JavaScript Module Pattern

By **RupeshKumar**, 31 Aug 2011

Download source - 65.97 KB

Introduction

In this article, I will discuss one of the best coding practices in JavaScript that is known as Module Pattern.

Module Pattern is very common in the JavaScript world with its great importance. It helps us to write clean object oriented JavaScript. Module Pattern restricts the developer to create multiple Global Variables. It is always a good practice to have less number of global variables for good performance of web application. Hence this pattern increases the performance of your website, at the same time it gives the well maintainable object oriented programing practice.

What is Module Pattern

In JavaScript programing language, functions can be used as a Module. Sometimes, it is required to create a singleton object instead of creating instance of a class. Inside the module, the inner functions do have the static scope with the other defined variables and functions.

Maximum applications written in JavaScript languages are singletons only. Hence, they all are written in Module pattern.

A Module can be considered as a Singleton Class in C#. In a Module, all the variables defined are visible only in the module. Methods in a module have scope and access to the shared `private` data and `private` methods. Hence, these methods are also called as **Privileged methods**.

Therefore, by creating one anonymous function returns a set of privileged methods in a simple object literal format. And because of the closure principle, those privileged methods have access to the inner methods and variables of the anonymous function. We call it as a **Module Pattern**.

Privileged Method

In Module Pattern, you can write your `private` methods, `private` variables those you don't want to expose to

the other world and at the same time you will be able to write `public` and privileged methods also. The great usefulness of this pattern is that you can very well access your `private` methods and variables from your methods those are called as **Privileged Methods**.

Closure

In order to learn Module Pattern, one must be aware of Closure principle. By using closure only modular pattern becomes very useful and powerful. Closure principle says that any inner function inside a parent function has scope to the other inner function and variables although after the parent function has returned. In Module Pattern, Privileged methods have scope to the other `private` variables and methods of the module.

Self-executed Function

In Module Pattern, we create one Global Singleton Object by writing self-executed method. Self-executed method is an anonymous method that is being called by itself only. See the below code that explains how we can write self-executed method.

Example of Self-executed function

```
(function ( ) {
    //create variable
    //var name = "rupesh";
    //create method
    //var sayName = function ( ) {
    //    alert(name);
    //};
})( );//calling this method.
```

In the above code, I have one anonymous method that is declared and just after declaration it has been called by `writing();` The moment we execute this JavaScript code, this method will get executed by itself only. Hence it is called a self-executed method.

Sample of Module Pattern

Having understood the idea of module pattern, self-executed method, closure now let me define the module pattern style. See the below code:

```
//Single Global Variable "Module"
var Module = ( function ( ) {
    var privateVariable = "some value";
    var privateMethod = function ( ) {
        //do something.
    }
    //returning one anonymous object literal that would expose privileged methods.
    return {
         //the method inside the return object are
         //called as privileged method because it has access
         //to the private methods and variables of the module.

         privilegedMethod : function ( ) {
            //this method can access its private variable and method
            //by using the principle of closure.
            alert(privateVariable); //accessing private variable.
```

```
            privateMethod( ); //calling private method
        }
    }
})( );
```

Here `Module` is the Single Global Variable that is exposed to the document. We are declaring, calling one anonymous method and assigning it to `Module` variable. Now we can call `privilegedMethod` by writing `Module.privilegedMethod();` internally privileged Method of module can access its `private` variable and `private` method. Because, they come under their `static` scope. If we have any data or method that we don't want to expose, we can put them in `private` methods.

We can also write the above code in a different manner, see the below way:

```
//Single Global Variable "Module"
var Module = ( function ( ) {
    var privateVariable = "some value";
    ,privateMethod = function ( ) {
        //do something.
    }
    //object literal that would have privileged methods.
    ,retObject = {
         //the method inside the return object are
         //called as privileged method because it has access
         //to the private methods and variables of the module.
         privilegedMethod : function ( ) {
            //this method can access its private variable and method
            //by using the principle of closure.
            alert(privateVariable); //accessing private variable.
            privateMethod( ); //calling private method
        }
    }
    //returning the object.
    return retObject;
})( );
```

`Public` Methods don't have access to `Private` Methods and variables, only the privileged method inside the Module can have access because of the closure. Let's see it.

We can create the `public` methods in the Module by the below method:

```
//augmenting public method in the module
Module.publicMethod1 = function () {
    //do something...
    //However here we will not have access to the private methods of the Module.
    //We can call the privileged method of the module object from here.
    //And by the privileged method we can call or access the inner private
    // methods and variables correspondingly
    Module.privateMethod ( ); // this is not possible.
    //calling a privileged method of module
    Module.privilegedMethod ( ); // this can be possible.
}
```

In the above example, the `public` method `publicMethod1` that we augmented into the module doesn't have access to the `privateMethod` of the `Module` object. In order to access the `private` Methods or Variables, we can call the privileged method of the Module inside the `public`. This is the benefit of this pattern. In our above example, we could call `Module.privilegedMethod()`.

Example of Module Pattern

I have created one sample .NET application, the same is attached here in this article. In that .NET Application, I have created one Module pattern JavaScript function. Let's discuss the same function. See the below code snippet:

```
//One Global object exposed.
var SearchEngine = (function ( ) {
    //Private Method.
    var luckyAlgo = function ( ){
        //create one random number.
        return Math.floor(Math.random()*11);
    }
    //Returning the object
    return {
        //privileged method.
        getYourLuckyNumber : function ( ){
            //Has access to its private method because of closure.
            return luckyAlgo();
        }
    }
} ) ( );//Self executing method.
```

In the above code snippet, I have one Global variable called `SearchEngine.` A self-executed anonymous method is assigned in to the `SearchEngine` variable.

`SearchEngine` has one `Private` method `luckyAlgo` and one Privileged Method `getYourLuckyNumber.` The Privileged method is being returned by encapsulating in an anonymous object. `SearchEngine` can be accessed globally that will have one Privileged method only. And the Privileged method can call its local method `luckyAlgo`, because of the principle of Closure. In the above example, if you call `getYourLuckyNumber` method then it will return one random number.

Creating Sub Modules

We can create the sub-module as well by augmenting our module object with a sub module. It will be again in the same module pattern fashion. See the below example, I have created one more `subSearch` object inside the `SearchEngine` object by following the same module pattern.

```
//Augmenting the module with submodule
SearchEngine.subSearch = (function ( ) {
   //Private variable.
   var defaultColor = "Orange";
    //private method.
    var myColorAlgo = function (num) {
        switch(num){
            case 1:
            defaultColor ="Green";break;
            case 2:
            defaultColor ="Black";break;
            case 3:
            defaultColor ="Yellow";break;
            case 4:
            defaultColor ="White";break;
            case 9:
            defaultColor ="Red";break;
        }
    }
```

```
    return {
    //privileged method
        getYourLuckyColor : function(){
            //access to private variable because of closure.
            myColorAlgo(SearchEngine.getYourLuckyNumber());
            return defaultColor;
        }
    }
})();
```

In the above example, if we call the `getYourLuckyColor` method, then it will give you one color name based on the random number. I know these methods are not much practical however, I could think of them only in order to explain the concept.

Extending Modules

We can extend the existing modules by wrapping it inside one new module. By doing that, we can very well access / override the existing method of the old module. See the below code:

```
var Module1 = (
function (oldModule) {
    //assigning oldmodule in to a local variable.
    var parent = oldModule;
    //overriding the existing privileged method.
    parent.privilegedMethod = function ( ){
         //do something different by overriding the old method.
    };
    //private method accessing privileged method of parent module.
    var privateMethod2 = function ( ) {
        parent.privilegedMethod();//can access privileged method of Module
        parent.publicMethod1(); //can access public method of Module
    }
    return {
        newMethod : function ( ) {
          ///do something for this brand new module.
          ///use some functionality of parent module.
          /// parent.privilegedMethod( );
        }
    }
}
)(Module);//Module object is the existing module that I want to extend.
```

In the above example, I have described the advanced features of Module pattern where one can re-use the existing module by extending, augmenting and overriding.

I have passed the old existing `Module` object *(see above in module pattern section, we have created one Module object)* while calling our anonymous method and then I have assigned it into a local variable called as `parent`. Now, I can access the `public`/privileged methods of the parent `Module` in my `private`, privileged and `public` method of new `Module1` object.

The sweetness of this pattern comes when I can also override the existing method of the old `Module`. And you can see, I have overridden the `privilegedMethod` of the `Module` object. Similarly, I could have overridden the `public` method of the `Module` object

Conclusion

I have given very simple examples. However, one can learn this pattern and create very complex coding also. You must be aware of YUI this is a great JavaScript library It is written on the principle of Module Pattern only. In my personal experience, I used to write JavaScript code more in Module Pattern and I enjoy it a lot. Now this pattern is my favorite one and I have lots of fun also. Hope this article will help you. Please give your comments and suggestions.

License

About the Author

RupeshKumar
Software Developer (Senior)

United States
Member

Follow on Twitter

I am a software developer working since 2005 in ASP.Net and related Technologies. I work on C#, Asp.Net, WCF, Silverlight, WPF, Jquery, Javascript, SQL, HTML5, CSS3, RAZOR, MVC, EntityFramework.

I enjoy on exploring new technologies and writing about them.

Great interest in Design Patterns. I love writing JavaScript codes, now my favorite JavaScript library is jQuery. I enjoy writing jQuery Plugins as well.

I write in Technical blog
I can Twitt.
I am in linkedin
I created website of Swami Vivekananda posted some speech of him read by me.

Comments and Discussions

28 messages have been posted for this article Visit **http://www.codeproject.com/Articles/247241/Javascript-Module-Pattern** to post and view comments on this article, or click **here** to get a print view with messages.