FrOSCon
Free & Open Source Software Conference

Advanced JavaScript

Konstantin Käfer

FrOSCon
Free & Open Source Software Conference

Contents

Why JavaScript?

- Lightweight language
- No static type system
- Functions as first class objects

- Because you have to.

JavaScript vs. the DOM

- Often confused
- DOM: API for accessing a web page
- The DOM is not JavaScript specific

- The DOM has a **bad reputation**:
  - Incompatibilities between browsers
  - Verbose syntax
  - Inconvenient to use

JavaScript vs. browsers

- JavaScript originated in Netscape as “LiveScript”
- JavaScript is not specific to browsers
- Other uses:
  - Scripting Mozilla’s interface
  - Scripting Adobe applications
  - Used as a base for ActionScript
  - Server-side JavaScript

Demo: AppJet

Other uses

- 3D rendering in the browser
- Reading ID3 tags from MP3 files
- Implementing a programming language

http://ejohn.org/blog/processingjs/

FrOSCon
Free & Open Source Software Conference

Contents

Functions

- Functions are **first class entities**
  - Store in variables, pass as parameters, return from functions
  - Can be defined at any place
- Functions can contain properties
- Anonymous functions
- Closures

Functions *(II)*

```
var foo = function(callback) {
  callback();
  return function() {
    print("Returned function called");
  };
};

foo(function() {
  print("Passed function called");
})();

foo.bar = "baz";
```

Prototypal OOP

- JavaScript doesn't have classes
- Prototype of a function used as base class

```
var Foo = function() { /* ... */ };

Foo.prototype = {
   'bar': function() { /* ... */ },
   'baz': function() { /* ... */ }
};

var instance = new Foo();
instance.bar();
instance.baz();
```

Prototypal OOP *(II)*

- Function is constructor
- “Instances” have an implicit link to the base class

```
var Foo = function() { /* ... */ };
Foo.prototype = {
  'bar': function() { /* ... */ }
};

var instance = new Foo();
instance.bar();

Foo.prototype.baz = function() { /* ... */ };
instance.baz();
```

Prototypal OOP *(III)*

- Any objects can be extended/changed at any time

```
Number.prototype.celsiusToFahrenheit = function() {
  return (this * 9 / 5) + 32;
};

js> (34).celsiusToFahrenheit();
93.2
js> (0).celsiusToFahrenheit();
32
```

Scope

- JavaScript has a lexical (static) scope
- Scope contains everything that is visible when the function is **defined**
- `this` is the context the function is executed in
- Functions can be executed in other contexts

Scope *(II)*

```
var bar = function() {
  var foo = "foo";

  return function() {
    console.log(foo);
  };
}();

js> bar();
foo
```

- Scope lives on even after the outer function returned

Scope *(III)*

```
var formulas = {
  'Celsius': {
    'Fahrenheit': 'this * 1.8 + 32',
    'Reaumur': 'this * 0.8'
  },
  'Fahrenheit': {
    'Celsius': '(this - 32) / 1.8',
    'Reaumur': '(this - 32) / 2.25'
  },
  'Reaumur': {
    'Celsius': 'this * 1.25',
    'Fahrenheit': 'this * 2.25 + 32'
  }
};
```

Scope *(IV)*

```
(function() {
  var formulas = ...;

  for (var from in formulas) {
    for (var to in formulas[from]) {
      Number.prototype[from + 'To' + to] =
      (function(formula) {
        return function() {
          return eval(formula);
        };
      })(formulas[from][to]);
    }
  }
})();
```

Array extensions

- Provide functional programming style for arrays

```
[1, 2, 3].forEach(function(i) {
  print(i);
});
```

JavaScript 1.7

```
function fib() {
  var i = 0, j = 1;
  while (true) {
    yield i;
    var t = i;
    i = j;
    j += t;
  }
}

var g = fib();
for (var i = 0; i < 10; i++) {
  print(g.next());
}
```

Contents

Singletons

- Regular object.

Inheritance

```
var foo = function() { };
foo.prototype.value = "foo";
foo.prototype.bar = function() {
  print(this.value);
};
```

```
var baz = function() { };
baz.prototype = new foo; // baz inherits from foo
baz.prototype.value = "baz";

(new foo).bar(); // prints "foo"
(new baz).bar(); // prints "baz"
```

Inheritance *(II)*

```
// ...

foo.prototype.betterBar = function() {
  print("value: "+ this.value);
};

(new baz).betterBar(); // prints "value: baz"
```

Decorator

- Creating an instance that is *slightly* different
- JavaScript allows overwriting methods

```
Function.prototype.decorate = function(pre, post) {
  var old = this;

  return function() {
    if (pre) pre.apply(this, arguments);
    old.apply(this, arguments);
    if (post) post.apply(this, arguments);
  };
};
```

Decorator *(II)*

```
var foo = function() { };
foo.prototype.bar = function(message) {
  print(message);
};
```

```
var baz = new foo();
baz.bar = baz.bar.decorate(function() {
  print("pre");
});

baz.bar("message");
```

Delegates

- Delegate tasks to another object
- One object can use many different delegate objects

```
var simpleDataSource = {
   'length': function() { return storage.length; },
   'item': function(i) { return storage[i]; }
};
```

Delegates *(II)*

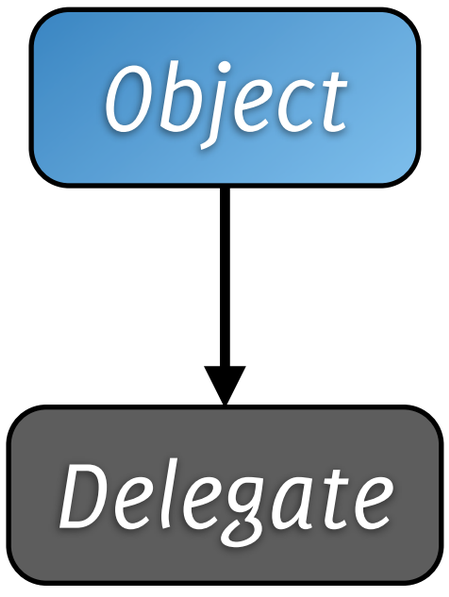
```
var urlDataSource = function(src) {
  this.source = src;
};

urlDataSource.prototype = {
  'length': function() {
    return loadData(this.source,
      { 'command': 'length' });
  },
  'item': function(i) {
    return loadData(this.source,
      { 'command': 'fetch', 'id': i });
  }
};
```

Delegates *(III)*

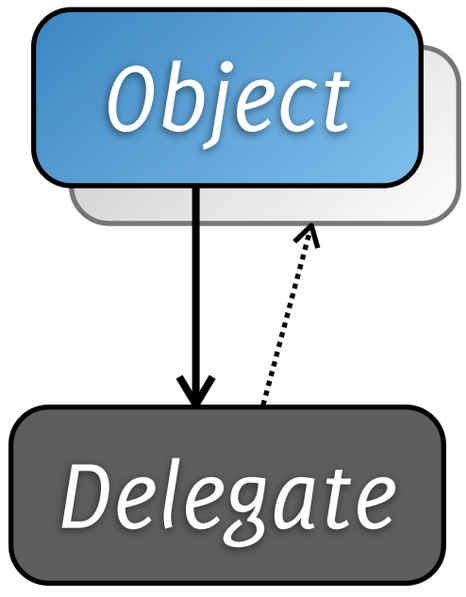
```
var obj = function(datasource) {
  this.datasource = datasource;
};

obj.prototype.foo = function() {
  var length = this.datasource.length();
  for (var i = 0; i < length; i++) {
    var item = this.datasource.item(i);
    // do something with item
  }
};

var foo = new obj(simpleDataSource);
var bar = new obj(new urlDataSource
                      ("http://example.com"));
```

Delegates *(IV)*

- Pull vs. Push



Synchronous Call



Asynchronous Call

(Calls supplied method when the delegated action is finished)

Named parameters

```
var extend = function(obj, vars) {
  for (var i in vars)
    if (obj[i])
      extend(obj[i], vars[i]);
    else
      obj[i] = vars[i];
};

var foo = function(params) {
  extend(this, params);
};

var bar = new foo({ position: 'left', start: 3 });
```

Currying

- Invented in 1967
- Named in honor of Haskell Curry
- Also used in languages like Ruby, Haskell, Smalltalk
- Pre-configure function parameters
- Alter the context when called (the `this` variable)

Currying *(II)*

```
Function.prototype.curry = function(scope) {
  var fn = this, args = [];

  for (var i = 1; i < arguments.length; i++)
    args.push(arguments[i]);

  return function() {
    var params = [].concat(args);

    for (var i = 0; i < arguments.length; i++)
      params.push(arguments[i]);

    return fn.apply([ this, scope || window ], params);
  };
};
```

Currying *(III)*

```
function add(a, b) {
   return parseInt(a) + parseInt(b);
}

var add5 = add.curry(null, 5);
add5(10); // Result: 15
```

Currying *(IV)*

```
var foo = function(str, e) {
  console.assert(this == 'test');
};

$('a.test').click(foo.curry('test'));
```

Memoization

- Save intermediate results

```
var fib = function(memo) {
  return function(n) {
    if (typeof memo[n] !== 'number')
      memo[n] = fib(n - 1) + fib(n - 2);

    return memo[n];
  };
}([0, 1]);

fib(24);
fib(8); // no new calculations needed
```

Contents

4. Debugging and Analyzing

Firebug

web development evolved

- Advanced JavaScript console
- Logging to the console (console.log())
- DOM inspector
- JavaScript debugger (with backtrace!)
- Profile JavaScript activity

- http://getfirebug.com

Firebug Lite

- Console for other browsers
- No profiling
- Doesn’t do your laundry

- http://getfirebug.com/lite.html
- http://drupal.org/project/firebug

- JavaScript debugger for IE
- Free of charge
- Some configuration work needed
- http://msdn.microsoft.com/vstudio/express/vwd/

WebDevHelper

- JavaScript console for IE
- HTTP Logger
- JavaScript backtracer

- http://projects.nikhilk.net/WebDevHelper/Default.aspx

JavaScript Lint

- Lint = tool for analyzing code
- Discovers sloppy coding
- Command line interface
- Use as pre-commit hook for `<insert RCS>`
- http://javascriptlint.com/
- TextMate bundle: http://andrewdupont.net/2006/10/01/javascript-tools-textmate-bundle/

Sources

- JavaScript: The Good Parts, *Douglas Crockford*, 2008.
- Pro JavaScript Design Patterns,
  Ross Harmes; Dustin Diaz, 2008.
- Pro JavaScript Techniques, *John Resig*, 2006.
- ...