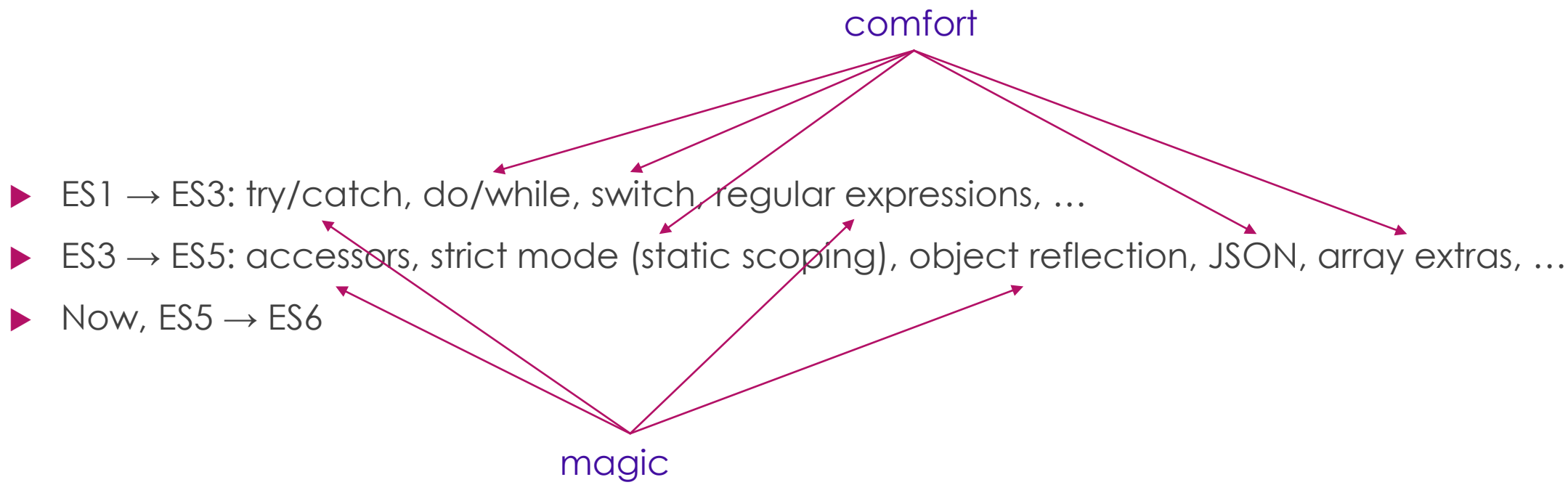
@domenic


PLEASE
IMPLEMENT
trans: prototype.
(ES6 is nish!)

Domenic Denicola

- https://github.com/domenic
- https://npmjs.org/~domenic
- @esdiscuss

History



- ES1 → ES3: try/catch, do/while, switch, regular expressions, …
- ES3 → ES5: accessors, strict mode (static scoping), object reflection, JSON, array extras, …
- Now, ES5 → ES6

Why?

"Stagnation on the web is a social ill."

—Brendan Eich

http://news.ycombinator.com/item?id=4634735

Why?

Also: say what you mean!

One JavaScript

- All ES5 code is ES6 code and has the same meaning
- No modes
- No backwards incompatibility
- No “fixes”
- ES6 is *purely additive*


@domenic


Parameter defaults
Arrow functions
Classes
Modules
@domenic


@domenic


Better object literals
Spread
Const
Sets and maps
Rest
Block scoping
Destructuring
Iteration
@domenic


@domenic


MAGIC
The Gathering®
Proper tail calls
Template strings
Binary data
Unicode
Symbols
Weak sets and maps
Generators
Proxies
@domenic

Comfort

- Better object literals
- Rest and spread
- Block scoping
- Const
- Destructuring
- Iteration

Better Object Literals

```
var empireJS = {
  attendees: "many",
  preParty() {
    console.log(this.attendees + " attendees are partying!");
  }
};

var conferences = { empireJS, cascadiaJS };
```

Rest and Spread

```
function sprintf(format, ...params) {
  assert(Array.isArray(params));
}

Math.max(...array);

Math.max(0, ...array, 5, ...array2);

new Date(...dateFields);

array.push(...array2);
```

More Spread

```
var prequels = ["tpm", "aotc", "rots"];
var movies = [...prequels, "anh", "tesb", "rotj"];

var nodeList = document.querySelectorAll("div");
[...nodeList].forEach(function (node) {
  // finally!
});
```

Destructuring

```
[a, b] = [b, a];

var [x, y] = f();

var re = /([a-z]+)(\s)(.*)([^a-z])/;
var [, hello, space, world, bang] = re.exec("hello world!");

var [first, ...allButFirst] = document.querySelectorAll("p");
```

Destructuring

```
var { tagName, textContent } = el;
var { viewModel, bindings } = doDataBinding();

var { firstElementChild: first, lastElementChild: last } = el;

var { firstElementChild: { tagName: firstTag } } = document.body;
var { children: [first, ...others] } = document.body;
```

Destructuring

```
function runTests({ reporter, ui }, [firstFile, ...others]) {
  // ...
}

try {
  runTests({ reporter: "spec", ui: "tdd" }, ["test1", "test2"]);
} catch ({ message }) {
  console.error(message);
}
```

Block Scoping

```
let log = console.log.bind(console);

function f(x) {
  if (Math.random() > 0.5) {
    let log = Math.log;
    x = log(x);
  }

  log("result: " + x);
}
```

Block Scoping

```
let log = console.log.bind(console);

function f(x) { // 5 refactorings later
  if (Math.random() > 0.5) {
    let log;
    x = log(x); // error! used a `let` before initialization.
    log = Math.log;
  }

  log("result: " + x);
}

f(Math.E); // but, the error doesn't get triggered until here: runtime, not static.
```

Block Scoping

```
function f(x) { // 10 refactorings later
  let log = console.log;
  let log = Math.log; // `SyntaxError`! No double `let`s.

  if (Math.random() > 0.5) {
    x = log(x);
  }

  log("result: " + x);
}
```

Block Scoping

```
for (let i = 0; i < a.length; ++i) {
  els[i].onclick = function () {
    return a[i];
  };
}

assert(typeof i === "undefined");
assert(els[0].onclick() === a[0]);
```

Block Scoping

```
if (Math.random() > 0.5) {
  function go() {
    console.log("gone!");
  }

  el.onmousedown = go;
  el.ontouchstart = go;
}

assert(typeof go === "undefined");
```

Const

```
const PI = Math.PI;

PI = 22/7; // error! cannot reassign

const E; // SyntaxError! need initializer

// plus all the yummy `let` semantics
```

Iteration

```
for (let x of ["one", "two", "three"]) { }

for (let value of set) { }

for (let [key, value] of map) { }

// customizing iteration requires some magic
```

Magic

- Template strings
- Symbols
- Generators
- Proxies

Template Strings

```
let x = 1;
let y = 2;

let s = `${x} + ${y} = ${x + y}`;

let multiline = `<html>
  <body>
    Hello world!
  </body>
</html>`;
```

Template Strings

```
let url = "http://example.com/";
let query = "Hello & Goodbye";
let s = safehtml`
<a href="${url}?q=${query}“
   onclick="alert('${query}')">
  ${query}
</a>`;

assert(s === `<a href="http://example.com/?q=Hello%20%26%20Goodbye"
   onclick="alert('Hello&#32;\x26&#32;Goodbye')">
  Hello &amp;amp; Goodbye
</a>`);
```

Symbols

```
function S3Bucket(apiKey, apiSecret) {
  this._apiKey = apiKey;
  this._apiSecret = apiSecret;
}

S3Bucket.prototype.request = function (url) {
  let signature = calculateSignature(this._apiKey, this._apiSecret);
  this._sendRequest(url, signature);
};

S3Bucket.prototype._sendRequest = function () { };
```

Symbols

```
function S3Bucket(apiKey, apiSecret) {
  this.request = function (url) {
    let signature = calculateSignature(apiKey, apiSecret);
    sendRequest(url, signature);
  };

  function sendRequest() { }
}
```

Symbols

```
let apiKey = new Symbol(), apiSecret = new Symbol(), sendRequest = new Symbol();

function S3Bucket(theApiKey, theApiSecret) {
  this[apiKey] = theApiKey;
  this[apiSecret] = theApiSecret;
}

S3Bucket.prototype.request = function (url) {
  let signature = calculateSignature(this[apiKey], this[apiSecret]);
  this[sendRequest](url, signature);
};

S3Bucket.prototype[sendRequest] = function () { };
```

Symbols

```
private @apiKey, @apiSecret, @sendRequest;

function S3Bucket(theApiKey, theApiSecret) {
  this.@apiKey = theApiKey;
  this.@apiSecret = theApiSecret;
}

S3Bucket.prototype.request = function (url) {
  let signature = calculateSignature(this.@apiKey, this.@apiSecret);
  this.@sendRequest(url, signature);
};

S3Bucket.prototype.@sendRequest = function () { };
```

Back to Iterators

```
import { iterator } from "@iter"; // a symbol in the standard library

let obj = {};
obj[iterator] = function () {
  return {
    next() { return 5; }
  };
};

for (n of obj) {
  assert(n === 5);
}
```

Generators

```
function* naturalNumbers() {
  let current = 0;
  while (true) {
    yield current++;
  }
}

let seq = naturalNumbers();
assert(seq.next() === 0);
assert(seq.next() === 1);
assert(seq.next() === 2);
```

Generators Return Iterators

```
// Generator functions return iterators
for (let n of naturalNumbers()) {
  console.log(n);
}

// Use them to create your own iterators
obj[iterator] = function* () {
  yield 5;
};
```

Generators Are Shallow Coroutines

```
function* demo() {
  console.log("a");
  yield;
  console.log("b");
  yield;
  console.log("c");
}

let seq = demo(); // “paused,” with no code executed
seq.next(); // execution resumes; logs "a"; then pause
seq.next(); // execution resumes; logs "b"; then pause
seq.next(); // execution resumes; logs "c"; enters “done” state
seq.next(); // throws `StopIteration`
```

Shallow Coroutines for Async

```
spawn(function* () {
  try {
    let post = yield getJSON("/post/1.json");
    let comments = yield getJSON(post.commentURL);

    yield fadeInCommentUI();
    comments.innerHTML = template(comments);
  } catch (e) {
    comments.innerText = e.message;
  }
});
```


http://taskjs.org/

Proxies

```
let handler = {
  getOwnPropertyDescriptor(target, name) { },
  getOwnPropertyNames(target) { },
  getPrototypeOf(target) { },
  defineProperty(target, name, desc) { },
  deleteProperty(target, name) { },
  freeze(target) { },
  seal(target) { },
  preventExtensions(target) { },
  isFrozen(target) { },
  isSealed(target) { },
  isExtensible(target) { },
  ⋮
```

Proxies

```
  ⋮
  has(target, name) { },
  hasOwn(target, name) { },
  get(target, name, receiver) { },
  set(target, name, val, receiver) { },
  enumerate*(target) { },
  keys(target) { },
  apply(target, thisArg, args) { },
  construct(target, args) { }
});

let theTarget = {};
let virtualObject = new Proxy(theTarget, handler);
```

The Future Now

- The future of JS: it's important!
- Follow @esdiscuss.
- Stay tuned for more on es6isnigh.com.
- Be adventurous; use a transpiler!
- *Skate where the puck will be.*