**Patrón Modular En JavaScript**

En el mundo de javascript existe mucha facilidad para desarrollar con este lenguaje, el único detalle es que no debemos abusar de esa facilidad que nos ofrece. Al final uno obtiene variables y funciones repartidas a lo largo de todo el código y sin un orden, lo que termina siendo inmantenible. Una de las mejores practicas a la hora de escribir código es usar patrones de diseño y en este post explicaremos el patron modular para Javascript.

En javascript el patron modular emula el concepto de clases, de manera que somos capaces de incluir métodos públicos/privados y propiedades dentro de un único objeto, protegiendo las datos particulares del ámbito global, lo que ayuda a evitar la colisión de nombres de funciones y variables ya definidas a lo largo de nuestro proyecto, o API’s de terceros, a continuación unos conceptos previos para poder entender mejor el patrón modular.

**OBJETO LITERAL**

EL patron modular se basa en parte en los objetos literales por ende es importante entenderlo.  
Un objeto literal es descrito como cero o más pares nombre/valor, separados por comas entre llaves.  
Los nombres dentro del objeto pueden ser cadenas o identificadores que son seguidas por 2 puntos, dichos objetos también pueden contener otros objetos y funciones.

var objetoLiteral = {

*/\* los objetos literales pueden contener propiedades y métodos \*/*

saludo : "soy un objeto literal",

miFuncion : function(){

*// código*

}

};

*/\* accediendo a una propiedad de nuestro objeto literal persona \*/*

objetoLiteral.saludo

Un ejemplo de un modulo usando un objeto literal.

var persona = {

*/\* definiendo propiedades \*/*

nombre : "adan",

edad : 33,

*/\* método simple \*/*

comer : function(){

console.log(this.nombre + " esta comiendo.");

}

};

*/\* accediendo al método comer de nuestro objeto literal persona \*/*

persona.comer();

**MÓDULO**

Un módulo es una unidad independiente funcional que forma parte de la estructura de una aplicación.  
Podemos usar funciones y closures(cierres) para crear módulos.

var modulo = (function(){

*//- - -*

});

Un ejemplo más completo:

var automovil = (function(colorDeAuto){

var color = colorDeAuto;

return{

avanzar : function(){

console.log("el auto "+ color +" esta avanzando");

},

retroceder : function(){

console.log("el auto "+ color +" esta retrocediendo");

}

}

})("azul");

*/\* accediendo los metodos retroceder y avanzar de nuestro módulo \*/*

automovil.retroceder();

automovil.avanzar();

**FUNCIÓN ANÓNIMA**

Las funciónes anónimas son funciónes sin nombre, comúnmente asociados a una variable.

var miFuncionAnonima = function(){

alert("Hola mundo!");

};

miFuncionAnonima();

**FUNCIÓNES AUTO-EJECUTABLES (IIFE)**

Estas funciónes una vez declaradas se llaman a sí mismas para inicializarse, los primeros paréntesis encierran el contenido, los segundos paréntesis asumen que el interior de los primeros paréntesis es una función y la ejecuta inmediatamente.

*/\* 01 \*/*

(function(){

alert("Hola mundo!");

})();

*/\* 02 \*/*

var miFuncionAnonima = function(){

alert("Hola mundo!");

};

*/\* 03 \*/*

var miFuncionAnonima = (function(mensaje){

alert(mensaje);

})("Hola mundo");

*// todo lo que le precede a los 2 últimos paréntesis se ejecuta inmediatamente*

**CLOSURE**

Los clousures son funciones definidas dentro de otras funciones, así mismo dicha función interna tiene acceso al ámbito de la función contenedora.

function ejemploClousure(arg1, arg2){

var localVar = 8;

function multiplicador(interArg){

return arg1 \* arg2 \* interArg \* localVar;

}

*/\* retornar una referencia de la función interna definida como:*

*multiplicador*

*\*/*

return multiplicador;

}

*/\* la función devuelve una función, por lo tanto necesita asignación \*/*

var globalVar = ejemploClousure(2,4);

*/\* y luego llamar a \*/*

globalVar(8);

**MÉTODOS PRIVADOS**

Los métodos privados son funciones que no pueden ser llamados desde fuera del ámbito donde han sido declarados, dichos métodos podrán ser invocados en nuestros métodos públicos.

var modulo = (function () {

var privateMethod = function (message1) {

console.log(message1);

};

var publicMethod = function (mensaje2) {

privateMethod(mensaje2);

};

return {

publicMethod: publicMethod

};

})();

*/\* pasando datos a un método privado \*/*

modulo.publicMethod("mi mensaje");

**ENTENDIENDO EL RETORNO**

Comúnmente los módulos retornan un objeto, la cual los métodos ligados a dicho objeto serán accesibles desde fuera del módulo.

var module = (function(){

*/\* simple método privado \*/*

var privateMethod = function(){

console.log("soy un método privado");

};

*/\* retornando un objeto literal \*/*

return{

publicMethod : function(){

privateMethod();

console.log("soy un método publico");

}

}

})();

*/\* accediendo nuestro método publico \*/*

module.publicMethod();

**VENTAJAS DEL PATRON MODULAR**

* Código limpio , separado y organizado.
* Soportan datos privados.
* Código Escalable.

Bien, ahora que ya tenemos los algunos conceptos definidos vamos a realizar un ejercicio la cual sera realizar tabs, con una estructura mas avanzada, vamos a realizarlo con jquery.

**1. DEFINIMOS NUESTRA ESTRUCTURA HTML**

<ul class="tabs">

<li class="tabs\_item">

<a class="tabs\_item\_target is\_selected" data-tab="tab\_one">JavaScript</a>

</li>

<li class="tabs\_item">

<a class="tabs\_item\_target" data-tab="tab\_two">Html5</a>

</li>

<li class="tabs\_item">

<a class="tabs\_item\_target" data-tab="tab\_three">Css</a>

</li>

</ul>

<div class="tabs\_content">

<div id="tab\_one" class="tabs\_content\_pane is\_active">

contenido JavaScript

</div>

<div id="tab\_two" class="tabs\_content\_pane">

contenido Html5

</div>

<div id="tab\_three" class="tabs\_content\_pane">

contenido Css

</div>

</div>

**2. DEFINIMOS NUESTRA ESTRUCTURA JAVASCRIPT**

A continuación se muestra una estructura modular mas avanzada y explicaremos paso a paso las partes que la componen.

*// Definimos la variable tabs la cual contendrá todo nuestro modulo.*

var tabs = (function(){

*// Objeto la cual establecemos valores que vamos a usar mas adelante en este ámbito.*

var st = {

*//- - -*

};

*// Objeto vacío que guardará elementos que se manejan por HTML.*

var dom = {}

*// Función que llenará al objeto dom con los objetos HTML a través de jQuery ($).*

var catchDom = function(){

*//- - -*

};

*// Función donde establecemos los eventos que tendrán cada elemento.*

var suscribeEvents = function(){

*//- - -*

};

*/\* Objeto que guarda métodos que se van a usar en cada evento definido*

*en la función suscribeEvents. \*/*

var events = {

*//- - -*

};

*// Función que inicializará los funciones decritas anteriormente.*

var initialize = function(){

*//- - -*

};

*/\* Retorna un objeto literal con el método init haciendo referencia a la*

*función initialize. \*/*

return{

init:initialize

}

})();

*// Ejecutando el método "init" del módulo tabs.*

tabs.init();

**3. CREACIÓN DE NUESTRO MODULO**

Finalmente luego de haber entendido nuestra estructura anterior, procedemos a la creación de nuestro modulo tabs.

var tabs = (function() {

var st = {

tabs: '.tabs',

tabs\_item: '.tabs\_item',

tabs\_item\_target: '.tabs\_item\_target'

};

var dom = {}

var catchDom = function() {

dom.tabs\_item\_target = $(st.tabs\_item\_target, st.tabs);

};

var suscribeEvents = function() {

dom.tabs\_item\_target.on('click', events.eSelectedTab);

};

var events = {

eSelectedTab: function(e) {

self = $(this);

self.addClass('is\_selected');

var brother = self.parent(st.tabs\_item).siblings();

$(st.tabs\_item\_target, brother).removeClass('is\_selected');

var pane = self.attr('data-tab');

$("#" + pane).addClass('is\_active');

$("#" + pane).siblings().removeClass('is\_active');

}

};

var initialize = function() {

catchDom();

suscribeEvents();

};

return {

init: initialize

}

})();

tabs.init();

Gracias por haber leido este post, [aquí](https://github.com/wilsson/module-pattern-javascript) dejo el enlace al repositorio para que puedas jugar con el. Escríbenos sobre cualquier duda o consulta que tengas, que nosotros estaremos gustosos a ayudarte.

Sacado de: <http://frontendlabs.io/2643--patron-modular-en-javascript>