



JavaScript Master Seminar

Module Pattern

Sirma Gjorgievska
Johannes Fischer

Technische Universität München

September 07, 2015



Agenda

- Introduction
- The Basics
- Augmentation
- Shared Private State
- Submodules
- Inheritance
- Object literal
- Module APIs
- Demonstration
- Conclusion



What is a Module?

- Integral piece of robust application's architecture
- Keeps the units of code separated and organized



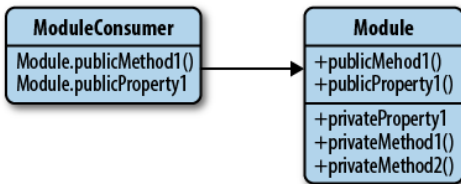
Implementation of modules

- The Module pattern
- Object literal notation
- AMD modules
- CommonJS modules
- ECMAScript Harmony modules



What is a Module pattern?

- JavaScript design pattern
- Developed in 2003
- Private and public encapsulation
- Mimic classes in software engineering



Advantages

- Cleaner approach for developers
- Supports private data
- Less clutter in global namespace
- Localization of functions and variables



Disadvantages

- Inability to create automated unit tests
- Loss of extensibility (sometimes)
- Problems when changing visibility of public/private members



The Basics

```
var name = $("#nameField").val();  
var password = "password";  
var login = new function () {  
    alert(name + " log in using '" + password + "' as password.");  
}
```

Simple code without patterns



Anonymous Closures

- Defined function is executed immediately
- Code inside the function lives in a **closure**
- It provides **private state**
- Maintains access to all globals

```
(function () {  
    var name = $("#nameField").val();  
    var password = "password";  
    var login = new function () {  
        alert(name + "log in using'" + password + "'as password.");  
    }  
    // ...  
})();
```





Private methods

- Methods locally declared in modules
- Inaccessible outside of the scope defined

```
(function () {  
    var name = $("#nameField").val();  
    var password = "password";  
    var login = new function () {  
        alert(name + "log in using'" + password + "'as password.");  
    }  
    // ...  
})();
```

Implied Globals

- Hard-to-manage code
- Not obvious (to humans) which variables are global



Global Import

- Better alternative
- Passing globals as parameters to anonymous function
- Better efficiency and readability

```
(function ($) {  
    var name = $("#nameField").val();  
    var password = "password";  
    var login = new function () {  
        alert(name + " trying to log in using '" + password + "' as password.");  
    }  
    // ...  
})(jQuery);
```

Module Export

- Declare globals for further use
- Return value of anonymous function
- Module variables readable afterwards
- Namespacing (avoids varname conflicts)

Module Export

```
var MODULE = (function ($) {  
    var m = {};  
    m.name = $("#nameField").val();  
    var password = "password";  
    m.login = new function () {  
        alert(m.name + " trying to log in using '" + password + "' as password.");  
    }  
    // ...  
    return m;  
})(jQuery);
```

Augmentation

Problems:

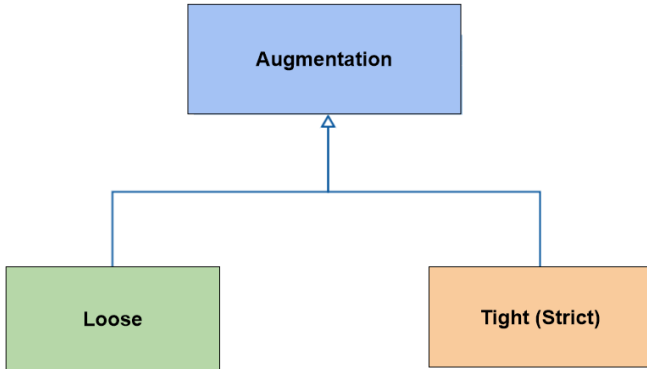
- Entire module must be in one file
- Not extendable



Solution: Augment modules



Augmentation



Tight (Strict) Augmentation

```
var MODULE = (function (m) {  
    m.login = new function () {  
        console.log(m.name + " trying to log in");  
        someLoginMethod(m.name, password);  
    }  
    // ...  
    return m;  
})(MODULE);
```

Tight (Strict) Augmentation

- Parameter: *MODULE*
- Loading order has to be fixed
- Properties of earlier module files can be used reliably
- Properties overwritable

Loose Augmentation

```
var MODULE = (function (m) {  
    m.method1 = new function () {  
        // ...  
    }  
    // ...  
    return m;  
})(MODULE || {});  
  
var MODULE = (function (m) {  
    m.method2 = new function () {  
        // ...  
    }  
    // ...  
    return m;  
})(MODULE || {});
```

Loose Augmentation

- Parameter: *MODULE* `// {}`
- Loading order irrelevant
- Module files can be loaded in parallel

Augmentation

Tight Augmentation

vs

Loose Augmentation

- Allows overrides
- Loading order fixed
- Parameter: *MODULE*

- Cannot override safely
- Loading order not fixed
- Parameter: *MODULE || {}*

Disadvantage

- Inability to share private variables between files





- Variable `_private` identical for all modules

Shared Private State

```
var MODULE = (function (m) {  
  var _private = m._private = m._private || {};  
  _private.seal = _private.seal || function () {  
    delete m._private;  
  };  
  _private.unseal = _private.unseal || function () {  
    m._private = _private;  
  };  
  m.loadModule = m.loadModule || function (url) {  
    _private.unseal()  
    loadJSFile(url);  
    _private.seal();  
  }  
  // ...  
  _private.seal();  
  return m  
})(MODULE || {}));
```



Submodules

- Creation is the same as regular modules
- All the advanced capabilities of normal modules

```
var MODULE = MODULE || {};  
  
MODULE.sub = (function () {  
    var s = {};  
    // ...  
    return s;  
})();
```



Inheritance

```
var MODULE = (function (parent) {  
    var m = {};  
  
    for(var key in parent)  
        if(parent.hasOwnProperty(key))  
            m[key] = parent[key];  
  
    m.parent = parent; // important to do this last  
    return m;  
})(PARENT));
```

- *Parent* module has to exist
- New module now has *parent* and *parent's* properties
- *m.parent* could be left out



Object Literal

- Comma-separated list of name-value pairs wrapped in curly braces
- Encapsulate data, enclosing it in a tidy package

```
var myObjectLiteral = {  
    variableKey: variableValue,  
    functionKey: function () {  
        // ...  
    }  
};
```

Object Literal

- Used in Module pattern as the return value from a scoping function

```
var Module = (function () {  
    var privateMethod = function () {};  
    return {  
        publicMethodOne: function () {  
            },  
        publicMethodtwo: function () {  
            }  
    };  
})();
```



Accessing properties

- Dot notation
- Bracket notation
 - allows a variable to specify all or part of the property name
 - allows property names to contain characters forbidden in dot notation

```
object.foo = object.foo + 1;  
object["foo"] = object["foo"] + 1;  
object["!@#$%^&*()."] = true;
```

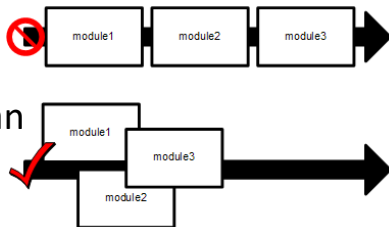
Script hell

```
3 <head>
4   <meta charset="utf-8" />
5   <meta name="Author" content="Apple Inc." />
6   <meta name="viewport" content="width=1024" />
7   <meta http-equiv="X-UA-Compatible" content="IE=EmulateIE7, IE=9" />
8   <link id="globalheader-stylesheet" rel="stylesheet" href="http://img
9
10  <title>Apple</title>
11  <meta name="omni_page" content="Apple - Index/Tab" />
12  <meta name="Description" content="Apple designs and creates iPod and
and the revolutionary iPhone and iPad." />
13  <link rel="home" href="http://www.apple.com/" />
14  <link rel="alternate" type="application/rss+xml" title="RSS" href="ht
15  <link rel="index" href="http://www.apple.com/sitemap/" />
16  <link rel="stylesheet" href="http://images.apple.com/global/styles/bg
17  <link rel="stylesheet" href="http://images.apple.com/v/home/s/styles/
18  <link rel="stylesheet" href="http://images.apple.com/v/home/s/styles/
19  <link rel="stylesheet" href="http://images.apple.com/home/styles/home
20  <script src="http://images.apple.com/global/scripts/lib/prototype.js"
21  <script src="http://images.apple.com/global/scripts/lib/scriptaculous
22  <script src="http://images.apple.com/global/scripts/lib/sizzle.js" ty
23  <script src="http://images.apple.com/global/scripts/browserdetect.js"
24  <script src="http://images.apple.com/global/scripts/apple_core.js" ty
25  <script src="http://images.apple.com/global/scripts/search_decorator.
26  <script src="http://images.apple.com/global/scripts/feedstatistics.js
27  <script src="http://images.apple.com/global/ac_base/ac_base.js" type=
28  <script src="http://images.apple.com/global/ac_retina/ac_retina.js" t
29
30  <script src="http://images.apple.com/global/scripts/promomanager.js"
31  <script src="http://images.apple.com/v/home/s/scripts/home.js" type="
32 </head>
33 <body id="home" class="home">
34
35   <script type="text/javascript">
36     var searchSection = 'global';
37     var searchCountry = 'us';
38     var aiRequestsEnabled = true;
39     var aiDisplaySuggestions = true;
40   </script>
41
42  <script src="http://images.apple.com/global/nav/scripts/globalnav.js" type="t
```



Asynchronous Module Definition

- Specifies a mechanism for defining modules
- Module and dependencies can be asynchronously loaded
- Many advantages:
 - Dependencies are easy to identify
 - Avoids global variables



AMD modules

- API consists of a single function:

```
define(id?, dependencies?, factory);
```

- Sample module that has dependencies on jQuery and jQuery cookie plugin:

```
define(['jQuery'],function($){  
    var version = '1.0';  
  
    return {  
        version: version,  
        init: function () {  
            // Do something  
        }  
    };  
});
```

CommonJS modules

- Module proposal that specifies a simple API for declaring modules
- Better suited for server-side environments than for browsers
- Most popular implementation is in Node.js
- Performance suffers when browsers must load a lot of modules



CommonJS modules

- Standardize the following scoped variables:
 - ***require***
 - ***exports***
 - ***module***

```
// say.js
exports.hello = function (name) {
    return 'Hello, ' + name + '!';
};

// alert.js
var say = require('say');

exports.alertHello = function (name) {
    alert(say.hello(name));
};
```





- Good format for users of CommonJS and AMD
- Compact syntax
- Direct support for asynchronous loading

ES6 Harmony modules

- Two kinds of exports:
 - Named exports (several per module)
 - Default exports (one per module)
- **Named export**

```
// lib.js
export function circumference(radius) {
    return radius * 2 * Math.PI;
}

// main.js
import {circumference} from 'lib';
console.log(circumference(3));
```



ES6 Harmony modules

- Another alternative is to import the complete module

```
// main.js imports everything by using a wildcard  
import * as lib from 'lib';  
console.log(lib.circumference(3));
```

- **Default export**

```
// lib.js has a single default export  
export default class { ... } // no semicolon!  
  
// main.js  
import MyClass from 'MyClass';  
let inst = new MyClass();
```



- Simulation of cellular automata
- Implemented using JavaScript and Module Pattern
- A **cellular automaton** is a discrete model
- Consists of a regular grid of *cells*
- The most popular is **Conway's Game of Life**

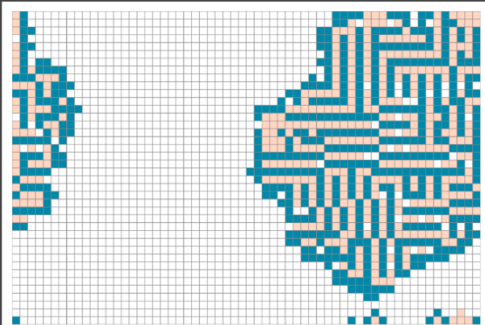
Live Demonstration



Life Family Automata

Simulation of multiple cellular automata, the most famous of which is Conway's game of life

Start the simulation by choosing a strategy and clicking "Play"



Start/pause your game now

▶ Play

Choose your favourite strategy

☐ Conway's rules

☒ Maze pattern

☐ Life without death

☐ Day and night

Make your canvas clear

↺ Clear canvas



- Use of Module pattern in our application

```
// main.js
var MODULE = (function (m) {
    /* This function calls the overridable function m.calculateCell for each cell
     * It then replaces the old cells array with the results of all these calls */
    m.updateCells = function () {
        for (var i = 0; i < cells.length; i++) {
            cells[i] = m.calculateCell(...);
        }
    }
    /* This function will be overwritten by the loaded modules */
    m.calculateCell = function (isActive, neighbours) {
    }
    m.drawCanvas = m.drawCanvas || function () {
    }
    return m;
})(MODULE || {});
```

- Use of Module pattern in our application

```
// extension.js
var MODULE = (function (m) {
    m.drawCanvas = function () {
    }
    return m;
})(MODULE || {}));
```

```
// strategy.js
var MODULE = (function (m) {
    m.calculateCell = function (isAlive, neighbours) {
    }
    return m;
})(MODULE));
```


Conclusion

- Module pattern is very common JavaScript pattern
- Faster download (parallel)
- No performance penalty
- Shorter reference chains might increase performance
- **Loose augmentation** allows easy non-blocking parallel download



Thank you

