mozilla
CORPORATION

JavaScript at Ten Years

Brendan Eich

brendan@mozilla.org

mozilla CORPORATION

Triptych

- The salad days, green in judgment
- Maturity, standardization, stagnation
- AJAX, the JS renaissance, and beyond

Netscape, Spring 1995

- Netscape > 90% browser market share
- MS buy-out attempt in late 1994 rebuffed
- Threat to Windows made explicit by Java
- “Worse is Better!” - marca channeling rpg
- “We are doomed!” - engineering refrain
- Very narrow window in which to innovate

The Web Platform Play

- Opportunity to do “HTML scripting language”
- Netscape/Sun “Java deal” cast long shadow
- Argument by code demonstration necessary
- I hacked the JS prototype in ~1 week in May
  - And it showed! Mistakes were frozen early
- Rest of year spent embedding in browser

Design Goals

- Make it easy to copy/paste snippets of code
- Tolerate “minor” errors (missing semicolons)
- Simplified onclick, onmousedown, etc., event handling, inspired by HyperCard
- Pick a few hard-working, powerful primitives
  - First class functions for procedural abstraction
  - Objects everywhere, prototype-based
- Leave all else out!

Marketing, sigh

- Netscape management fretted: “Why two programming languages?”
- Answer: division of labor, specialization
- Java for high-priced components/widgets
- “Mocha” for mass market web designers
- “Object-based”, if not object-oriented

More Marketing

- Then they changed the name to “LiveScript”
- Finally, to “JavaScript” in late 1995
- Mostly a marketing scam
- Engineering pressure to “be like Java” did cause us to follow Java into some dead ends (Date inherited y2k bugs from java.util.Date!)
- Confusion ever since

JavaScript in Two Slides

- Objects map strings to values (properties):

```
var obj = new Object;
obj[“prop”] = 42;       => obj.prop
obj[“0”] = “hello”;     => obj[0]
```

- Functions are first-class objects:

```
function fact(n) {
  return (n <= 2) ? n : n * fact(n-1);
}
fact.desc = “Factorial function”;
```

JS in Two Slides (2)

- So methods are function-valued properties:

```
obj.frob = function (n) {
    this.prop += n;
};
obj.frob(6);        => obj.prop == 48
```

- Permissiveness throughout. Oops.

```
grob = obj.frob;   => var not necessary
grob(6);           => undefined + 6 == NaN
prop = “hello”;    => reset global prop
grob(6);           => prop == “hello6”
```

Full Lexical Closures

```
function Y(g) {
  return function (f) {return f(f);}(
    function (f) {return g(function (x) {
      return f(f)(x);
    });
  });
}
var fact = Y(function (fact) {
  return function (n) {
    return (n <= 2) ? n : n * fact(n-1);
  }
});
alert(fact(5));  => 120
```

JS Is Self-ish

- All functions can construct:

```
function Car(make, model) {
  this.make = make, this.model = model;
}
myCar = new Car("Porsche", "Boxster");
```

- All functions have a prototype property:

```
Car.prototype.color = "black";    => default color
old = new Car("Ford", "T");       => black Model T
myCar.color = "silver";           => my override
```

- Powerful when combined with closures

Where Did All This Lead?

- Web in early 1996 was text+images
  - Yahoo! home page
- JS caught on like a bad cold (per plan!)
- Annoyances (now suppressed by good browsers) proliferated
- My colleagues made a home page for me

Things Got Better

- We've come a long way
- Google Maps
- Yahoo! webmail (based on oddpost.com)
- Zimbra, another "AJAX" webmail app
- The "X" in AJAX is for "XML", specifically XMLHttpRequest, which MS added to IE when they gave Java the boot – ironic!

What Changed?

- JS and the “DOM” stabilized
- Up and coming browsers matched MS IE’s platform features (e.g., XMLHttpRequest)
- Moore’s Law compounded
- Hackers discovered JS’s FP and Self-ish OOP features
- And good hackers like those features

Hackers Like JS?

- It's easy to extend user-defined objects
  - ruby.js for Ruby generic method emulation
- Or built-in objects (Prototype example):

```
Function.prototype.bind = function(object) {
  var method = this;
  return function() {
    method.apply(object, arguments);
  }
}
```

mozilla CORPORATION

Did JS Matter?

- Wouldn't this have happened with any winner-take-all browser-based language?
  - Tcl, Perl, Python, Java, VBScript(!)
  - In 1995, not good choices per design goals
  - Event handlers in HTML DOM + JS "easy to use" design goal imply first class functions
  - Hard to work around lack of closures (e.g. using Java anonymous inner classes)

Closures > Objects

- JS lacks information hiding *a la* Java
- But closures save us again:

```
Function Car(make, model) {
  this.make  = function() {return make;}
  this.model = function() {return model;}
}
var myCar = new Car("Audi", "A8L");
alert(myCar.make());
// No way to subvert make or model
```

- Private static members can be done similarly

What's Not To Like?

- Aping Java's use of + for string concatenation
  - Compatibility is king, too late to fix this one
- Permissiveness:
  - Can call with too few or too many arguments
  - Types convert freely, e.g., "1.0" == 1 && "1" == 1 but "1.0" != "1" (use === instead)
  - Information hiding requires unfamiliar closures
- Lack of a standard library mechanism

mozilla CORPORATION

Where To Next?

- ECMA standardized JavaScript in 1997
- Revised for ISO in 1998
- Revised again in 1999 for major features that missed v1 (what was in Netscape 2-3)
- E4X extends JS with XML first class object subtype and literal syntax
- Edition 4 of the ECMA-262 standard under way again, after lengthy hiatus

ECMA Edition 4

- Not deprecating prototypes or dynamic types
- Will support writing the language in itself
- Express property get/set methods, declare property attributes, other missing MOP stuff
- Classes introduced as Self-ish traits objects
- Namespaces for API versioning
- Packages for standard library mechanism

mozilla CORPORATION

Conclusion

- If in a rush, target your audience and simplify
- Pick the right primitives, support extensions
- The right primitives for event handling include first class functions and closures
- Proof: languages such as C# start with Java and grow such features (delegates, lambdas)
- Don’t let Marketing name your language