Parts

# Piezas

Podemos componer objetos a partir de conjuntos de partes. Por ejemplo, podemos crear una función que añada funciones sencillas de procesamiento de eventos a cualquier objeto. Añade un método on, un método firey un registro privado de eventos:

var eventuality = function (that) {  
 var registry = {};  
  
 that.fire = function (event) {  
  
// Fire an event on an object. The event can be either  
// a string containing the name of the event or an  
// object containing a type property containing the  
// name of the event. Handlers registered by the 'on'  
// method that match the event name will be invoked.  
  
 var array,  
 func,  
 handler,  
 i,  
 type = typeof event === 'string' ? event : event.type;  
  
// If an array of handlers exist for this event, then  
// loop through it and execute the handlers in order.  
  
 if (registry.hasOwnProperty(type)) {  
 array = registry[type];  
 for (i = 0; i < array.length; i += 1) {  
 handler = array[i];  
  
// A handler record contains a method and an optional  
// array of parameters. If the method is a name, look  
// up the function.  
  
 func = handler.method;  
 if (typeof func === 'string') {  
 func = this[func];  
 }  
  
// Invoke a handler. If the record contained  
// parameters, then pass them. Otherwise, pass the  
// event object.  
  
 func.apply(this,  
 handler.parameters || [event]);  
 }  
 }  
 return this;  
 };  
  
 that.on = function (type, method, parameters) {  
  
// Register an event. Make a handler record. Put it  
// in a handler array, making one if it doesn't yet  
// exist for this type.  
  
 var handler = {  
 method: method,  
 parameters: parameters  
 };  
 if (registry.hasOwnProperty(type)) {  
 registry[type].push(handler);  
 } else {  
 registry[type] = [handler];  
 }  
 return this;  
 };  
 return that;  
};

Podríamos llamar a eventuality en cualquier objeto individual, dotándolo de métodos de tratamiento de eventos. También podríamos llamarlo en una función constructora antes de que se devuelva that:

eventuality(that);

De este modo, un constructor podría ensamblar objetos a partir de un conjunto de partes. La tipificación laxa de JavaScript es una gran ventaja en este caso, porque no estamos agobiados por un sistema de tipos preocupado por el linaje de las clases. En su lugar, podemos centrarnos en el carácter de su contenido.

Si quisiéramos que eventuality tuviera acceso al estado privado del objeto, podríamos pasarle el paquete my.