Modern JavaScript: The Smalltalk Influence

Talk, by Allen Wirfs-Brock
Mozilla Research Fellow
Project Editor, Ecma-262 (The JavaScript Standard)
@awbjs

Smalltalks 2014
November 7, 2014
Córdoba, Argentina
Every Computing Era Has a Dominant Application Platform

- Corporate Computing Era: IBM Mainframes
- Personal Computing Era: Microsoft/Intel PC
- Ambient Computing Era: The Web is the Platform?
The Web Is the Platform

The Web Application Platform

Frameworks and Libraries

HTML
CSS
JavaScript
SVG

Language Runtime

Rendering
Layout
Styling
Network
Local storage
User Input

HTML5

The Web belongs to all of us. Make your mark and show your support for the Web's largest public resource.

About Mozilla

Mozilla Firefox Start Page

Google
Each Computing Era has had Canonical Programming Languages

- Corporate Computing Era – COBOL/Fortran
- Personal Computing Era – C/C++ family
- Ambient Computing Era – JavaScript ??
Why JavaScript?

Because “Worse is Better”

The economics of ubiquity.

- It’s already there
- Widest reach
- Lowest risk
- Write libraries and apps once
- Single knowledge and skill set

Is it even possible to replace it?

Dick Gabriel
http://www.dreamsongs.com/WorseIsBetter.html

http://odotocode.com/Blogs/scott/archive/2009/03/18/signs-that-your-javascript-skills-need-updating.aspx
Not Just In Web Browsers

http://nodejs.org/

Firefox OS

http://nodebots.io/

NodeBots

Robots powered by JavaScript

Johnny-Five

https://github.com/rwaldron/johnny-five

“SEVEN DAYS IN MAY”

THE JOHN FRANKENHEIMER-JOEL PRODUCTION
PRODUCED BY EDWARD LEWIS • DIRECTED BY JOHN FRANKENHEIMER
A PARAMOUNT RELEASE
JavaScript Influences

• Scheme
  ✓ First class closures

• Self
  ✓ Objects with individual behaviors
  ✓ Prototypal Inheritance

• C (Java)
  ✓ Syntax
  ✓ Corporate Strategy Tax
JavaScript Early history

• May 1995, Created in ten days by Brendan Eich at Netscape: “Mocha”
• September 1995, shipped in beta of Netscape Navigator 2.0: “LiveScript”
• December 1995, Netscape 2.0b3: “JavaScript”
• August 1996, JavaScript cloned in Microsoft IE 3.0: “JScript”
• 1996-1997, Standardization ECMA-262 Ed. 1: ”ECMAScript” aka ES1
• 1999, ES3 – modern JS baseline
What is ECMAScript?

- ECMAScript is the name of the international standard that defines JavaScript
- Developed by Technical Committee 39 (TC-39) of Ecma International
- Issued as a Ecma-262 and ISO/IEC 16262
- Not part of W3C
The ECMAScript Standard Timeline


“Web 2.0” / AJAX

“ES4”  E4X  “ES4”

ES.next/“Harmony”

“ES4”  E4X  “ES4”
Interoperability is TC-39’s highest priority

- A detailed and highly prescriptive algorithmic specification
- Large, non-normative test suite for implementers

http://test262.ecmascript.org/
Things TC-39 focused on for ES 6

- Modularity
- Better Abstraction Capability
  - Better functional programming support
  - Better OO Support
- Expressiveness
- Things that nobody else can do
What Kind of Language Is JavaScript?

• Functional?
• Object-oriented?
• Class-based?
• Prototype-based?
• Permissive?
• Secure?
Some ECMAScript 6 Enhancements

- More concise and expressive syntax
- Modules and Sanding-boxing
- Class Declarations
- Block scoped declarations
- Control abstraction via iterators and generators
- Promises
- String interpolation/Internal DSL support
- Subclassable built-ins
- Binary Array Objects with Array methods
- Built-in hash Maps and Sets + weak variants.
- More built-in Math and String functions
- Improved Unicode support

https://github.com/lukehoban/es6features
Classic Smalltalk Objects

Instance Objects → Class Objects → Method Dictionaries → Compiled Methods
VSE Smalltalk Objects

Instance Objects → Method Dictionaries → Compiled Methods → Class Objects
VSE Smalltalk Objects + Per-instance methods

Instance Objects → Method Dictionaries → Compiled Methods → Class Objects
JavaScript Object Model

Prototype Link

Objects

Methods
(Function Objects)
Creating a JavaScript object

```javascript
var o = new Object;
o.counter = 0;
o.incr = function (n) {
    this.o+=n;
};
o.toString = function() {
    return this.counter.toString();
};
```

Imperatively

```javascript
var o = {
    counter: 0,
    incr: function (n) {
        this.o+=n;
    },
    toString: function() {
        return this.counter.toString();
    }
};
```

Object Literal

```javascript
var o = {
    counter: 0,
    incr (n) {
        this.o+=n;
    },
    toString () {
        return this.counter.toString();
    }
};
```

Object Literal (ES6)
Or, Define a Factory

```javascript
function CounterFactory (start) {
    return {
        counter: start,
        incr: function (n) {
            this.counter += n;
        },
        toString: function() {
            return this.counter.toString();
        }
    }
};

var o = CounterFactor(0);
```
Each object created this way has its own distinct methods.
Or, Define a Factory plus a prototype

```javascript
var counterPrototype = {
  incr: function (n) {
    this.counter += n;
  },
  toString: function() {
    return this.counter.toString();
  }
};

function CounterFactory2 (start) {
  var newObj = Object.create(counterPrototype);
  newObj.counter = start;
  return newObj;
}
```

```javascript
var o = CounterFactory2(0);
```
Instance objects share methods via prototype

```
var o1 = CounterFactory2(0)

var o2 = CounterFactory2(0)
```

- CounterPrototype
  - counter:
  - incr
  - toString

- Properties
  - counter:

-Properties
The Constructor Pattern

```javascript
function Counter(start) {
    this.counter = start;
}
Counter.prototype.incr = function(n) {this.counter += n};
Counter.prototype.toString = function() {return this.counter.toString()};
var o = new Counter(0);
```
Instance objects share methods via prototype

```javascript
var o1 = new Counter(0);
var o2 = new Counter(0);
```

**CounterPrototype**

**Methods**
- `incr`
- `toString`

**Properties**
- `counter:`
 constructor pattern with "subclassing"

//define Employee as a subclass of Person
function Employee(name,id) {
    Person.call(this, name);  //super initialize: name
    this.id = id;
}
Employee.prototype = Object.create(Person.prototype);
Employee.prototype.constructor = Employee;  //often forgotten
Employee.__proto__ = Person;  //class-side inheritance: seldom done
Employee.withId = function(id) {...};  //a "class" method
Employee.prototype.hire = function() {...};
Employee.prototype.fire = function () {...};
...
JavaScript “Constructor” Pattern

Instance Objects

Prototype Objects

Methods
(Function Objects)

Constructor Functions
VSE Smalltalk Objects

Instance Objects → Method Dictionaries → Compiled Methods → Class Objects
Classes in ES 6

//ES6: define Employee as subclass of Person

class Employee extends Person {
    constructor(name, id) {
        super(name);
        this.id = id;
    }
    hire() {...}
    fire() {...}
    static withId(id) {...}
    ...
}
Classes Today vs ES 6

//ES5 define Employee as subclass of Person

function Employee(name, id) {
    Person.call(name);
    this.id = id;
}

Employee.prototype = Object.create(Person.prototype);
Object.defineProperty(Employee.prototype, “constructor”,
    {value:Employee.prototype, enumerable:false, configurable: true});

Employee.__proto__ = Person;
Employee.withId = function (id) {...}
Employee.prototype.hire = function() {...};
Employee.prototype.fire = function () {...};

...

//ES6 define Employee as subclass of Person

class Employee extends Person {
    constructor(name, id) {
        super(name);
        this.id = id;
    }
    hire () {...}
    fire () {...}
    static withId (id) {...}
}

...

Both create the same object structure
var self = this;
var pop = peeps.filter(function(person) {
    return person.age > self.age});

Becomes:

var pop = peeps.filter(person => person.age > this.age);
But Arrow Functions Are Not Quite Smalltalk Blocks

• Return returns from the arrow function rather than the surrounding method.

• There was consider interest in: “block lambdas”:

```javascript
peeps.filter({|person|
  if (person.age>100) return;
  person.age>this.age});
```

• But too many semantics pitfalls with break/continue/return statements.
ES6 Modules

//------ lib.js ------
export const sqrt = Math.sqrt;
export function square(x) {
    return x * x;
}
export function diag(x, y) {
    return sqrt(square(x) + square(y));
}

//------ main.js ------
import { square, diag } from 'lib';
console.log(square(11)); // 121
console.log(diag(4, 3)); // 5

//------ main.js ------
import * as lib from 'lib';
console.log(lib.square(11)); // 121
console.log(lib.diag(4, 3)); // 5

or

//------ main.js ------
import * as lib from 'lib';
console.log(lib.square(11)); // 121
console.log(lib.diag(4, 3)); // 5
JavaScript Performance 2008

SunSpider (Vista SP1)
Time (ms) - Smaller is better

Milliseconds

- Chrome b1
- Safari 3.1.2
- Safari 4.0
- Opera 9.5.2
- Firefox 3.1 (no tracing)
- IE 8b2
- Firefox 3.0.1
- IE 7

JavaScript Performance 2013

SunSpider 1.0.2
lower is better

Milliseconds

How did JavaScript get fast

• Smalltalk-inspired JIT technology
• Restarted the dynamic language VM innovation that stalled when commercial Smalltalks became legacy
  ✓ Dynamic specialization based on
  ✓ Runtime monitoring/tracing
  ✓ Driving classic optimization algorithms
  ✓ Multiple execution strategies

Large teams / multi-year development projects
A Modern JS Engine

Mozilla SpiderMonkey circa 2014

https://blog.mozilla.org/luke/2014/01/14/asm-js-aot-compilation-and-startup-performance/
asm.js – C level Performance

- Subset of JavaScript that approximates a classic Von Neumann computer
- asm.js code executes identically on any JavaScript engine
- But a JS engine may recognize asm.js code and optimize for it.
- asmjs.org
C++ to JavaScript

Development Time

Clang: C++ Compiler Front-end
LLVM Optimizer
Emscripten

App Run Time

JavaScript (asm.js) source code
JavaScript Engine (asm.js aware)
Run time normalized to clang 3.2 (lower is better)

Source: Alon Zakai (@kripken)

people.mozilla.org/~lwagner/gdc-pres/gdc-2014.html
Unity game engine heading to the browser without plug-ins
WebGL and asm.js offer an alternative to the Web player.

Mozilla and Unity today announced that Unity 5, to be released later this year, will include an early access preview of a version of the 3D engine that supports WebGL and asm.js, enabling plug-in-free access to the Web.

The Unity game engine has found huge success among game developers as it can target Windows, iOS, Android, OS X, Linux, PlayStation 3, Xbox 360, and more. Unity games can also be deployed on the Web, but this function currently uses a browser plug-in, the Unity Web Player. The early access will remove the need for the plug-in. Initially, it will only support desktop Firefox and desktop Chrome, due to their performance and (in Firefox’s case) explicit support for the high performance asm.js subset.

Mozilla and Epic Games have showed the power of the Web as a platform for gaming by porting Unreal Engine 3 to the Web and showcasing Epic Citadel, using asm.js, a supercharged subset of JavaScript pioneered by Mozilla. In less than 12 months, optimizations have increased the performance of Web applications using asm.js from 40% to within 67% of native, and we expect it to get even faster. This performance opens up new opportunities for giving users an astonishing and delightful experience, from within their choice of Web browser. Any modern browser can run asm.js content, but specific optimizations currently present only in Firefox, ensure the most consistent and smooth experience.
Why not a web bytecode engine?
asm.js code is just YAIR
(Yet Another Intermediate Representation)

```javascript
function strlen(ptr) {
  ptr = ptr|0;
  var curr = 0;
  curr = ptr;
  while (MEM8[curr]|0 != 0) {
    curr = (curr + 1)|0;
  }
  return (curr - ptr)|0;
}
```
ECMAScript Resources

The Official ECMAScript 5.1 Specification (HTML)
http://www.ecma-international.org/ecma-262/5.1/

ES6 Specification Drafts

ES6 Feature Summary
https://github.com/lukehoban/es6features

ES6 translators and tools
https://github.com/addyosmani/es6-tools

The TC-39 ECMAScript Design Discussion Mail List
https://mail.mozilla.org/listinfo/es-discuss

Test262: The Official ECMAScript Implementation Test Suite
http://test262.ecmascript.org/

Please report bugs
http://bugs.ecmascript.org