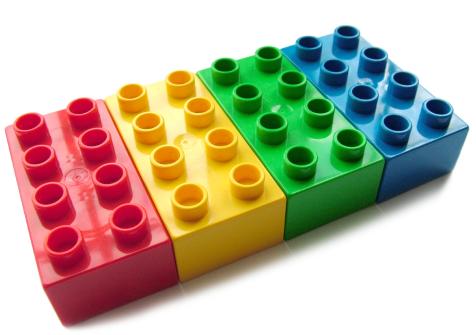
**[Patrón Módulo en JavaScript (Module Pattern)](http://hardcoders.mx/blog/2014/09/12/patron-modulo-en-javascript-module-pattern/)**

Sep 12th, 2014 | [Comments](http://hardcoders.mx/blog/2014/09/12/patron-modulo-en-javascript-module-pattern/#disqus_thread) | Escrito por: Isaac Zepeda



Los módulos son muy importantes ya que nos permiten mantener nuestro código encapsulado, sin contaminar el scope global y evitar colisión de nombres. Además nos ayudan a mantener en el proyecto unidades de código separadas y organizadas.

El *Module Pattern* es considerado un Patrón de Diseño y en JavaScript nos ofrece la posibilidad de simular propiedades y métodos privados. Digo “simular” porque las variables y funciones en JavaScript no tienen modificadores de acceso, pero a través de los *closures* podemos simular este comportamiento.

El *Module Pattern* se implementa creando una función anónima que se auto-invoca y regresa un objeto literal.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | var myModule = (function () {  var counter = 0;  return {  incrementCounter: function () {  return counter++;  },  resetCounter: function () {  console.log('Valor de counter antes de reset: ' + counter);  counter = 0;  }  }  })();  // Uso:  myModule.incrementCounter();  myModule.incrementCounter();  myModule.resetCounter(); // Imprime 2 |

De esta forma al tener una función anónima creamos un scope dentro de la función, evitando así contaminar el ámbito global. Regresamos un objeto literal que contiene dos métodos, ambos métodos pueden acceder a la variable counter ya que se ha creado un closure. Así podemos simular propiedades privadas.

Entonces básicamente el *Module Pattern* se define de la siguiente forma.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | var myModule = (function () {  var privateProperty = 10;  var privateMethodOne = function () {  // Algo  };  var privateMethodTwo = function () {  // Algo  };  return {  publicProperty: "foo",  publicMethodOne: function () {  //...  },  publicMethodTwo: function () {  // Invocar método privado  privateMethodOne();  },  publicMethodThree: privateMethodTwo //Alias de privateMethodTwo  }  })(); |

**Revealing Module Pattern (Patrón Módulo Revelado)**

El *Revealing Module Pattern* es una variante del *Module Pattern* en el cual definimos nuestras propiedades públicas y privadas dentro la función anónima y en el return apuntamos a las funciones que deseamos exponer. Así tenemos un buen manejo de código, mas limpio y entendible.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | var Counter = (function () {  var counter = 0;  var privateMethod = function () {  // Algo privado  };  var incrementCounter = function () {  return counter++  };  var resetCounter = function () {  console.log('Valor de counter antes de reset: ' + counter);  counter = 0;  };  return {  increment: incrementCounter,  reset: resetCounter  }  })(); |

Podemos ver claramente cuales métodos son los expuestos por el módulo.

Otra de las razones de este patrón es que en el *Module Pattern* cuando necesitas invocar un miembro público se hace necesario usar el this, y hay quienes consideran esto confuso y menos consistente.

Module Pattern, usando this

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | var Module = (function () {  return {  methodOne: function () {  // Something  },  methodTwo: function () {  // Invocar al método público methodOne  this.methodOne();  }  };  }); |

En el *Revealing Module Pattern* ya no se hace necesario el this.

Revealing Module Pattern, no usa this

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | var Module = (function () {  var methodOne = function () {  // Something  };  var methodTwo = function () {  // Invocar al método público methodOne  methodOne();  };  return {  methodOne: methodOne,  methodTwo: methodTwo  };  }); |

**Convención para métodos privados**

Puede llegar el momento que nuestro módulo crezca demasiado (regularmente sucede) y se hace un poco tedioso tratar de ver cuales métodos y propiedades son públicos y cuales privados. Por esta razón muchos siguen una convención de usar el prefijo \_ en el nombre del miembro cuando se trata de un miembro privado.

Convención para miembros privados

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | var myModule = (function () {  var \_privateProperty = 10;  var \_privateMethodOne = function () {  // Algo  };  var \_privateMethodTwo = function () {  // Algo  \_privateMethodOne();  };  var publicMethodOne = function () {  //...  };  var publicMethodTwo = function () {  // ...  };  return {  publicProperty: "foo",  publicMethodOne: publicMethodOne,  publicMethodTwo: publicMethodTwo,  publicMethodThree: \_privateMethodTwo //Alias de \_privateMethodTwo  }  })(); |

**Conclusión**

Las principales ventajas de usar el *Module Pattern* son la encapsulación y limpieza de tu código. Además de tener miembros privados.

El *Revealing Module Pattern* nos permite ser mas consistentes en nuestros scripts.

Sin embargo hay algunas desventajas:

* Los miembros públicos y privados se acceden de forma distinta y si queremos cambiar la visibilidad en algún momento tendríamos que hacer cambios en cada parte que se usa ese miembro (solo aplica al *Module Pattern*).
* No podemos acceder a los métodos privados desde métodos definidos en el módulo después de su creación.
* No se pueden realizar pruebas de unidad sobre los métodos privados.
* Su extensión es complicada.

Sacado de: <http://hardcoders.mx/blog/2014/09/12/patron-modulo-en-javascript-module-pattern/>