Modern JavaScript and PhoneGap

PhoneGap Day EU 2017

Kerri Shotts • @kerrishotts

Hi!

- Used PhoneGap for over six years
- Authored Five books about PhoneGap
- Apache Cordova committer
- One of many moderators:
    - Cordova Google Group
    - PhoneGap Adobe Forums
- I love retro technology! :-)

Modern JavaScript Versions

Remember ECMAScript 5?

Release year: 2009

- The version we all know and love (~ish?)
- Supported by all modern mobile web views[1]
  - iOS 6+, IE 10+, Edge (forever), Android 4.4+
- Reasonably modern (`map` , `reduce` , getters/setters, etc.)
- Things have changed a lot since then…

1. http://caniuse.com/#feat=es5

2015[1]	Block-scoped `let` & `const`	Destructuring and named parms
	Default parameters	Rest and Spread operator (`...`)
	`for...of` loops and Iterators	Arrow functions (`=>`)
	Template strings & interpolation	Improved literals (object, `0b10`)
	Generators (`*` / `yield`)	Symbols, Maps & Sets, Promises
	`class` syntactic sugar & `super`	Modules (`import` , `export`)
2016[2]	Exponent (`**`)	`Array.prototype.incudes()`
2017[3]	`async / await`	String padding 😉
	Shared memory	Atomics

1. https://github.com/lukehoban/es6features#readme; the list here is not a complete representation of *all* features
2. http://www.2ality.com/2016/01/ecmascript-2016.html
3. http://www.2ality.com/2016/02/ecmascript-2017.html

Before we go any further…

Some Very Important Caveats!

Caveats

- *NOT* a performance optimization
- Adds a build step
- Debugging can be… interesting
- Best iOS performance requires `WKWebView`
- May need some time to use effectively

Performance Change	Chrome 55	Edge 15	Safari 10
Arrow functions	N/C	+1.2x	N/C
let compound	-1.6x	N/C	N/C
Classes	N/C	-1.5x	N/C
super	-4x	-1.7x	-15x
Destructuring	-16x	-53x	-23x
for … of array	-17x	-7x	-1.3x
for … of object	-1.8x	-4x	-2.3x
Map & Set	-4x	-23x	-8x
rest	+1.3x	+14x	-33x
spread	-22x	-1.7x	-5x
Template string	-1.2x	+1.4x	-18x

Source: https://kpdecker.github.io/six-speed/ (2017/01/04) | N/C: “no change”

Don’t Despair!

Don’t let those numbers scare you!

- Micro-benchmarks don’t always reflect the real world
- Performance is steadily improving
- Capable of running an emulator at full tilt
  - … on iOS using `WKWebView` (JIT compilation FTW)

Some unscientific numbers

Device	GB4	Web View	Mode	ES6 IPF (mips)	ES5 IPF (mips)	ES3 IPF (mips)
MacBook Pro	3574	Safari 10	reg	75650 (4.51)	79783 (4.75)	78381 (4.67)
			min	72167 (4.30)	80301 (4.77)	72953 (4.35)
iPad Pro 12.9"	3000	Safari 10	reg	81344 (4.88)	81720 (4.89)	83584 (5.01)
			min	80542 (4.83)	72315 (4.34)	81182 (4.87)
iPad Mini 4	1638	Safari 10	reg	32791 (1.97)	36222 (2.17)	39195 (2.35)
			min	36501 (2.19)	38676 (2.32)	36715 (2.20)
Tab S 8.4"	783	Chrome 54	reg	2614 (0.13)	3350 (0.17)	2394 (0.11)
			min	2847 (0.14)	3557 (0.19)	1950 (0.09)
iPad Pro 12.9"	3000	UIWebView	reg	100 (0.01)	100 (0.01)	100 (0.01)
			min	100 (0.01)	100 (0.01)	100 (0.01)

Note: Of course, this is *highly sensitive* to the ES2015+ features that you use.
MacBook Pro: Late 2014, 2.2GHz i7 16GB RAM; *GB4* = Geekbench 4 single-core score; *min* = minified & dead code removed

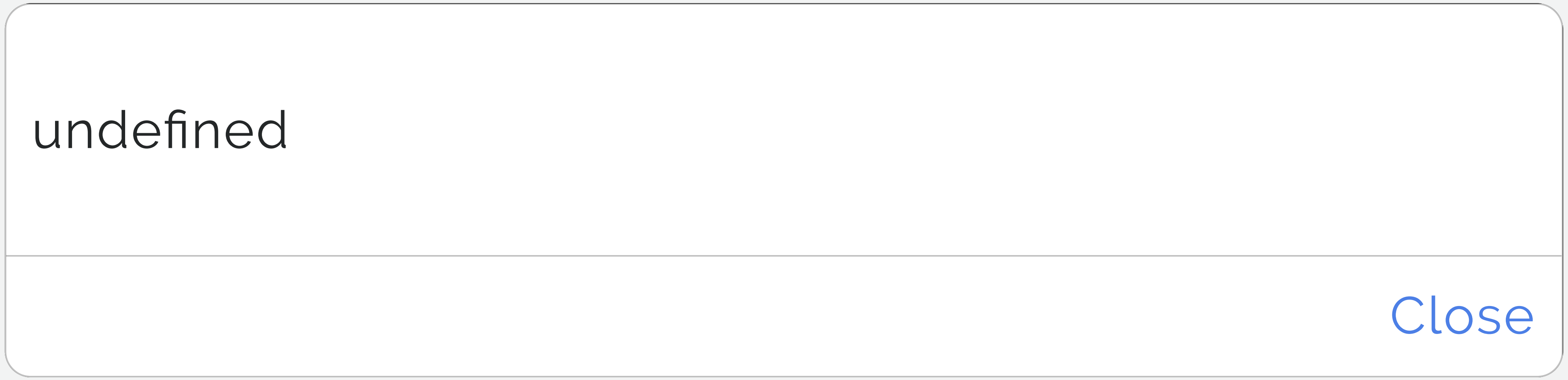
A whirlwind tour

Dang it, *this!*

```
var app = {
  text: "Hello, PhoneGap Day Attendees!",
  sayHi: function() { alert(this.text); },
  start: function() {
    document.getElementById("clickme")
      .addEventListener("click", this.sayHi, false);
  }
}

app.start();
```

Wah wah 🚫

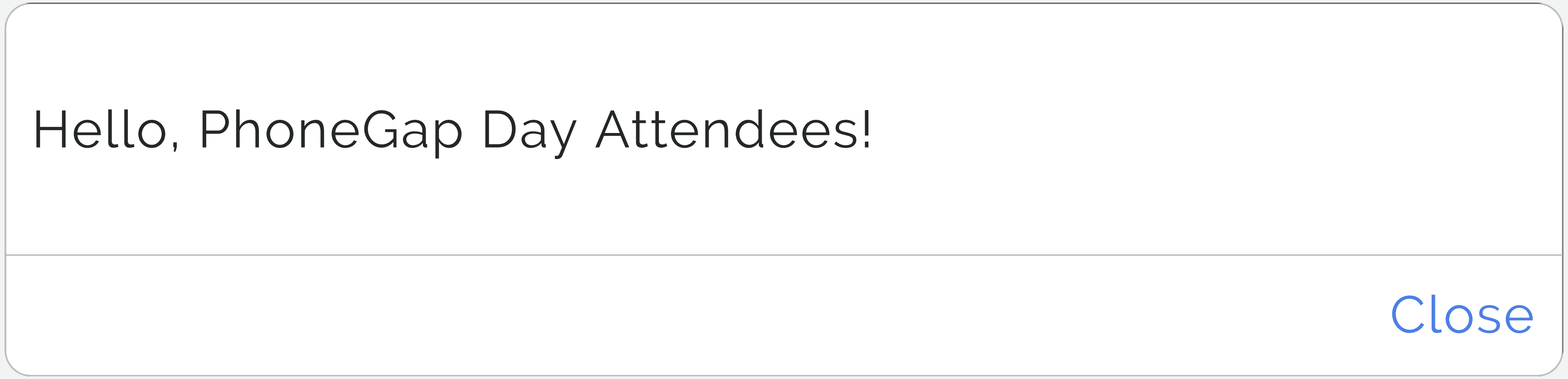

undefined

Arrow functions (=>) & Classes

```
class App {
  constructor({text = "Hello, world!"} = {}) {
    this.text = text;
  }
  start() {
    document.getElementById("clickme")
        .addEventListener("click", () => this.sayHi(), false);
  }
  sayHi() { alert(this.text); }
}
const app = new App({text: "Hello, PhoneGap Day Attendees!"});
app.start();
```

ES5 equivalent: `(function() { this.sayHi(); }).bind(this)`

Hi! 🎉


Hello, PhoneGap Day Attendees!

Array-like conversion

ES5 requires `slice`:

```
var elList = document.querySelectorAll("a"),
    elArr = [].slice.call(elList, 0);
```

ES2015+ (with the standard library):

```
let elArr = Array.from(document.querySelectorAll("a"));
```

Spread/Rest is awesome (...)

Even shorter than `Array.from`:

```
let elArr = [...document.querySelectorAll("a")];
```

Easy variadic arguments:

```
function sum(start = 0, ...nums) {
  return nums.reduce((acc, val) => acc + val, start);
}
console.log(sum(1, 5, 10, 99)); /* 115 */
```

Destructuring

```
[a, b] = [b, a]  // swap!
```

“Multiple return values”:

```
function duplicate(str) {
  return {result: str + str,
          error: !str ? "no string" : null};
}

let {result, error} = someFunction("abc");
let {result:r, error:err} = someFunction("acb"); // you can rename
let {result} = someFunction("abc");              // or even ignore!
```

Named Parameters & Defaults

```
class Button {
  constructor({type = "default", text = "",
              x = 0, y = 0, w = 100, h = 44} = {}) {
    this.type = type;
    this.text = text;
    this.frame = {x, y, w, h};
    this.bounds = {x: 0, y: 0, w, h};
  }
}
let button = new Button ({type: "round", text: "Click me",
                          x: 100, y: 100});
```

Template Strings

```
let x = 4, y = 10;
console.log(`x + y => ${x} + ${y} => ${x + y}`);
```

⇒ x + y => 4 + 10 => 14

Multi-line strings (preserving ↵):

```
let template=`<ul>
    <li><span></span></li>
</ul>`;
```

Promises, promises

Hopefully already familiar to you…

```
function requestFileSystem({type = window.PERSISTENT,
                            quota = 5 * 1024 * 1024} = {}) {
  return new Promise((resolve, reject) => {
    window.requestFileSystem(type, quota, resolve, reject);
  });
}
```

But ES2017 has something better…

async / await

```
async function readFile(name) {
  const fs = await requestFileSystem({
    type: window.PERSISTENT, quota: 10 * 1024 * 1024});
  return await readFile(await getFile(name));
}
async function start() {
  try {
    const data = await readFile("poem.txt");
    readPoemAloud(data);
  } catch (err) {
    alert (err);
  }
}
```

Modules

Static Analysis, FTW!

📄 math.js:

```
export function add(a, b) {
    return a+b;
}
```

📄 index.js:

```
import {add} from "math.js";
console.log(add(4, 3)); /* 7 */
```

PhoneGap Examples

Geolocation with ES2017

```
function getLoc(options) {
  return new Promise((resolve, reject) => {
    navigator.geolocation.getCurrentPosition(p => resolve(p.coords),
      reject, options);
  });
}
async function start() {
  try {
    const {timestamp, coords:{latitude, longitude}} = await getLoc();
    console.log(`At ${latitude}, ${longitude} on ${timestamp}`);
  } catch(err) {
    console.log(`Error ${err.code}: ${err.message}`);
  }
}
```

File Transfer with ES2017

```
function uploadFile({source, target, options} = {}) {
  return new Promise((resolve, reject) => (new FileTransfer()).
    upload(url, to, resolve, reject, options));
}
async function start() {
  try {
   const {responseCode, response, bytesSent} = uploadFile({
      url: "cdvfile://localhost/persistent/test.txt",
      to: "http://www.example.com/upload.php",
      options: { mimeType: "text/plain",
                 fileKey: "file", fileName: "test" }});
  } catch (err) { /* do something with the error */ }
}
```

Do you sense a pattern?

```
function promisify(fn, thisArg = this, {split = 0} = {}) {
  return function __promisified__(...args) {
    const afterArgs = args.splice(split), beforeArgs = args;
    return new Promise((resolve, reject) => {
      try {
        fn.apply(thisArg, beforeArgs.concat(resolve, reject,
          ...afterArgs));
      } catch (err) { resolve(err); }
    });
  }
}
```

Easy wrappers for Cordova plugin APIs! *

```
const getLoc = promisify(
  navigator.geolocation.getCurrentPosition,
  navigator.geolocation // "this" arg
);
const {timestamp, coords:{latitude, longitude}} = await getLoc();

const ft = new FileTransfer();
// upload signature: url, to [split], success, error, options
const uploadFile = promisify(ft.upload, ft, {split: 2});
const r = await uploadFile(url, to, options);
```

* Applies to Cordova plugin APIs that use the success, error form; could be made more generic

Where can I use this now?

Native support (%coverage)

OS	ES2015	ES2016	ES2017
Android (Chrome)	97% (51+)	100% (55+)	53% (56+)
Edge 15	100%	100%	39%
Edge 14	93%	-	-
iOS 11*	100%	100%	98%
iOS 10	100%	61%	42%
iOS 9	54%	-	-

* Based on current status in Safari Technological Preview 11
Note: Some of the tests based on existence, not completeness.
Sources: ES2015, ES2016, ES2017

But, I want it everywhere!

ES2015+ ⇒ ES5 😄 💃

or, *The Rise of the Transpilers*

Common Transpilers

These can all transpile ES2015* (feature support may vary)

- Babel (née es6to5)
- TypeScript
- Bublé **
- Traceur

* **Note:** Not every ES2015+ feature can be transpiled effectively (if at all), such as proxies, shared memory, atomics, built-in subclassing, and tail call elimination
* **Note:** Most transpilers need core-js to polyfill the standard library.
** Doesn't attempt to transform non-performant or non-trivial ES6 features; *also very young*

Re: Module syntax

No implementation! 😱

But we can fix that…

Module support using Bundling 🛍

Dependency management & `import` / `export` (and CommonJS, AMD, etc.) support

Bundler	Babel	Bublé	Coffee	Typescript	Traceur
Webpack	✓	✓	✓	✓	✓
JSPM	✓	—	—	✓	✓
Browserify	✓	✓	✓	✓	✓

Plugins exist for just about every transpiler if you look hard enough.

PhoneGap Integration

- Manual
  - Just run each tool's CLI… *every time…*
  - Error prone — you might forget!
- Automatic
  - `gulp` / `grunt` task runners
  - `npm run` scripts
  - Plugin / Project hooks

Setting up (npm run scripts)

- Determine ES2015+ code location
- Install Webpack & Transpiler
- Configure Webpack & Transpiler
- Add build scripts to `package.json`

Sibling Structure

```
project-root/
    config.xml
    www/
        index.html
        (ts|es)/
            index.(ts|js)
        js/
            bundle.js ← (gen)
```

External Structure

```
project-root/
    config.xml
    www.src/
        index.html
        (ts|es)/
            index.(ts|js)
    www/
        index.html ← (copied)
        js/
            bundle.js ← (gen)
```

Install Webpack & Transpiler

```
[user@dev] $ | npm install --save-dev webpack
```

Typescript:

```
[user@dev] $ | npm install --save-dev ts-loader typescript core-js
```

Babel:

```
[user@dev] $ | npm install --save-dev babel-loader babel-core babel-polyfill \
             |   babel-preset-es2015 babel-preset-es2016 babel-preset-es2017 \
             |   babel-plugin-transform-runtime
```

Note: `core-js` is a standard library polyfill; depending on your feature use and targets you may not need it.

Configure TypeScript

Create `tsconfig.json`:

```
{
  "compilerOptions": {
    "allowJs": true,
    "target": "es5",          // es2015, es5, es3
    "module": "es2015",       // required for tree shaking
    "inlineSourceMap": true
  },
  "include": [
    "www.src/es/**/*"         // or www/es/**/* if sibling
  ]                             // "ts" if using typescript features
}
```

Configure Babel

Create `.babelrc`:

```
{
  "presets": [
    ["es2015", {
      "loose": true,   // best performance
      "modules": false // required for tree shaking
    }], "es2016", "es2017"
  ],
  "plugins": ["transform-runtime"] // reduces repetition in bundle
}
```

Configure Webpack

Create `webpack.config.js`:

```
module.exports = {
  devtool: "inline-source-map",
  context: path.resolve(__dirname, "www.src"), // if sibling, use   __dirname, "www"
  entry: "./" + path.join("es", "index.js"),   // will fail without ./!; ts if typescript
  output: { filename: "bundle.js",
            path: path.resolve(__dirname, "www", "js") },
  module: { loaders: [{
              test: /\.(ts|js|jsx)$/,          // remove ts for babel
              loader: 'ts-loader',             // or babel-loader
              exclude: /node_modules/,
              options: { entryFileIsJs: true } // only for js with typescript
          }]
  }
}
```

Add run script to package.json

(assuming `cordova` and `webpack` are installed locally)

```
"scripts": {
    "build:ios":
        "webpack && cordova build ios"
}
```

```
[user@dev] $ | npm run build:ios
```

Note: if using *sibling* layout, you might want to delete the duplicate code in the platform `www/es` folders. Otherwise, you'll end up copying your ES2015+ code *and* the resulting bundle to the app bundle.

Magic!

cordova-plugin-webpack-transpiler can do this for you:

```
[user@dev] $ | cordova plugin add --save \
                 cordova-plugin-webpack-transpiler \
                 --variable CONFIG=typescript|babel
```

Or, you can use templates, too:

- Typescript: cordova-template-webpack-ts-scss
- Babel: cordova-template-webpack-babel-scss

Fork, translate, and/or improve it: https://github.com/kerrishotts/cordova-plugin-webpack-transpiler

What about tests?

… and code coverage?

… and linting?

Tests

```
[user@dev] $ | npm install --save-dev mocha chai
[user@dev] $ | npm install --save-dev ts-node        # for TypeScript
[user@dev] $ | npm install --save-dev babel-register # for Babel
```

Add `test` to `package.json:scripts`[*]

```
"test": "mocha" // TypeScript (need ./test/_bootstrap.js)
"test": "mocha --compilers js:babel-register"   // Babel
```

Then `npm test`

[*] Assumes tests are in `./test`
_bootstrap.js: `require("ts-node").register();`

Code coverage (Babel)

`npm install --save-dev instanbul`, then in `.babelrc`:

```
{
  "presets": ["es2015", ...],
  "plugins": ["transform-es2015-modules-commonjs", ...]
  "env": {
    "test": {
      "plugins": ["istanbul"]
    }
  }
}
```

Code coverage (Babel, 2)

`npm install --save-dev cross-env nyc` and configure (in `package.json`):

```
"nyc": {
  "require": ["babel-register"],
  "reporter": ["text", "html"],
  "sourceMap": false,
  "instrument": false // instanbul instrumented already
}
```

And create a `npm run` script:

```
"cover": "cross-env NODE_ENV=test nyc npm test"
```

Linting

`eslint` works just fine with ES2015! (`tslint` for Typescript)

```
[user@dev] $ │ npm install --save-dev eslint
```

📄 `package.json` :

```
"scripts": {
    "lint": "eslint www.src test"
}
```

```
[user@dev] $ │ npm run lint     # or, write a plugin /
             │                  # project-lvel hook! ;-)
```

Tips

Tips

- You don't have to convert overnight — a little at a time is fine
- `var` hasn't gone away
- Don't get carried away — eye-strain alert!
  - True especially with descructuring and template strings
- Use `for...of` instead of `for...in` & `hasOwnProperty()`
- Don't assume `=>` functions are drop-in replacements
- Careful using arrow functions with `describe` & `it` in your tests

Tips (2)

- Try to declare `let` / `const` at the top of each scope (for Chrome's benefit)
- Use `var` instead of `let` in tight, nested loops where performance is critical
- *Do* minify & tree shake — reduces file size and startup time
- Don't count on minified code as a performance optimization (results highly variable)
- Do use `const` to identify unchanging *references*
    - But don't think of the variable as *immutable* — it isn't

Thanks!

https://github.com/kerrishotts/pgday/2017/modern-javascript-and-phonegap

@kerrishotts

This slide intentionally left blank