JavaScript Functions : The Good Parts - Idioms for Encapsulation and Inheritance

Scott Bale, BJC HealthCare

@scottbale

scott@balehaus.org

JavaScript Functions : The Good Parts - Idioms for Encapsulation and Inheritance

or

A Thinly-Veiled Ripoff of Douglas Crockford's JavaScript: The Good Parts Chapters 4 and 5

One-Slide Intro

Pretty much anything interesting in JavaScript happens with functions

Functions are the *only* way to have nested scope

Constructor functions (with 'new' operator) are the *only* way to set an object's prototype

Whirlwind Review - Objects

Let's make an object

```
var COINS = new Object();
COINS.nickel = 5;
COINS.dime = 10;
COINS.quarter = 25;
COINS.isCoin = function(cents){
    return [this.NICKEL, this.DIME,
     this.QUARTER].contains(cents);
};
```

Let's make an object - JSON

```
var COINS = {
    NICKEL : 5,
    DIME : 10,
    QUARTER : 25,
    isCoin : function(cents){
        return [this.NICKEL, this.DIME,
         this.QUARTER].contains(cents);
    }
};
```

Accessing object properties

```
var COINS = {
    NICKEL : 5,
    DIME : 10,
    QUARTER : 25,
    isCoin : function(cents){
        return [this.NICKEL, this.DIME,
         this.QUARTER].contains(cents);
    }
};

result = COINS.NICKEL;
```

Accessing object properties

```
var COINS = {
    NICKEL : 5,
    DIME : 10,
    QUARTER : 25,
    isCoin : function(cents){
        return [this.NICKEL, this.DIME,
         this.QUARTER].contains(cents);
    }
};

result = COINS["NICKEL"];
```

Iterating an object's properties

```
var COINS = {
    NICKEL : 5,
    DIME : 10,
    QUARTER : 25,
    isCoin : function(cents){
        return [this.NICKEL, this.DIME,
         this.QUARTER].contains(cents);
    }
};
for (var name in COINS){
    CORE.out(name + ':' + COINS[name]);
}
```

Cleaner Iteration - hasOwnProperty() and typeof

```
var COINS = {
    NICKEL : 5,
    DIME : 10,
    QUARTER : 25,
    isCoin : function(cents){
        return [this.NICKEL, this.DIME,
         this.QUARTER].contains(cents);
    }
};
for (var name in COINS){
    if (COINS.hasOwnProperty(name)
     && typeof COINS[name] !== 'function'){
        CORE.out(name + ':' + COINS[name]);
    }
}
```

Functions

Function Statement

```
var coinReturn = [25, 25, 10];
var coins = [];

function isSufficientFunds(purchasePrice){
    var funds = 0;
    coinReturn.each(function(coin){
        funds+=coin;
    });
    return (funds >= purchasePrice);
}

result = isSufficientFunds(60);
```

Function Expression

```
var coinReturn = [10, 5, 25];
var coins = [];

var isSufficientFunds = function(purPrice){
    var funds = 0;
    coinReturn.each(function(coin){
        funds+=coin;
    });
    return (funds >= purPrice);
};

result = isSufficientFunds(75);
```

Functions as parameters

```
var coins = [];

var makeChange = function(changeDue){
    coins.sort(function(a,b){
        return b-a;
    });

    // some other stuff...
};
```

Functions are objects, too

```
var foo = function(){return false;};
foo.bar = function(){return true;};
foo.baz = 3;

result = foo();
```

Functions are objects, too

```
var foo = function(){return false;};
foo.bar = function(){return true;};
foo.baz = 3;

result = foo.bar();
```

Function Invocation Patterns

What is 'this'?

```
function(){
    return this.foo;
}
```

Four function invocation patterns

function

method

constructor

apply

'this' detector function

```
var thisDetector = function (){
    if (this === thisDetector){
        result = '\'this\' is me!';
    } else {
        result = '\'this\' is ' + this;
    }
};
```

Function invocation pattern

```
var thisDetector = function (){
    if (this === thisDetector){
        result = '\'this\' is me!';
    } else {
        result = '\'this\' is ' + this;
    }
};
thisDetector();
```

Function invocation pattern

```
var containingFunction = function(){

    var thisDetector = function (){
        if (this === thisDetector){
            result = '\'this\' is me!';
        } else {
            result = '\'this\' is ' + this;
        }
    };
    thisDetector();
}();
```

Method invocation pattern

```
var thisDetector = function (){
    if (this === thisDetector){
        result = '\'this\' is me!';
    } else {
        result = '\'this\' is ' + this;
    }
};
var testObject = {
    testMe:thisDetector,
    toString:function(){return "testObject"}
};
testObject.testMe();
```

Constructor invocation pattern

```
var thisDetector = function (){
    if (this === thisDetector){
        result = '\'this\' is me!';
    } else {
        result = '\'this\' is ' + this;
    }
};
thisDetector.prototype.toString = function(){
    return "newly constructed object";
};
new thisDetector();
```

'apply()' invocation pattern

```
var thisDetector = function (){
    if (this === thisDetector){
        result = '\'this\' is me!';
    } else {
        result = '\'this\' is ' + this;
    }
};

thisDetector.apply(thisDetector, []);
```

Apply different context

```
var thisDetector = function (){
    if (this === thisDetector){
        result = '\'this\' is me!';
    } else {
        result = '\'this\' is ' + this;
    }
};

thisDetector.apply(
  {toString:function(){return 'foo';}}, []);
```

'apply()' with arguments

```
var argCounter = function (){
    result = arguments.length;
};

argCounter.apply({}, [false, 3, {}]);
```

Encapsulation Idioms

Why Encapsulation?

No linker, only a single global object

No native namespacing or visibility modifiers

Anything can modify most anything

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" l
<head>
<meta http-equiv="Content-Type" content="text/html; charse
<title>foo</title>
<script type="text/javascript" src="/js/jquery-1.4.min.js"
<script type="text/javascript" src="/js/jquery.cycle.all.j
<script type="text/javascript" src="/js/jquery-print.js"><
<script type="text/javascript" src="/js/jquery.batchImageL
<script type="text/javascript" src="/js/jquery.doubletap-(
<script type="text/javascript" src="/js/fg.menu.js"></scri
<script type="text/javascript" src="/js/showoff.js"></scri
<script type="text/javascript" src="/js/jTypeWriter.js"> <
<script type="text/javascript" src="/js/sh_main.min.js"></
<script type="text/javascript" src="/js/core.js"></script>
<script type="text/javascript" src="/js/showoffcore.js"></
...
```

Encapsulation Techniques

simple namespacing

private state with function closures

Module pattern combines these two

Namespacing

Avoid global vars

```
var COINS = {
     NICKEL : 5,
     DIME : 10,
     QUARTER : 25,
     isCoin : function(cents){
          return [this.NICKEL, this.DIME,
           this.QUARTER].contains(cents);
     }
};

result = COINS.NICKEL;
```

Function Closures

```
var coinReturn = [25, 25, 10];

function isSufficientFunds(purchasePrice){
    var funds = 0;
    coinReturn.each(function(coin){
        funds+=coin;
    });
    return (funds >= purchasePrice);
}
```

```
var func = function(){
    var coinReturn = [25, 25, 10];

    function isSufficientFunds(purchasePrice)
        var funds = 0;
        coinReturn.each(function(coin){
            funds+=coin;
        });
        return (funds >= purchasePrice);
    }
    return isSufficientFunds;
}();

CORE.out(func(20));
CORE.out(func.coinReturn);
```

A Module...

```
var ACM = function(){
    var coinReturn = [];
    var coins = [];

    var doPurchase = function(price){
        if (ifSufficientFunds(price)){
            // ...
        }
    };
    //...
    return {
        deposit : function(coin){
            coinReturn.push(coin);
        },
        purchase : function(price){
            return doPurchase(price);
        }
        //...
    };
}();
```

The Module Pattern

```
var MODULE = function(){
    var private state = //...

    var privateMethod = function(arg){
        // ...
    };

    return {
        publicMethod : function(arg){
            privateMethod(arg);
        },
    };
}();
```

Slight Variation

```
var MODULE = function(anotherModule){
    var private state = //...

    var privateMethod = function(arg){
        anotherModule.doSomething();
        // ...
    };

    return {
        publicMethod : function(arg){
            privateMethod(arg);
        },
    };
}(ANOTHER_MODULE);
```

Another Variation

```
(function(module){
    var private state = //...

    var privateMethod = function(arg){
        // ...
    };

    module.publicMethod = function(arg){
        privateMethod(arg);
    };
}(ANOTHER_MODULE));
```

My Core Module

```
var CORE = function(){
    // stuff omitted...
    return {
        out : function(output){
            //implemented by env
        },
        require : function(toImport){
            this.out(
             "'require' not implemented!");
        }
    };
}();
```

Core module

modified for Mozilla Rhino

```
(function(CORE){
    // ...
    CORE.require = function(toImport){
        // load from src or testsrc
    };
    CORE.out = function(output){
        // Rhino-specific!
        print('\n'+output);
    };

    return CORE;
}(CORE));
```

Core module

modified for browser

```
(function(CORE){

    CORE.require = function(toImport){
        //nothing to do
    };
    CORE.out = function(output){
        document.write('<br>'+output+'</br>')
    };

    return CORE;
}(CORE));

CORE.out('some sort of browser');
```

Key Point

Functions are the *only* means of nesting scope

Prototype

JavaScript Prototypes

All objects have a *hidden* link to a prototype

If not specified, defaults to the prototype belonging to Object constructor

All functions have a ".prototype" property, which is used to set the prototype link of any objects created by calling that function as a constructor

built-in constructors

```
CORE.out(typeof Object);
CORE.out(typeof Object.prototype);
CORE.out(typeof Function);
CORE.out(typeof String);
CORE.out(typeof Array);
CORE.out(typeof RegExp);
CORE.out(typeof Number);
```

prototype

changes to prototype inherited by all instances

```
delete Object.prototype.foo;

var obj = new Object();

CORE.out(obj.foo);

Object.prototype.foo = function(){
    return "I am an object.";
};
CORE.out(obj.foo());
```

prototype

modifying 'obj' does not affect prototype

```
Object.prototype.foo = 2112;

var obj = new Object();
CORE.out(obj.foo);

obj.foo = 42;
CORE.out(obj.foo);
CORE.out(Object.prototype.foo);

delete obj.foo;
CORE.out(obj.foo);
```

Extending the Language

```
Function.prototype.method
                    = function (name, func) {
     if (!this.prototype[name]){
          this.prototype[name] = func;
          return this;
     }
};
Array.method('each', function(f, index){
     for (var i=0; i<this.length; i++){
          f(this[i], i);
     }
});

[2,3,5].each(function(item){CORE.out(item);})
```

Inheritance Idioms

Pseudoclassical Inheritance

```
// a constructor aka pseudo-class
var FreeACM = function() {
    this.coinReturn = [];
};
FreeACM.prototype.purchase = function(price)
    return true;
};

var acm = new FreeACM();
CORE.out(acm.purchase(85));
```

"Subclass"

```
var FreeACM = function() {
    this.coinReturn = [];
};
FreeACM.prototype.purchase = function(price)
    return true;
};

var PayACM = function() {};
PayACM.prototype = new FreeACM();
PayACM.prototype.purchase = function(price) {
    return this.coinReturn.length != 0;
}

var acm = new PayACM();
CORE.out(acm.purchase(85));
```

Prototype.js

```
var FreeACM = Class.create({
  initialize: function() {
    this.coinReturn = [];
  },
  purchase: function(price) {
    return true;
  }
});

var PayACM = Class.create(FreeACM, {
  purchase: function(price) {
    return this.coinReturn.length != 0;
  }
});
```

Key Point

An object's hidden prototype link can *only* be set by using the new operator

Prototypal Inheritance

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

Prototypal Inheritance

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

```
var freeACM = {
    coinReturn : [],
    purchase : function(price) {
        return true;
    }
}
var payACM = Object.create(freeACM);
payACM.purchase = function(price) {
    return this.coinReturn.length != 0;
}

CORE.out(freeACM.purchase(85));
CORE.out(payACM.purchase(85));
```

Functional Inheritance

Combine prototypal inheritance with encapsulation

```
var constructor = function(spec,
                           secretStuff){
    secretStuff = secretStuff || {};
    var a, b; // more secret stuff

    var publicMethod = function(){
        // accesses secret stuff
    };

    var obj = Object.create(spec);
    obj.publicMethod = publicMethod;
    return obj;
};
```

```
var freeACM = function(coinReturn){
    doPurchase = function(price) {
        return true;
    }
    return {
        coinReturn : coinReturn,
        purchase : doPurchase
    }
};
var payACM = function(coinReturn){
    var doPurchase = function(price) {
        return this.coinReturn.length != 0;
    };
    var obj = Object.create(freeACM(coinRetur
    obj.purchase = doPurchase;
    obj.coinReturn = coinReturn;
    return obj;
};
CORE.out(freeACM([]).purchase(85));
CORE.out(payACM([]).purchase(85));
```

The Wrap-Up

jQuery

```
$('div.clickable').click(function(){
    $(this).attr('class', 'clicked');
});
```

jQuery core.js module

```
(function() {

    // Define a local copy of jQuery
    var jQuery = function( selector, context
            // The jQuery object is just the
            // init constructor 'enhanced'
            return new jQuery.fn.init(
                        selector, context );
        },

    // ...900+ lines later

    // Expose jQuery to the global object
    window.jQuery = window.$ = jQuery;
})();
```

Links

Douglas Crockford

JavaScript: The Good Parts

The Module Pattern

slides by showoff

http://github.com/scottbale