JavaOne℠

Functional and Object-Oriented Programming in the JavaScript™ Programming Language

Roberto Chinnici
Sun Microsystems, Inc.

The World is Changing

JavaScript has come of age

New ECMAScript 5 standard coming up

JavaScript for offline apps (e.g. Google Gears)

JavaScript on the server

Major advances on JavaScript virtual machines

Frameworks fast, stable, more popular than ever

What is JavaScript, Really?

A functional core, inspired by Scheme

A prototype-based object model

Very dynamic language

- (Almost) any property of any object can be redefined

Object notation (JSON) now ubiquitous

What About the Bad Parts?

Forget them!

Remaining goodness overwhelms the bad

Potential to do great things in JavaScript

Lots of people pushing the envelope every day

Contents

Functional JS

Object-oriented JS

How they blend

What they enable

What's missing

Examples from the real world:

- JQuery / ProtoScript / Cappuccino / Caja

Functional JavaScript

Functions are Closures

```
function createCounter() {
    var i = 0;
    return function() {
        return ++i;
    }
}

var counter = createCounter();
```

- counter() ⇒ 1
- counter() ⇒ 2
- counter() ⇒ 3

Function are First-Class

```
function triple(x) { return x * 3; }

var addTwo = function(y) { return y + 2; }

function compose(f, g) {
    return function(z) {
        return f(g(z));
    }
}
```

- `compose(addTwo, triple)(5)` ⇒ 17

One Scope per Function

Not all {...} introduce a scope (unlike in Java)

```
function verbose() {
    var x = 1;
    var y = 3;
    var z = 4;

    function f() {
        var y = 2; // masks outer y
        return x + y + z;
    }

    return f() + y;
}
```

- `(verbose())()` ⇒ 7 // not 8

Let Statements in JavaScript 1.7

Scopes without functions

```
function verbose() {
    var x = 1;
    var y = 3;
    var z = 4;
    var t;

    let (y = 2) {
        t = x + y + z;
    }

    return t + y;
}
```

- `(verbose())() ⇒ 7 // not 8`

What About “document” or “window”?

Top-level scope (“global”), visible everywhere

If a variable hasn't been declared, its global

- Unfortunate choice!
- Variables are global by default

Use JSLint (http://www.jslint.com)

- Catches lots and lots of common problems

Function Application

The bread-and-butter of functional programming

You can do one of two things with a function:

- treat it as a value (e.g. to create new functions, or to pass it on to some other piece of code)
- apply it

Only application DOES anything

Three Ways to Apply a Function

```
function timesTwo(x) { return x*2; }
```

- `timesTwo(4)` ⇒ 8
- `timesTwo.call(undefined, 4)` ⇒ 8
- `timesTwo.apply(undefined, [4])` ⇒ 8

```
function sum(x, y) { return x+y; }
```

- `sum(3,5)` ⇒ 8
- `sum.call(undefined, 3, 5)` ⇒ 8
- `sum.apply(undefined, [3, 5])` ⇒ 8

Note: you can replace "undefined" by any value here

Variadic Functions

The special `arguments` variable

```
function sum() {
    var total = 0;
    for (var i = 0; i < arguments.length; ++i) {
        total += arguments[i];
    }
    return total;
}
```

- `sum(3, 5, 7, 9)` ⇒ 24
- `sum.call(undefined, 3, 5, 7, 9)` ⇒ 24
- `sum.apply(undefined, [3, 5, 7, 9])` ⇒ 24

`arguments` is an array (almost)

Recursive Anonymous Functions

```
var mysterious = function(n) {
    return n <= 1 ? 1 : n*arguments.callee(n-1);
}
```

- `mysterious(5)` ⇒ `120`

Simulating Control Structures

Anonymous functions (lambdas) as blocks

```
if (n % 2 == 0 ) {
   handleEven(n);
}
else {
   handleOdd(n);
}

function ifFn(testFn, thenFn, elseFn) {
  return testFn() ? thenFn() : elseFn();
}

ifFn(function() { return n % 2 == 0; },
     function() { handleEven(n); },
     function() { handleOdd(n); });
```

Issue: Non-local Returns

Naïve way is broken

```
for (x in mylist) {
   if (found(x)) return x;
}
return undefined;

function for_in(aList, eachFn) {
  for (var x in aList) {
    eachFn.call(undefined, x);
  }
}

for_in(mylist, function(x) {
   if (found(x)) return x; // doesn't work!
});
return undefined;
```

Controlling the Control Flow

Normal flow is function call / return

Need ability to return to an outer scope

Three ways to do it in the literature:

- jump to a dynamic label (tagbody/go in Lisp)
- structured exceptions
- continuations (Scheme)

JavaScript only has exceptions

Using Exceptions For Control Flow

Example: Prototype 1.6.0

```
var $break = { };

var Enumerable = {
  each: function(iterator, context) {
    var index = 0;
    try {
      this._each(function(value) {
        iterator.call(context, value, index++);
      });
    } catch (e) {
      if (e != $break) throw e;
    }
    return this;
  },
…
}
```

Fixing for_in example

Use appropriate higher-order function

```
for (x in mylist) {
   if (found(x)) return x;
}
return undefined;

return mylist.include(found);

// edited from Prototype 1.6.0
include: function(object) {
  var found = false;
  this.each(function(value) {
    if (value == object) {
      found = true;
      throw $break;
    }
  });
  return found;
}
```

Functional.js Library

http://osteele.com/sources/javascript/functional/

Extensive library of functional primitives

Basics: map, select, reduce, foldr, foldl…

- `Functional.map(function(x){return x*2}, [1,2,3])` ⇒ `[2,4,6]`
- `F.select(function(x){return x%2==0}, [2,3,4])` ⇒ `[2,4]`

Selection/predicates: some, every, and, or, not…

- `F.every(function(x){return x%2==0}, [2,3,4])` ⇒ `false`

Higher-order functions: bind, curry, sequence…

Compact string-based syntax for functions

- `"_+2".lambda()` ⇒ `function(x){return x+2}`

Example: select function

From Functional.js 1.0.2

```
Functional.select = function(fn, sequence, object) {
    fn = Function.toFunction(fn);
    var len = sequence.length,
        result = [];
    for (var i = 0; i < len; i++) {
        var x = sequence[i];
        fn.apply(object, [x, i]) && result.push(x);
    }
    return result;
}
```

Why Functional Programming?

Loop-free programming

- The intent of a loop is not easy to read

Instead, use explicit operations: map, select…

- Intent apparent from the name of the function

Variable-free programming (almost)

- Single assignment

Functions always introduce their own scope

- Fewer clashes

Use established patterns, like generate/filter/map

One (Partial) Alternative

Array comprehensions in JavaScript 1.7

- `[x*2 for each (x in [1,2,3])]` ⇒ `[2,4,6]`
- `[x*2 for each (x in [1,2,3]) if x%2==1]` ⇒ `[2,6]`

Useful to initialize arrays

In reality, tied to generators

- `[x*x for each (x in range(1,6))]` ⇒ `[1,4,9,16,25]`

`range(a,b)` is a generator producing `a,...,b-1`

Kind of equivalent to map/select combination

Digression: How To Play With This Stuff

Using a JavaScript shell

Rhino

```
$ java -jar js.jar
js> load("to-function.js")
js> load("functional.js")
```

Google V8 (from svn)

```
$ scons sample=shell
$ ./shell
> load("to-function.js")
> load("functional.js")
```

Object-oriented JavaScript

Prototype-oriented Programming

Objects without classes

JavaScript objects exist on their own account

Objects have named properties

(Almost) every object has a prototype

Prototypes form chains

Property lookups follow prototype chain

`Object.prototype` is the default “root”

- `Object.prototype.prototype ⇒ undefined`

Methods

Methods are function-valued properties

Special notation to make calling them easier

- `obj.m(x, y, z)` ⇒ `(obj[m]).apply(obj, [x, y, z])`

The first argument to call/apply is passed as `this`

```
var obj = {};
obj.m = function(x) { return x + this.y; }
obj.y = 3;
```

- `obj.m(5)` ⇒ `8`

`this` is a special variable, like `arguments`

There are no other special variables!

How Do I Set the Prototype of an Object?

JavaScript 101 question

This DOES NOT work:

```
var obj = {};
obj.prototype = {compute: function(x,y) { … }};
```

There is no **`prototype`** property of an object

- internal [[Prototype]] property is inaccessible

So what was **`Object.prototype`**?

- **`Object`** is a function!
- So **`Object.prototype`** is a property of a function object !!

Regrettably... It's All Backwards!

Desire to preserve the **`new`** keyword

Prototypes are set on functions

```
function fn() { this.x = 4; }
fn.prototype = { y: 8 };
var obj = new fn();
```

- `obj.x ⇒ 4`
- `obj.y ⇒ 8`

The function that created an object becomes its **`constructor`**

- `obj.constructor == fn ⇒ true`

How To Retain Your Sanity

Forget you ever heard about constructors

Forget about classes

Use Douglas Crockford's `object` function

- `object(x)` ⇒ `new object whose prototype is x`

  Will be in ECMAScript 5 as `Object.create`

Use `extend` to add properties to a prototype

- `extend(x, {foo: 2, bar: 5})`
- `x.foo` ⇒ `2`

  In Prototype, Dojo, and many other libraries

Implementing `Object.create`

```
Object.create = function(o) {
    function F() {}
    F.prototype = o;
    return new F();
}
```

See http://javascript.crockford.com/prototypal.html

Example: bind function (higher-order)

From Functional.js 1.0.2

```
var f2 = f1.bind(foo, bar)
```

- `f2(x, y, z)` ⇒ `f1.apply(foo, [bar, x, y, z])`
  ⇒ `foo.f1(bar, x, y, z)`

```
Function.prototype.bind = function(object) {
    var fn = this;
    var args = Array.slice(arguments, 1);
    return function() {
        return fn.apply(object,
           args.concat(Array.slice(arguments, 0)));
    }
}
```

Some cool stuff

Functions + Objects = Power

Language features build on each other

First-class functions give units of behavior

- Functions compose/chain naturally

Prototype-based object model is very flexible

- Object can be created on the fly, in any shape

What can you do with it?

- Domain Specific Languages
- Language extensions (include radical ones)
- Generic target for code generation

JQuery
A DSL for DOM manipulation

Select nodes via a CSS-like expression

```
$("a")
$("tr:first")
$("input:not(:checked)")
```

Refine or expand the nodeset as you go

```
$("#container").children()
$("div").find("p")
```

Operate on the nodeset

```
$("#status").find("p").addClass("hilite")
```

JQuery Example

It's all objects + functions

```
$(document).ready(function(){
 $("a")
    .filter(".clickme")
      .click(function(){
        alert("You are now leaving the site.");
      })
    .end()
    .filter(".hideme")
      .click(function(){
        $(this).hide();
        return false;
      })
    .end();
});
```

`this` Hits Back

There's only one `this` in scope at any time

Look for the directly enclosing `function`

```
$(document).ready(function() {
  // here "this" is a DOM document
  alert(this.domain);

  $("input")
      .blur(function () {
              // here "this" is a DOM node
              $(this).next("span")
                     .css('display','inline')
                     .fadeOut(1000);
          });
});
```

Protoscript
Data as code

```
$proto('img#avatar', {
  Click: {
    onClick: {
      Animate: {
        height: {to: 100},
        width : {to: 100},
        duration: 0.5
      },
      onComplete: {
        Fade: {
          opacity: {to: 0}
        }
      }
    }
  }
});
```

Adapted from code on http://docs.protoscript.com/Main_Page

Cappuccino
Rewriting Objective-J to JavaScript

```
var myPerson = [[Person alloc] init];
[myPerson setName: "John"];
[myPerson setJobTitle: "Founder" company: "280 North"];
```

becomes:

```
var myPerson = objj_msgSend(
    objj_msgSend(Person, "alloc"), "init");
objj_msgSend(myPerson, "setName:", "John");
objj_msgSend(myPerson, "setJobTitle:company:",
             "Founder", "280 North");
```

Code sample from http://cappuccino.org/learn/tutorials/objective-j-tutorial.php

Compiling Objective-J Classes

Cappuccino defines an object model from scratch!

```
@implementation Person : CPObject
{
CPString name;
}
…
```

becomes:

```
var the_class = objj_allocateClassPair(CPObject, "Person"),
     meta_class = the_class.isa;
class_addIvars(the_class, [new objj_ivar("name")]);
objj_registerClassPair(the_class);
objj_addClassForBundle(the_class,
       objj_getBundleWithPath(OBJJ_CURRENT_BUNDLE.path));
var instance_methods = [];
…
```

Google Caja

A secure JavaScript subset

Capability-oriented model to sandbox untrusted scripts embedded on a web page

Implemented by code rewriting

Must prevent access to global state

Common shared objects are frozen

Constructors are frozen at first use

Simple functions are safe:

- don't mention `this`
- only access lexically scoped variables (no globals)

Example: A Point "Class" in Caja

```
function Point(x, y) {
    this.x_ = x;
    this.y_ = y;
}

caja.def(Point, Object, {
   toString: function() {  return …; },
   getX: function() { return this.x_ ; },
   getY: function() { return this.y_ ; }
);

function WobblyPoint(x, y) {
    WobblyPoint.super(this, x, y);
}

caja.def(WobblyPoint, Point, {
…
}
```

Conclusions

A Major Shift is Underway

JavaScript as fertile ground for innovation

Constant performance improvements

Expanding beyond the browser

Explosion in number of libraries

Functional and object-oriented aspects mesh together to enable DSLs and more

Multiple, innovative language extensions using clever rewriting techniques

Thank You

Roberto Chinnici
roberto.chinnici@sun.com