



# *JavaScript*

## Sesión 7 - jQuery Avanzado



## Índice

- Efectos
- AJAX
- Utilidades
- Plugins
- Rendimiento



## 7.1 Efectos

- Facilita el uso de animaciones y efectos
- Mostrar y ocultar elementos
- Efectos de aparecer y desvanecer elementos
- Mover elementos a través de la pantalla
- <http://api.jquery.com/category/effects/>



## Mostrar y Ocultar

Método	Propósito
<b>show()</b>	Muestra cada elemento del conjunto de resultados, si estaban ocultos
<b>show(velocidad[,callback])</b>	Muestra todos los elementos del conjunto de resultados mediante una animación, y opcionalmente lanza un <i>callback</i> tras completar la animación
<b>hide()</b>	Oculto cada elemento del conjunto de resultados, si estaban visibles
<b>hide(velocidad[,callback])</b>	Oculto todos los elementos del conjunto de resultados mediante una animación, y opcionalmente lanza un <i>callback</i> tras completar la animación
<b>toggle()</b>	Cambia la visualización (visible u oculto, de manera contraria a su estado) para cada elemento del conjunto de resultados
<b>toggle(checkbox)</b>	Cambia la visualización para cada elemento del conjunto de resultados dependiendo del <i>checkbox</i> (verdadero muestra todos los elementos, falso para ocultarlos)
<b>toggle(velocidad[,callback])</b>	Cambia la visualización de todos los elementos del conjunto de resultados mediante una animación, y opcionalmente lanza un <i>callback</i> tras completar la animación



## Velocidad de Animación

- Valor numérico → cantidad de milisegundos de duración del efecto
- Cadena → `slow` (lento - 600ms), `normal` (400ms) o `fast` (rápido - 200ms).

```
$(function() {  
  $("#mostrar").click(function() {  
    $("#laCapa").show("normal");  
  });  
  $("#ocultar").click(function() {  
    $("#laCapa").hide(2000); // ms  
  });  
  $("#cambiar").click(function() {  
    $("#laCapa").toggle("slow");  
  });  
});
```

<http://jsbin.com/zakaga/1/edit?html,css,js,output>



## Aparecer y desvanecer

Método	Propósito
<b><code>fadeIn(velocidad[,callback])</code></b>	El contenido aparece a la <i>velocidad</i> indicada para cada elemento del conjunto de resultados, y opcionalmente lanza un <i>callback</i> tras completar la animación
<b><code>fadeOut(velocidad[,callback])</code></b>	El contenido se desvanece a la <i>velocidad</i> indicada para cada elemento del conjunto de resultados, y opcionalmente lanza un <i>callback</i> tras completar la animación
<b><code>fadeTo(velocidad,opacidad[,callback])</code></b>	El contenido cambia a la <i>opacidad</i> y <i>velocidad</i> indicadas para cada elemento del conjunto de resultados, y opcionalmente lanza un <i>callback</i> tras completar la animación



## Ejemplo aparecer y desvanecer

```
$(function() {  
  $("#aparecer").click(function() {  
    $("#laCapa").fadeIn(300);  
  });  
  $("#desvanecer").click(function() {  
    $("#laCapa").fadeOut("normal");  
  });  
  $("#fade03").click(function() {  
    $("#laCapa").fadeTo("slow", 0.3); // opacidad 0,3  
  });  
  $("#fade10").click(function() {  
    $("#laCapa").fadeTo("slow", 1.0);  
  });  
});
```

<http://jsbin.com/holumo/1/edit?html,css,js,output>



## Uso de callbacks

- Si a las animaciones le pasamos una función *callback*, se ejecutará al terminar la animación

```
$(this).fadeOut(1000, function() {  
    console.log("He acabado");  
});
```

```
$(this).fadeOut(500, function() {  
    $(this).delay(2000).fadeIn(500);  
});
```

- Si hubiésemos puesto la función fuera del parámetro del `fadeOut`, se hubiese ejecutado justamente después de iniciarse la animación, y no al finalizar la misma.

```
$(this).fadeOut(500).css("margin", "50 px");
```





## Enrollar y desenrollar (persiana)

Método	Propósito
<b>slideDown(velocidad[,callback])</b>	El contenido se desenrolla a la <i>velocidad</i> indicada modificando la altura de cada elemento del conjunto de resultados , y opcionalmente lanza un <i>callback</i> tras completar la animación
<b>slideUp(velocidad[,callback])</b>	El contenido se enrolla a la <i>velocidad</i> indicada modificando la altura de cada elemento del conjunto de resultados , y opcionalmente lanza un <i>callback</i> tras completar la animación
<b>slideToggle(velocidad[,callback])</b>	Cambia la visualización del contenido enrollando o desenrollando el contenido a la <i>velocidad</i> indicada modificando la altura de cada elemento del conjunto de resultados, y opcionalmente lanza un <i>callback</i> tras completar la animación



## Ejemplo enrollar y desenrollar

```
$(function() {  
    $("#enrollar").click(function() {  
        $("#laCapa").slideUp("normal");  
    });  
    $("#desenrollar").click(function() {  
        $("#laCapa").slideDown(2000);  
    });  
    $("#cambiar").click(function() {  
        $("#laCapa").slideToggle("slow");  
    });  
});
```

<http://jsbin.com/hunoq/1/edit?html,css,js,output>



## Creando animaciones

Método	Propósito
<b><i>animate</i>(<i>parámetros</i>, <i>duración</i>, <i>easing</i>, <i>callback</i>)</b>	Crea una animación personalizada donde <i>parámetros</i> indica un objeto CSS con las propiedades a animar, con una <i>duración</i> y <i>easing</i> (linear o <i>swing</i> ) determinados y lanza un <i>callback</i> tras completar la animación
<b><i>animate</i>(<i>parametros</i>, <i>opciones</i>)</b>	Crea una animación personalizada donde <i>parametros</i> indica las propiedades a animar y las <i>opciones</i> de las animación ( <i>complete</i> , <i>step</i> , <i>queue</i> )
<b><i>stop</i>()</b>	Detiene todas las animaciones en marcha para todos los elementos



## Ejemplo animaciones

```
$(function() {  
  $("#derecha").click(function() {  
    $("#laCapa").animate({ width: "500px" }, 1000);  
  });  
  $("#texto").click(function() {  
    $("#laCapa").animate({ fontSize: "24pt" }, 1000);  
  });  
  $("#mover").click(function() {  
    $("#laCapa").animate({ left: "500" }, 1000, "swing");  
  });  
  $("#todo").click(function() {  
    $("#laCapa").animate({ width: "500px", fontSize: "24pt", left: "500" },  
      1000, "swing");  
  });  
});
```

<http://jsbin.com/wiyodo/2/edit?html,css,js,output>



## Carrusel

```
<div id="carrusel">
  <div class="actual"></div>
  <div></div>
  <div></div>
  <div></div>
</div>
```

```
$(function () {
  setInterval("carruselImagenes()", 2000);
});

function carruselImagenes() {
  var fotoActual = $('#carrusel div.actual');
  var fotoSig = fotoActual.next();
  if (fotoSig.length == 0) {
    fotoSig = $('#carrusel div:first');
  }
  fotoActual.removeClass('actual').addClass('anterior');
  fotoSig.css({ opacity: 0.0 }).addClass('actual')
    .animate({ opacity: 1.0 }, 1000,
      function () { fotoActual.removeClass('anterior'); });
}
```





## Deshabilitando efectos

- Si el sistema donde corre nuestra aplicación es poco potente, podemos deshabilitar todos los efectos y animaciones haciendo uso de la propiedad booleana `jQuery.fx.off`
- Los elementos aparecerán y desaparecerán sin ningún tipo de animación.

```
$.fx.off = true;  
// Volvemos a activar los efectos  
$.fx.off = false;
```



## 7.2 AJAX

- <http://api.jquery.com/category/ajax/>
- `$.ajax()`

Método	Propósito
<code>selector.load(url)</code>	Incrustra el contenido crudo de la <i>url</i> sobre el <i>selector</i>
<code>\$.get(url, callback, tipoDatos)</code>	Realiza una petición GET a la <i>url</i> . Una vez recuperada la respuesta, se invocará el <i>callback</i> el cual recuperará los datos del <i>tipoDatos</i>
<code>\$.getJSON(url, callback)</code>	Similar a <code>\$.get()</code> pero recuperando datos en formato JSON
<code>\$.getScript(url, callback)</code>	Carga un archivo <i>JavaScript</i> de la <i>url</i> mediante un petición GET, y lo ejecuta.
<code>\$.post(url, datos, callback)</code>	Realiza una petición POST a la <i>url</i> , enviando los datos como parámetros de la petición





## `$.ajax()`

- Recibe como parámetro un objeto

Propiedad	Propósito
<b>url</b>	URL a la que se realiza la petición
<b>type</b>	Tipo de petición (GET o POST)
<b>dataType</b>	Tipo de datos que devuelve la petición, ya sea texto o binario.
<b>success</b>	<i>Callback</i> que se invoca cuando la petición ha sido exitosa y que recibe los datos de respuesta como parámetro
<b>error</b>	<i>Callback</i> que se invoca cuando la petición ha fallado
<b>complete</b>	<i>Callback</i> que se invoca cuando la petición ha finalizado





## Ejemplo `$.ajax()`

```
$.ajax({  
  url: "fichero.txt",  
  type: "GET",  
  dataType: "text",  
  success: todoOK,  
  error: fallo,  
  complete: function(xhr, estado) {  
    console.log("Petición finalizada " + estado);  
  }  
});
```

```
function todoOK(datos) {  
  console.log("Todo ha ido bien " + datos);  
}
```

```
function fallo(xhr, estado, msjErr) {  
  console.log("Algo ha fallado " + msjErr);  
}
```

```
function todoOK(datos) {  
  $("#resultado").append(datos);  
}
```



## selector.**load**(url)

- Permite incrustar contenido vía una URL

- Similar a `include` estático JSP `$( "body" ).load( "contacto.html" );`

- Podemos filtrar el contenido indicando un selector

```
$( "body" ).load( "contacto.html .contenido" );
```



```
<a href="contacto.html">Contacto</a>
<div id="contenedor"></div>

<script>
$( 'a' ).on( 'click', function( evt ) {
    var href = $( this ).attr( 'href' );
    $( '#contenedor' ).load( href + ' .contenido' );
    evt.preventDefault();
});
</script>
```



## \$.get()

- Petición GET

- `$.get(url [,datos], callback(datosRespuesta) [,tipoDatosRespuesta])`

```
$.get("fichero.txt", function(datos) {  
    console.log(datos);  
});
```

- Permite trabajar con XML

```
<hero>  
  <nombre>Batman</nombre>  
  <email>batman@heroes.com</email>  
</hero>
```

```
$.get("heroes.xml", function(datos) {  
    var nombre = datos.getElementsByTagName("nombre")[0];  
    var email = datos.getElementsByTagName("email")[0];  
    var val = nombre.firstChild.nodeValue + " " + email.firstChild.nodeValue;  
    $("#resultado").append(val);  
}, "xml");
```



## `$.getJSON()`

- Espera que la petición GET devuelva contenido JSON
- `$.getJSON(url [,datos], callback(datosRespuesta))`

```
var flickrAPI = "http://api.flickr.com/services/feeds/photos_public.gne?jsoncallback=?";
$.getJSON( flickrAPI, {
  tags: "proyecto víbora ii",
  tagmode: "any",
  format: "json"
}, formateaImágenes);

function formateaImágenes(datos) {
  $.each(datos.items, function(i, elemento) {
    $("<img>").attr("src", elemento.media.m).appendTo("#contenido");
    if (i === 4) {
      return false;
    }
  });
}
```

<http://jsbin.com/hunape/1/edit?html,js,output>



## `$.each()`

- Método auxiliar de jQuery para iterar sobre una colección.
  - un array mediante `$.each(colección, callback(índice, elemento))`
  - un conjunto de selectores con `$(selector).each(callback(índice, elemento))`
  - las propiedades de un objeto mediante `$.each(objeto, callback(clave, valor))`
- Se emplea mucho al trabajar con AJAX para recorrer los datos obtenidos.
- <http://api.jquery.com/jquery.each/>



## `$.getScript()`

- `$.getScript(urlScript, callback)`
- Permite inyectar código *al vuelo*
- Recupera un archivo JavaScript y lo ejecuta a continuación.
- Una vez finalizada la ejecución del script, se invocará al callback.

```
// script.js  
console.log("Ejecutado dentro del script");  
$("#resultado").html("<strong>getScript</strong>");
```

```
$.getScript("script.js", function(datos, statusTxt) {  
    console.log(statusTxt);  
})
```



## \$.post()

- `$.post(url, datos, callback(datosRespuesta))`
- Modos de adjuntar los datos:
  - Creando un objeto cuyos valores obtenemos mediante `val()`.
  - Serializando el formulario mediante el método `serialize()`, el cual codifica los elementos del formulario mediante una cadena de texto

```
<form name="formCliente" id="frmClnt" action="#">
<fieldset id="infoPersonal">
  <legend>Datos Personales</legend>
  <p><label for="nombre">Nombre</label>
    <input type="text" name="nombre" id="idNombre" /></p>
  <p><label for="correo">Email</label>
    <input type="email" name="correo" id="idEmail" /></p>
</fieldset>
<p><button type="submit">Guardar</button></p>
</form>
```

```
$( 'form' ).on( 'submit', function( evt ) {
  evt.preventDefault();
  // var nom = $(this).find( '#inputName' ).val();
  var datos = $(this).serialize(); // nombre=asdf&email=asdf
  $.post( "/GuardaFormServlet", datos, function ( respuestaServidor ) {
    console.log( "Completado " + respuestaServidor );
  });
});
```





## Tipos de datos

- Fragmentos **HTML** → mediante `load()` podemos cargarlos sin necesidad de ejecutar ningún callback.
  - Los datos puede que no tengan ni la estructura ni el formato que necesitemos
  - Acopla nuestro contenido con el externo.
- Archivos **JSON** → permiten estructurar la información para su reutilización. Compactos y fáciles de usar, donde la información es auto-explicativa y se puede manejar mediante objetos mediante `JSON.parse()` y `JSON.stringify()`.
  - Hay que vigilar los errores en el contenido de los archivos ya que pueden provocar efectos colaterales.
- Archivos **JavaScript** → ofrecen flexibilidad pero no son realmente un mecanismo de almacenamiento, ya que no podemos usarlos desde sistemas heterogéneos.
  - La carga de scripts JavaScripts en caliente permite refactorizar el código en archivos externos, reduciendo el tamaño del código hasta que sea necesario.
- Archivos **XML** → han perdido mercado en favor de JSON
  - Se sigue utilizando para permitir que sistemas de terceros sin importar la tecnología de acceso puedan conectarse a nuestros sistemas.





## Eventos AJAX

- Métodos globales para interactuar con los eventos que se lanzan al realizar una petición AJAX.
- No los llamamos dentro de la aplicación, sino que es el navegador el que realiza las llamadas.

Método	Propósito
<b>ajaxComplete()</b>	Registra un manejador que se invocará cuando la petición AJAX se complete
<b>ajaxError()</b>	Registra un manejador que se invocará cuando la petición AJAX se complete con un error
<b>ajaxStart()</b>	Registra un manejador que se invocará cuando la primera petición AJAX comience
<b>ajaxStop()</b>	Registra un manejador que se invocará cuando todas las peticiones AJAX hayan finalizado
<b>ajaxSend()</b>	Adjunta una función que se invocará antes de enviar la petición AJAX
<b>ajaxSuccess()</b>	Adjunta una función que se invocará cuando una petición AJAX finalice correctamente



## Ejemplo eventos

<http://jsbin.com/loqace/5/edit?js,console>

```
AJAX comenzando
Antes de enviar la información...
▶XHR finished loading: GET "http://localhost:63342/Pruebas/jquery-ajax/fichero.txt".
Datos recibidos y adjuntándolos a resultado
Parece que ha funcionado todo!
Todo ha finalizado!
AJAX petición finalizada
```

```
$( "document" ).ready( function() {
    $(document).ajaxStart( function () {
        console.log( "AJAX comenzando" );
    });
    $(document).ajaxStop( function () {
        console.log( "AJAX petición finalizada" );
    });
    $(document).ajaxSend( function () {
        console.log( "Antes de enviar la información..." );
    });
    $(document).ajaxComplete( function () {
        console.log( "Todo ha finalizado!" );
    });
    $(document).ajaxError( function (evt, jqXHR, settings, err) {
        console.log( "Houston, tenemos un problema: " + err );
    });
    $(document).ajaxSuccess( function () {
        console.log( "Parece que ha funcionado todo!" );
    });
    getDatos();
});
```

```
function getDatos() {
    $.getJSON( "http://www.omdbapi.com/?s=batman&callback=?", todoOk );
}

function todoOk( datos ) {
    console.log( "Datos recibidos y adjuntándolos a resultado" );
    $( "#resultado" ).append( JSON.stringify( datos ) );
}
```



## 7.3 Utilidades - Comprobación de tipos

- jQuery ofrece un conjunto de utilidades como funciones globales → `$.funcion()`

Función	Propósito
<code>\$.isArray(array)</code>	Determina si <i>array</i> es un <i>Array</i> . Si es un objeto <i>array-like</i> devolverá falso
<code>\$.isFunction(función)</code>	Determina si <i>función</i> es una <i>Función</i>
<code>\$.isEmptyObject(objeto)</code>	Determina si <i>objeto</i> esta vacío
<code>\$.isPlainObject(objeto)</code>	Determina si <i>objeto</i> es un objeto sencillo, creado como un objeto literal (mediante las llaves) o mediante <code>new objeto</code> .
<code>\$.isXmlDoc(documento)</code>	Determina si el <i>documento</i> es un documento XML o un nodo XML
<code>\$.isNumeric(objeto)</code>	Determina si <i>objeto</i> es un valor numérico escalar.
<code>\$.isWindow(objeto)</code>	Determina si <i>objeto</i> representa una ventana de navegador
<code>\$.type(objeto)</code>	Obtiene la clase <i>Javascript</i> del <i>objeto</i> . Los posibles valores son <code>boolean</code> , <code>number</code> , <code>string</code> , <code>function</code> , <code>array</code> , <code>date</code> , <code>regexp</code> , <code>object</code> , <code>undefined</code> o <code>null</code>



## Manipulando Colecciones

Función	Propósito
<b><code>\$.makeArray(objeto)</code></b>	Convierte el <i>objeto</i> en un <i>array</i> . Se utiliza cuando necesitamos llamar a funciones que sólo soportan los <i>arrays</i> , como <code>join</code> o <code>reverse</code> , o cuando necesitamos pasar un parametro a una función como <i>array</i>
<b><code>\$.inArray(valor, array)</code></b>	Determina si el <i>array</i> contiene el <i>valor</i> . Devuelve <code>-1</code> o el índice que ocupa el <i>valor</i> . Un tercer parámetro opcional permite indicar el índice por el cual comienza la búsqueda.
<b><code>\$.unique(array)</code></b>	Elimina cualquier elemento duplicado que se encuentre en el <i>array</i>
<b><code>\$.merge(array1, array2)</code></b>	Combina los contenidos de <i>array1</i> y <i>array2</i>
<b><code>\$.map(array, callback)</code></b>	Construye un nuevo array cuyo contenido es el resultado de llamar al <i>callback</i> para cada elemento
<b><code>\$.grep(array, callback [,invertido])</code></b>	Filtra el <i>array</i> mediante el <i>callback</i> , de modo que añadirá los elementos que pasen la función, la cual recibe un objeto DOM como parámetro, y devuelve un <i>array JavaScript</i>



## Copiando Objetos

- `$.extend([boolRecursivo,] destino, origen)`
- Permite copiar miembros de un objeto fuente en uno destino, sin realizar herencia, sólo clonado las propiedades.
- Si hay un conflicto, se sobrescribirán con las propiedades del objeto fuente, y si tenemos múltiples objetos fuentes, de izquierda a derecha.
- Si los objetos a clonar contienen objetos anidados → primer parámetro a `true`
- <http://api.jquery.com/jquery.extend/>

```
var animal = {
  comer: function() {
    console.log("Comiendo");
  }
}

var perro = {
  ladrar: function() {
    console.log("Ladrando");
  }
}

$.extend(perro, animal);
perro.comer(); // Comiendo
```





## Ejemplo `$.extend()`

```
var perroCopia = {};  
  
$.extend(perroCopia, perro);  
perroCopia.acciones.ladrar(); // Ladrando  
  
$.extend(true, perroCopia, animal);  
perroCopia.acciones.comer(); // Comiendo  
perroCopia.acciones.ladrar(); // Ladrando  
  
$.extend(perro, animal);  
perro.acciones.comer(); // Comiendo  
perro.acciones.ladrar(); // error
```

```
var animal = {  
  acciones: {  
    comer: function() {  
      console.log("Comiendo");  
    },  
    sentar: function() {  
      console.log("Sentando");  
    }  
  }  
};  
  
var perro = {  
  acciones: {  
    ladrar: function() {  
      console.log("Ladrando");  
    },  
    cavar: function() {  
      console.log("Cavando");  
    }  
  }  
};
```



## 7.4 Plugins

- *jQuery* soporta una arquitectura de plugins para extender la funcionalidad de la librería.
- Repositorio → <http://plugins.jquery.com/>,
  - Listado con demos, código de ejemplo y tutoriales para facilitar su uso.
  - jQuery UI → plugins con controles (botones, calendario, menús, pestañas, acordeón, etc...)
- En ocasiones necesitamos escribir nuestro propio código que podemos empaquetar como un nuevo plugin.
  - Nombrado: `jquery.nombrePlugin.js` o `jquery.nombrePlugin-1.0.js`
  - En el fichero, poner un comentario con la versión



## Creando un *plugin*

- A la hora de crear un plugin, se asume que *jQuery* ha cargado.
- No asumir que el alias `$` esté disponible.
- Dentro del plugin se recomienda utilizar el nombre completo `jQuery` o redefinir el `$`.
- Podemos hacer uso de una IIFE para poder usar el `$` dentro de nuestro plugin.

```
(function($) {  
    // código del plugin  
})(jQuery);
```





## Funciones globales

- Similares a `$.ajax()`
- No necesitan ningún objeto `jQuery` para funcionar
- Amplian el abanico de funciones de utilidades.
- Para añadir una función al espacio de nombre de `jQuery` → asignar la función como una propiedad del objeto `jQuery`:

```
(function($) {  
  $.suma = function(array) {  
    var total = 0;  
    $.each(array, function (indice, valor) {  
      valor = $.trim(valor);  
      valor = parseFloat(valor) || 0;  
  
      total += valor;  
    });  
    return total;  
  };  
})(jQuery);
```

```
var resultado = $.suma([1,2,3,4]); // 10
```



## Espacio de nombres

- Hay que evitar el conflicto de nombres.
- Asociar las funciones a un objeto.
- El objeto contiene como propiedades las funciones que queremos añadir al plugin.

```
(function($) {  
  $.MathUtils = {  
    suma : function(array) {  
      // código de la función  
    },  
    media: function(array) {  
      // código de la función  
    }  
  };  
})(jQuery);
```

```
var resultado = $.MathUtils.suma([1,2,3,4]);  
var resultado = $.MathUtils.media([1,2,3,4]);
```



## Métodos de objeto

- Para extender las funciones de *jQuery*, mediante prototipos podemos crear métodos nuevos que se apliquen al objeto `jQuery` activo.
- *jQuery* utiliza el alias `fn` en vez `prototype`.

```
$.fn.nombreNuevaFuncion = function() {  
    // código nuevo  
}
```

- Al no saber si la función trabajará sobre un sólo objeto o sobre una colección (un selector puede devolver cero, uno o múltiples elementos.), hay que plantear un escenario donde recibimos un array de datos.
- Solución → iterar sobre la colección mediante el método `each()`.

```
$.fn.nombreNuevaFuncion = function() {  
    this.each(function() {  
        // Hacemos algo con cada elemento  
    });  
};
```



## Encadenar funciones

- Al crear una función prototipo después de llamar a nuestra función, es posible que el desarrollador quiera seguir encadenando llamadas.
- La función tiene que devolver un objeto `jQuery` para permitir que continúe el encadenamiento.
  - Normalmente es `this`.

```
(function($) {  
    $.fn.cambiarClase = function(clase1, clase2) {  
        return this.each(function() {  
            var elem = $(this);  
            if (elem.hasClass(clase1)) {  
                elem.removeClass(clase1).addClass(clase2);  
            } else if (elem.hasClass(clase2)) {  
                elem.removeClass(clase2).addClass(clase1);  
            }  
        });  
    };  
})(jQuery);
```



## Opciones

- Una buena práctica es permitir que el plugin reciba un objeto con las opciones de configuración.

```
$.fn.pluginConConf = function(opciones) {  
  var confFabrica = {prop: "valorPorDefecto"};  
  var conf = $.extend(confFabrica, opciones);  
  
  return this.each(function() {  
    // código que trabaja con conf.prop  
  });  
};
```

- Para permitir modificar los valores de fabrica → extraer a una propiedad del método

```
$.fn.pluginConConf = function(opciones) {  
  var conf = $.extend({}, $.fn.pluginConConf.confFabrica, opciones);  
  
  return this.each(function() {  
    // código que trabaja con conf.prop  
  });  
};  
$.fn.pluginConConf.confFabrica = {prop: "valorPorDefecto"};
```



## 7.5 Rendimiento - Consejos

- Consejo N° 0: A ser posible, **siempre es mejor utilizar *JavaScript* "puro" a *jQuery***
  - *jQuery* envuelve a *JavaScript*, y al ofrecer *Cross-Browser* hay tareas sencillas que las hace más costosas computacionalmente
- Consejo N° 1: Utilizar la **última versión de *jQuery***, ya que siempre contienen mejoras de rendimiento que repercutirán en nuestra aplicación.
- Leer la '*release note*' antes de modificar la librería en producción



## Consejo N° 2 - Cachear los selectores

- Evitar búsquedas innecesarias.

```
console.time("Sin cachear");  
for (var i=0; i < 1000; i++) {  
    var s = $("div");  
}  
console.timeEnd("Sin cachear");
```

8.990 ms

```
console.time("Cacheando");  
var miCapa = $("div");  
for (var i=0; i < 1000; i++) {  
    var s = miCapa;  
}  
console.timeEnd("Cacheando");
```

0.045 ms

±200 veces menos





## Consejo N° 3 - Cachear otros elementos

- Cachear llamadas a métodos, propiedades
  - No usar `.length` en la condición de un `for` → Mejor al iniciar la variable o fuera del bucle
  - Cachear llamadas AJAX

```
function getDatos(id) {
  if (!api[id]) {
    var url = "http://www.omdbapi.com/?s=" + busqueda + "&callback=?";
    console.log("Petición a " + url);
    api[id] = $.getJSON(url);
  }

  api[id].done(todoOk).fail(function() {
    $("#resultado").html("Error");
  });
}

function todoOk(datos) {
  console.log("Datos recibidos y adjuntándolos a resultado");
  $("#resultado").append(JSON.stringify(datos));
}
```



## Consejo N° 4 - Utilizar propiedades de elementos

- Se basa en el consejo N° 0
- En vez de usar un método *jQuery*, acceder a la propiedad mediante *JavaScript*.
  - cuidado que al acceder mediante una propiedad el resultado deja de ser un objeto *jQuery*, con lo que no vamos a poder encadenar el resultado en una llamada posterior

```
var lento = miCapa.attr("id");  
var rapido = miCapa[0].id;
```



**¿Preguntas?**