ES6 and Web Components

JavaScript has a long history of being difficult to structure and maintain. To deal with this complexity a swath of frameworks appeared over the years. Prototype.js was quickly followed by jQuery and hounded by Dojo, YUI, Mootools and many others. A second generation emerged focused on structuring apps heralded by Backbone, Angular, Ember, React and Polymer.

JavaScript can be a hard scene to follow. *So what's next?* Nothing! Well ok, not quite nothing, the web platform inches forward very slowly using cues from the frameworks of yore. The W3C and WhatWG incorporate the concepts into the living standards and then browsers implement them eventually.

Two key recent additions to the web platform now just within reach and worth consideration for serious use:

- JavaScript version 6 (ES6)
- Custom Elements (a part of the broader Web Components initiative)

Come to this talk to learn how to get started writing structured apps using future friendly vanilla web platform code.

ES6 and Web Components

Futuristic Flavors

What to expect

This talk walks through a possible future of the web.

- A Short History of JS Frameworks
- Web Components: The Good Part™
- ES6 Syntax: ZOMGCAKES

JS Frameworks

A Short History

Original Series 2006-2008

- Prototype and Scriptaculous
- jQuery
- Mootools
- YUI
- Dojo

Prototype

- the first, somewhere in 2006
- no docs for a very long time
- concerned with leveling out DOM API across browsers
- extends built in DOM Elements with new methods
- Ajax became a thing

Prototype

```
var div = document.createElement('div')

Element.extend(div)

div.addClassName('pending').hide()

document.body.appendChild(my_div)
```

Prototype

```
new Ajax.Request('/hello', {

  method:'get',

  onSuccess: function(transport) {
    var response = transport.responseText || "no response"
    alert("Success! \n\n" + response)
  },

  onFailure: function() {
    alert('Something went wrong...')
  }
})
```

Scriptaculous

- primary value prototypejs w/ effects
- utilized timeouts
- completely unnecessary today

Scriptaculous

```
<div id="shake_demo">
  <a href="#" onclick="new Effect.Shake('shake_demo'); return false;">
    Click me to shake!
  </a>
</div>
```

jQuery

- an alt view: selection, iterative apply, chaining
- built in animations
- dominated early, still does, but not as necessary
- heavily influenced the browser
- later grew an ecosystem: plugins, ui, mobile
- fantastic open source foundation btw

jQuery DOM API

```
$('button.continue').html('say hello').click(function() {
  alert('Hi!')
})
```

jQuery Ajax

```
$.ajax({

  url: '/hello',

  success: function(data) {
    $('#hi').html('<b>' + data + '</b>')
  }
})
```

jQuery Animation

```
$('#book').slideDown('slow', function() {
  console.log('animation complete')
})
```

Mootools

- very clean Class system foundation

Saaaame problems being solved but differently. (Cross browser DOM, Events, Ajax and Animation.)

Mootools DOM API

```
var el = new Element('div#bar.foo')

el.addEvent('click', function() {
  alert('clicked!')
})
```

```
var req = new Request({

  url: '/hello',

  method: 'get',

  onRequest: function() {
    el.set('text', 'loading...')
  },

  onSuccess: function(res) {
    el.set('text', res)
  },

  onFailure: function() {
    el.set('text', 'Sorry!')
  }
})

// and to send it:
req.send()
```

Mootools Class System

```
var Animal = new Class({
  initialize: function(age){
    this.age = age
  }
})

var Cat = new Class({
  Extends: Animal,
  initialize: function(name, age) {
    this.parent(age)
    this.name = name
  }
})
```

YUI

- extremely modular and configurable
- bailed on early java-ish api for a jquery-like thing
- no longer maintained *(a key problem w/ corp stewardship)*

YUI

```
YUI().use('node', function(Y) {

  var onClick = function(e) {
    Y.one('#event-result').setHTML('hello world')
  }

  Y.one('#container').on('click', onClick)
})
```

Dojo

“Dojo already did that” - Pete Higgins

(It did and still does, quite literally, everything aforementioned.)

Dojo

```
require(['dojo/dom', 'dojo/dom-construct'], function (dom, domConstruct) {

  var greetingNode = dom.byId('greeting')

  domConstruct.place('<i> Dojo!</i>', greetingNode)
})
```

Problems

- differences between browser APIs smoothing out
- selector engines built in “natively”
- animation w/ CSS became the better practice
- the mobile was increasingly happening
- most apps lacked structure, modules
- iterative apply and chaining *can be* super hard to test

THE Problem

These libraries are bloated, creating tightly coupled code incompatible elsewhere and mutating state everywhere creating janky experiences.

Rise of the Microlibrary

- XUI, 2008
- Lawnchair, 2009
- Zepto, 2010

XUI

- 2008 post iPhone
- mobile Safari constraints motivated: <10kb
- a jQuery work-alike
- introduction text read: micro-tiny *

* possibly seeding the term microlibrary in single origin coffeeshops and craft beer establishments

XUI

```
x$('#hello').text('click me').click(function() {
  console.log('hi')
})
```

Lawnchair

- models!
- 2009 as PhoneGap got first traction
- abstracting the baffling array of client storage methods
- still trucking today

Lawnchair

```
Lawnchair({name:'testing'}, function(store) {

  // Create an object
  var me = {key:'brian'}

  // Save it
  store.save(me)

  // Access it later... Yes even after a page refresh!
  store.get('brian', function(me) {
    console.log(me)
  })
})
```

Zepto

- sometime in 2010
- same spirit of XUI but an exact clone of the jQuery API
- code is beautiful: simple and clear
- still quite popular today

Zepto Touch Addons

```
<style>.delete { display: none; }</style>

<ul id="items">
  <li>List item 1 <span class=delete>DELETE</span></li>
  <li>List item 2 <span class=delete>DELETE</span></li>
</ul>

<script>
$('#items li').swipe(function(){
  $('.delete').hide()
  $('.delete', this).show()
})

$('.delete').tap(function(){
  $(this).parent('li').remove()
})
</script>
```

Still not quite right

- good: solved the same problems with less code
- bad: didn’t solve the structure problems
- ugly: concerns being separated *but* inconsistently

Next Generation

MV*

- Backbone
- Angular
- Ember
- React
- Polymer

Backbone

- first thing of its kind arriving 2010ish
- Views, Models, Collections and a Router
- very tiny, extremely well written
- almost no coupling, use a little or a lot
- opinionated about not being opinionated
- huge ecosystem, of note: Marionette and AmpersandS
- still v popular today

```
var DocumentRow = Backbone.View.extend({

  tagName: "li",

  className: "document-row",

  events: {
    "click .icon":          "open",
    "click .button.delete": "destroy"
  },

  initialize: function() {
    this.listenTo(this.model, "change", this.render)
  },

  render: function() {
    this.el.innerHTML = 'draw stuff to screen'
  }
})
```

Angular

- Google
- rigor of design familiar to the enterprise
- kind of strange nomenclature
- abstractions are *very* tightly coupled
- cross browser testing built in (Karma)
- great mobile lib: Ionic
- **custom elements concept: Angular Directives**

```
<helloPerson person="brian"></helloPerson>

<script>
var app = angular.module('app', [])

app.directive('helloPerson', function() {
  return {
    restrict: 'E',
    template: 'Hello {{ person.name }}',
    scope: {person:'=person'}
  }
})

app.controller('ctlr', function($scope, $http) {
  $scope.brian = {}
  $scope.brian.name = "Brian LeRoux"
})
</script>
```

Ember

- Models, Views, Controllers, Components, Router
- strong sep of concerns but pieces tightly coupled
- Rails style convention over config and metaprogramming
- really well writ docs, onboarding and governance
- slow and kind of large: not appropriate for mobile
- **Components! a sort of custom Mustache partials thing ***

* not even close to Web Components spec despite docs claim to such

Ember

```
<script type="text/x-handlebars">
{{ hello-control name="Brian LeRoux" }}
</script>

<script type="text/x-handlebars"
        data-template-name="components/hello-control">
Hello {{ name }}
</script>

<script>
Ember.Application.create()
</script>
```

React

- Facebook
- only concerned with Views
- not initially well received for freaky template syntax
- introduced the concept of a Virtual DOM
- deceptively simple programming model
- crazy fast
- promising early mobile libs: TouchstoneJS and Reapp
- explicitly avoids mutating state

...and more custom elements!

React

```
var HelloMessage = React.createClass({
  render: function() {
    return <div>Hello {this.props.name}</div>
  }
})

// ok, this is nice
React.render(<HelloMessage name="Brian LeRoux" />, document.body)
```

Polymer

- wait, Google again?
- battle hardening next gen specs
- implementing backwards compat polyfills and tools
- kind of big and also slow
- **has actual browser ready Custom Elements!**

Polymer

```
<link rel="import" href="../bower_components/polymer/polymer.html">

<polymer-element name="hello-world" attributes="who">
  <template>
    <p>Hello <strong>{{who}}</strong> :)</p>
  </template>
  <script>
    Polymer('hello-world', {
      who: 'Brian LeRoux'
    })
  </script>
</polymer-element>
```

State of the art

- module systems coalescing but remain incompatible
- network, routing and data sync all v different
- ***everyone has a custom elements implementation***

Web Components

The Good Part

Web Components

A tale of four w3c specifications:

- <template> tag
- HTML imports
- Shadow DOM
- Custom Elements

<template></template>

```
<!-- this kinda thing, blocking call to the paint -->
<script id="hello" type="text/ejs">
Hello <%= name %>.
</script>

<!-- becomes this sorta deal, big diff is this tmpl is inert -->
<template id="hello">
Hello <%= name %>.
</template>
```

meh

Web Components

A tale of four w3c specifications:

- ~~<template> tag~~
- HTML imports
- Shadow DOM
- Custom Elements

HTML Imports

```
<!-- imports a Web Component -->
<link rel="import" href="../bower_components/polymer/polymer.html">
```

Troubles

- its slow
- networks fail
- Mozilla doesn’t want to add HTML Imports
- Google does
- Apple has a different proposal
- MSFT is watching it play out

Web Components

A tale of four w3c specifications:

- ~~<template> tag~~
- ~~HTML imports~~
- Shadow DOM
- Custom Elements

Shadow DOM

- sounds v cool bb
- hides the implementation of a Web Component
- CSS no longer leaks globally
- not *a thing you do* so much as *thing you get*

Okaaaaay

Web Components

A tale of four w3c specifications:

- ~~<template> tag~~
- ~~HTML imports~~
- ~~Shadow DOM~~
- Custom Elements

Custom Elements!

The Good Part™ of Web Components

- Angular, Ember, React and Polymer all impl the concept
- a single blocking API call to register
- supports extending built ins
- otherwise a good old fashioned DOM Element

Custom Elements: Basics

```
var Hello = document.registerElement('hello-world')

document.body.appendChild(new Hello)

// creates <hello-world></hello-world>
```

Custom Elements: Extending

```
<script>

  var SupaButton = document.registerElement('supa-button', {

    prototype: Object.create(HTMLButtonElement.prototype),

    extends: 'button'
  })
</script>

<button is=supa-button>Hello World</button>
```

Custom Elements: Lifecycle

```
var ptype = Object.create(HTMLElement.prototype, {

  createdCallback: function() {
    this.innerHTML = 'hello world'
  },

  attachedCallback: function() {
    console.log('attached to the DOM')
  },

  detachedCallback: function() {
    console.log('removed from the DOM')
  },

  attributeChangedCallback: function(attr, prev, current) {
    console.log('attr changed')
  }
})

var Supa = document.registerElement('supa-hello', {prototype:ptype})
```

How?

- WebComponents.js polyfill (has all the toys)
- we only need to polyfill document.registerElement()
- ideally we incorporate with a module system

Browser support

- Chromes only (As in Chrome and Opera)
- Firefox likely
- IE a maybe
- WebKit (Safari) coming on board (DOM != ES6 classes...)

ES6

Hey lets talk about that for a hot sec!

After JS

- Programming language popularity is fickle
- JS won b/c it runs everywhere and is super fast

The next thing isn’t Go or Rust. Its JS, again.

Module defn

```
// define a module in hello.js
export default function(person) {
  console.log('Hello ' + person)
}
```

Module consumption

```
// consume said module elsewhere
import hi from 'hello'

hi('Brian LeRoux')
// logs: Hello Brian LeRoux
```

Terse obj literal methods

```
var o = {

  hello(msg) {
    console.log('hi there ' + this.fmt(msg))
  },

  fmt(str) {
    return str.toUpperCase()
  }
}

o.hello('people')
// logs: hi there PEOPLE
```

Classes!

```
class SkinnedMesh extends THREE.Mesh {

  constructor(geometry, materials) {
    super(geometry, materials)

    this.idMatrix = SkinnedMesh.defaultMatrix()
    this.bones = []
    this.boneMatrices = []
  }

  update(camera) {
    super.update()
  }

  static defaultMatrix() {
    return new THREE.Matrix4()
  }
}
```

Fat arrows

```
['banana', 'orange', 'apple'].forEach(fruit => fruit.toUpperCase())
```

Template Strings

```
let msg = `Dear ${ person }

Thanks for being amazing!

                  <3 Brian
`
```

Other niceties

- let, const
- destructuring assignment
- default, rest and spread

Compiling

- browserify: bring the node module system to the browser
- babeljs: compile ES5 src to ES6 src
- npm scripts: like grunt or gulp without a dep

Publishing

I prefer npm to bower.

- has a module concept
- dist is separated from revision control: and should be

Bringing it together

- author src in ES6
- dist src as ES5
- package.json point “main” key to ES5 dist src
- deploy to Github pages!
- src maps for debugging

```
class DateSpan extends HTMLSpanElement {

   createdCallback() {
     this.textContent = "Today's date: " + new Date().toJSON().slice(0, 10)
   }
}
```

Problems

- Browsers still not quite implementing
- the webcomponents.js polyfill is kind of weird
- CSS is still a total global clusterfuck
- blocking call on document.registerElement could be slow
- nothing mutates state ***but nothing stopping it either***

The Future

- unlikely to be the same as today
- optimize for change!
- this CSS business srsly we need to fix that
- separate and independently revision your concerns
- immutable state is A Good Thing™
- vdom and batching concepts likely to lift to platform

thx!

find me

b@brian.io

http://twitter.com/@brianleroux

http://github.com/brianleroux

https://www.npmjs.com/~brianleroux

resources

http://goo.gl/7pRxiu

https://github.com/brianleroux/date-today

http://brian.io/date-today (view the src!!)

https://medium.com/@brianleroux/es6-modules-amd-and-commonjs-c1acefbe6fc0

http://h3manth.com/new/blog/2015/custom-elements-with-es6/