



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
- AMD modules
- CommonJS modules
- Harmony modules
- Demonstration
- Conclusion



What is Module?

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



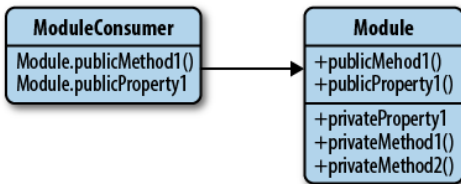
Implementation of modules

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



What is 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

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

Augmentation

Problems:

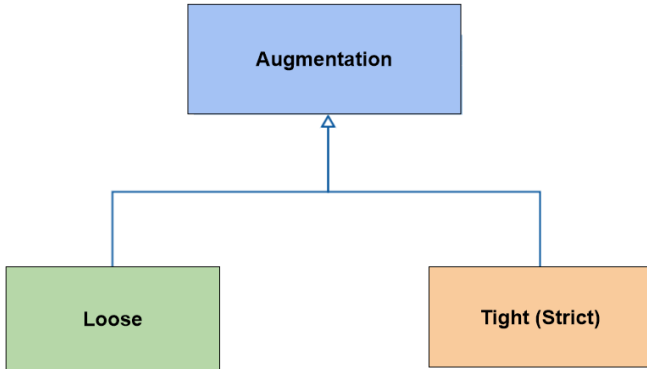
- Entire module must be in one file
- Not extendable



Solution: Augment modules



Augmentation



Tight (Strict) Augmentation

```
MODULE = (function (m) {  
    m.login = new function () {  
        console.log(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 modules usable reliably
- Properties overwritable

Loose Augmentation

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



Shared Private State

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





- Variable *_private* identical for all modules
- Only accessible from old module files
- Unlocks *_private* for later module file loading

Submodules

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

```
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* needs to be set last



Object Literal

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

```
var myObject = {  
  sProp: 'some string value',  
  numProp: 2,  
  bProp: false  
};
```

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;
```

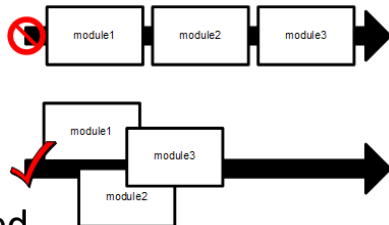
AMD modules

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



- 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

- 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));
};
```

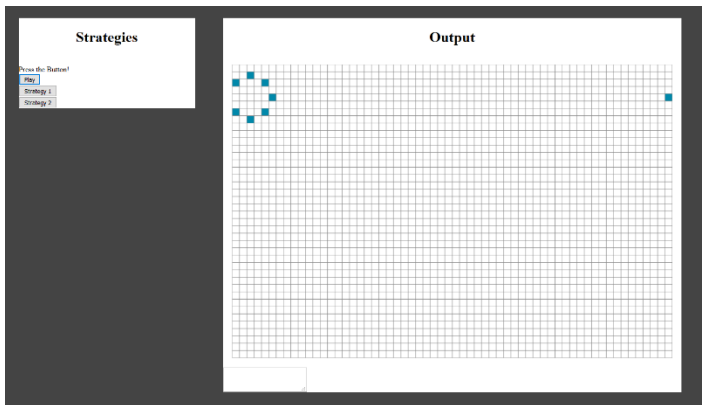



- Format good for users of CommonJS and AMD
- Modules have a compact syntax
- Modules have direct support for asynchronous loading

ES6 Harmony modules



- JavaScript implementation of John Conway's **Game of Life**



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!

