DECLARATIVE OBJECT AND CLASS ABSTRACTIONS

BASED UPON EXTENDED OBJECT INITIALISERS

Allen Wirfs-Brock

Mozilla

March 23, 2011

Overall Goal

- Enable declarative definition of more object based abstractions
- Build off of Object Initialiser Syntax

- Easier for humans to read, write, understand and maintain
- Easier to mechanically analyze
- Easier to optimize

Object Literal Meta Properties

ES5

```
var obj = Object.preventExtensions(Object.create( someObject,
    { prop1: {value: expr, writable: true, enumerable: true,
              configurable: true},
      prop2: {get: function(){return computeSomeValue()},
              set: undefined, enumerable: true, configurable: true}
}));
```

Harmony

```
var obj = {<proto: someObject, closed>,
     prop1:  false,
     get prop2 function(){return computeSomeValue()},
};
```

Meta properties: proto:, closed, sealed, frozen, …

Array Literal Literal Meta Properties

Harmony

```
var arr= [<proto: myProto>, “a”,2, false];
```

Creates an object with length property and Array’s [[DefineOwnProperty]]

Object Literal Method Properties

ES5

```
var obj = Object.create( Object.prototype,
    { a: {value: 1, writable: true, enumerable: true, configurable: true},
      b: {value: 2, writable: true, enumerable: true, configurable: true},
      toString: {value: function () {return “”+(this.a+this.b)} confugurable: true},
});
```

Harmony

```
var obj = {
     a:  1,
     b:  2,
     method  toString () {return “”+(this.a+this.b)}
};
```

Object Literal Property Modifiers

Orthogonal set of modifiers generate all property attribute combinations

Harmony

```
return {
 p7: 7,                  //configurable: true, enumerable: true, writable: true
 p6 const: 6,            //configurable: true, enumerable: true, writable: false
 var p5: 5,              //configurable: true, enumerable: false,writable: true
 var p4 const: 4,        //configurable: true, enumerable: false,writable: false
 sealed p3: 3,           //configurable: false,enumerable: true, writable: true
 sealed p2 const: 2,     //configurable: false,enumerable: true, writable: false
 sealed var p1: 1,       //configurable: false,enumerable: false,writable: true
 sealed var p0 const: 0,//configurable: false,enumerable: false,writable: false

 method m4 () {},        //configurable: true, enumerable: false,writable: false
 sealed method m0 () {},//configurable: false,enumerable: false,writable: false

 set a3 (v) {},          //configurable: true, enumerable: true
 var get a2 () {},       //configurable: true, enumerable: false
 sealed put a1 (v) {},   //configurable: false,enumerable: true
 sealed var get a2 () {}//configurable: false,enumerable: false
}
```

Object Literal Initialization Blocks

ES5

```
var obj = { a: 1, b: 2 };
ObjectMonitor.newObjectToMonitor(obj);
```

Harmony

```
var obj = {
   a: 1,
   b: 2,
   {ObjectMonitor.newObjectToMonitor(this);}
};
```

Private Names in Initialisers

Unique lexically scoped non-textual property names

Harmony

```
var obj;
{
   private a;  //privates go at top
   obj = {a:1, b:2};
   print(obj.a)   //1
}
print(obj.a)   //undefined
print(obj.b)   //2
```

Other keyword possibilities:

```
unique a;
define a;   //☺☺
```

Declarative Class Initialisers

- Declarative definitions of constructor/prototype/instance triads based upon “class model” used by Chapter 15
- Build off of exended Object Initialiser Syntax

- Both statement and expression forms:

Harmony

```
class A { };

let B = class {};
```

Declarative Classes: Basic Triad

Harmony

```
class C { };
```

ES5

```
function c() {};
Object.defineProperty(c,'prototype',
    {writable: false, configurable: false}};
Object.defineProperty(c.prototype,'constructor
',
    {writable: false, configurable: false}};
```

Declarative Classes: Constructor body

Harmony

```
class c {
   new (a,b) {// an object initialiser
       a : a,
       b : b,
       {this.x = somebody(this)}
   }
};
```

ES5

```
function c(a,b) {
  Object.defineProperty(this,"a",
     {value: a,enumerable: true, writable: true, configurable: true});
  Object.defineProperty(this,"b",
     {value: a,enumerable: true, writable: true, configurable: true});
  (function () {this.x = somebody(this)}).call(this);
};
Object.defineProperty(c,'prototype',
    {writable: false, configurable: false}};
Object.defineProperty(c.prototype,'constructor',
    {writable: false, configurable: false}};
```

Declarative Classes: Prototype Inheritance

Harmony

```
class c {
    <prototype: p>
};
```

ES5

```
function c() {};
Object.defineProperty(c,'prototype',
    {value: Object.create(p),
     writable: false, configurable: false}};
Object.defineProperty(c.prototype,'constructor'
,
    {writable: false, configurable: false}};
```

Declarative Classes: Class Inheritance

Harmony

```
class D {
    <superclass: C>
};
```

ES5

```
function D() {};
Object.defineProperty(D,'prototype',
    {value: Object.create(c.prototype),
     writable: false, configurable: false}};
Object.defineProperty(D.prototype,'constructor'
,
    {writable: false, configurable: false}};
```

And some other things …

Declarative Classes: Prototype Properties

Harmony

```
class c {
   method m() {return this.a+this.x}
   new (a) {
       a : a,
       {this.x = somebody(this)}
   }
};
```

ES5

```
function c(a,b) {
  Object.defineProperty(this,"a",
     {value: a,enumerable: true, writable: true, configurable: true});
  (funtion () {this.x = somebody(this)}).call(this);
};
Object.defineProperty(c,'prototype',
    {writable: false, configurable: false}};
Object.defineProperty(c.prototype,'constructor',
    {writable: false, configurable: false}};
Object.defineProperty(c.prototype,'m',
    {value: function () {return this.a+this.x}, writable: false,
     enumerable: false, configurable: true});
```

Declarative Classes: Constructor Properties

Harmony

```
class c {
   class method cm(a,x) {return a.a+x.b}
   new (a) {
       a : a,
       {this.x = somebody(this)}
   }
};
```

ES5

```
function c(a,b) {
  Object.defineProperty(this,"a",
     {value: a,enumerable: true, writable: true, configurable: true});
  (funtion () {this.x = somebody(this)}).call(this);
};
Object.defineProperty(c,'prototype',
    {writable: false, configurable: false}};
Object.defineProperty(c.prototype,'constructor',
    {writable: false, configurable: false}};
Object.defineProperty(c,’cm',
    {value: function (a,x) {return a+x.x}, writable: false,
     enumerable: false, configurable: true});
```

Declarative Classes: Freezing, Sealing, etc.

Harmony

```
class D {
    <superclass: C, frozen>,
    new (a) {
        <closed>,
        a : a,
        {this.x = somebody(this)}
    }
};
```

Declarative Classes: Super Initialization

Harmony

```
class D {
    <superclass: C>,
    new (a,b) {
        super new(a),
        b : b,
        {this.x = somebody(this)}
    }
};
```

Calls super class' constructor code on subclass instance. Includes create and initializing super class instance properties.

Declarative Classes: Super Method Calls

Harmony

```
class D {
   <superclass: C>,
   method validate() {
      super.validate();
      …
   }
};
```

ES5

```
function validate() {
   C.prototype.validate();
   …
}
```

Class declaration solves static binding of super lookup start point

Declarative Classes: Unique property names

Harmony

```
class D {
    private m,  //class scope
    method m() {this.m();},
    class method m(obj) {obj.m()},
    new (a) {
        private n,  // instance scope
        method m() {super.m()},
        n : 0,
        method x() {D.m(this.n)}
    }
};
```

Another Chapter 15 Concept

- Encapsulated per instance state:
  - [[PrimitiveVale]]
  - [[TargetFunction]]
  - [[BoundThis]]
  - [[Scope]]

Instance Variable

- Non-property per instance state, name/value pairs
  - Not accessible using . or [ ]
  - Not reflected
  - Not trapped to proxies
  - Not enumerated
- Accessed via special syntax and lexically scoped private names.
  - @name //RHS
  - obj@name //RHS
  - {@name: init} //declarative definition
  - @name = //imperative LHS definition (in constructor)
  - @name = //LHS anywhere
- Instance variable set closed after construction

Declarative Classes: Instance Variables

Harmony

```
class D {
   private iv1, //class private
   method m() {return @iv1},
   class method m(obj) {obj@iv1()},
   new (a) {
       private iv2,  // instance private
       @iv1: 0,  //declared inst var
       method x(obj) {
          return @iv2+obj@iv2},
       {@iv2 = Math.random(),
            //imperative inst var creation
       }
   }
};
```