

USAR INTERNA SOLAMENTE

SOPRA
Curso de Maquetación

jQuery

Versión 1.00 del viernes, 06 de mayo de 2016

Estado : Trabajo

Índice

1. Herramientas a utilizar en el curso	6
2. Qué es jQuery	7
2.1. Ventajas de jQuery con respecto a otras alternativas	7
3. Lo Básico	8
4. Los selectores	9
4.1. Elección de selectores	9
4.2. Comprobar selecciones	9
4.3. Guardar selecciones	10
4.4. Refinamiento y filtrado de selecciones	10
4.5. Selección de elementos de un formulario	10
4.6. Getters y Setters	11
5. Eventos	13
5.1. El objeto Event	13
5.2. El principio de propagación (bubbling)	14
5.3. Eventos globales o del DOM	15
5.3.1. Document Ready	15
5.3.2. Window load	15
5.3.3. Window resize	15
5.4. El scroll	16
5.4.1. Detectar si se está haciendo scroll en una página	16
5.4.2. El método scrollTop()	16
5.4.3. El método scrollLeft()	16
5.4.4. Animaciones suaves con el scroll	16
5.5. Manejadores de eventos	17
5.5.1. Eventos relacionados con el ratón	17
5.5.2. Eventos relacionados con el teclado	18
5.5.3. Eventos relacionados con el foco	18
5.6. Ejemplos	20
6. Funciones nominales y anónimas	21
6.1. Funciones nominales	21

6.2.	Funciones anónimas	21
7.	Métodos	22
7.1.	El Dream-Team de jQuery	22
7.2.	Métodos “suplentes” del Dream-Team	23
7.3.	Manipulación de elementos	24
7.3.1.	Obtener y establecer información en elementos	24
7.3.2.	Mover, copiar y eliminar elementos	25
7.3.3.	Crear nuevos elementos	26
7.3.4.	Manipulación de atributos	27
7.4.	El método data	28
7.6.	El método attr	30
7.7.	Otros métodos útiles	31
7.7.1.	\$.fn.CSS	31
7.7.2.	\$.trim	31
7.7.3.	\$.each	31
7.7.4.	\$.inArray	31
7.7.5.	\$.extend	31
7.7.6.	\$.isFunction	32
7.7.7.	\$.isPlainObject	32
7.7.8.	\$.isArray	32
7.7.9.	\$.proxy	32
8.	Efectos	34
8.1.	Efectos de transparencia (show y hide)	34
8.2.	Efectos de atenuación (fadein y fadeout)	34
8.3.	Efectos de deslizamiento (slideDown y slideout)	35
8.4.	Cambiar de efecto basado en estados (toggle)	35
8.5.	Añadiendo funcionalidades después del efecto	36
8.6.	Otros métodos para gestionar animaciones	36
8.7.	Efectos personalizados con animate	36
8.7.1.	Flexibilización	37
9.	Recorrer el DOM	38
10.	Observadores de Mutación	39
10.1.	El objeto MutationObserver	39
10.1.1.	Compatibilidad en navegadores	39
10.1.2.	Tipos de cambios del objeto mutationobserver	39

10.1.3.	Cómo usar el objeto <code>mutationobserver</code>	40
10.1.4.	Detener la observación	40
10.2.	El objeto <code>MutationRecord</code>	41
10.3.	Ejemplo básico	41
11.	Peticiones Ajax	42
11.1.	<code>\$.ajax</code>	42
11.1.1.	Utilizar el método <code>\$.ajax</code>	42
11.1.2.	Opciones del método <code>\$.ajax</code>	43
11.2.	Métodos convenientes	44
11.2.1.	Ejemplos	44
11.3.	<code>\$.fn.load</code>	45
11.3.1.	Ejemplos	45
11.4.	La especificación CORS	45
11.4.1.	Cómo funciona	45
11.4.2.	Cabeceras para la CORS	46
11.4.3.	Ejemplo	46
12.	Evitar conflictos con otras bibliotecas JavaScript	47
13.	Creación de Plugins	48
13.1.	Objetos a extender	48
13.1.1.	Ejemplo de definición jQuery	48
13.1.2.	Ejemplo de definición <code>jQuery.fn</code>	48
13.2.	El método <code>extend()</code>	49
13.3.	Estableciendo parámetros por defecto	49
13.4.	Creando un plugin con parámetros y valores por defecto	49
14.	Mejorar el rendimiento	50
14.1.	Guardar la longitud de los bucles	50
14.2.	Añadir Nuevo Contenido por Fuera de un Bucle	50
14.3.	Utilizar la Delegación de Eventos	50
14.4.	Asignar objetos a variables	51
14.5.	No abusar de <code>\$.fn.css</code>	51
14.6.	No abusar de <code>\$(this)</code>	51
14.7.	Utilizar <code>\$.data</code> en Lugar de <code>\$.fn.data</code>	51
14.8.	Evitar interacciones con elementos vacíos	52

14.9. evitar el uso de \$.each	52
15. Referencias	53

1. Herramientas a utilizar en el curso

Navegador **Firefox** con los plugins:

- Colorzilla
- Pixel Perfect
- Firebug
- Greasemonkey
- IE Tab V2
- Firesizer
- Status-4-Evar
- Senseo

Navegador **Chrome** con plugins:

- Colorzilla
- Window Resizer
- IE Tab
- Perfect pixel
- Meta SEO Inspector

2. Qué es jQuery

jQuery es una biblioteca de JavaScript, creada inicialmente por John Resig, que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web. Fue presentada el 14 de enero de 2006 en el BarCamp NYC. jQuery es la biblioteca de JavaScript más utilizada.¹

jQuery es software libre y de código abierto, posee un doble licenciamiento bajo la Licencia MIT y la Licencia Pública General de GNU v2, permitiendo su uso en proyectos libres y privados.² jQuery, al igual que otras bibliotecas, ofrece una serie de funcionalidades basadas en JavaScript que de otra manera requerirían de mucho más código, es decir, con las funciones propias de esta biblioteca se logran grandes resultados en menos tiempo y espacio.

Las empresas Microsoft y Nokia anunciaron que incluirán la biblioteca en sus plataformas.³ Microsoft la añadirá en su IDE Visual Studio⁴ y la usará junto con los frameworks ASP.NET AJAX y ASP.NET MVC, mientras que Nokia los integrará con su plataforma Web Run-Time.

2.1. VENTAJAS DE JQUERY CON RESPECTO A OTRAS ALTERNATIVAS

Es importante comentar que jQuery no es el único framework que existe en el mercado. Existen varias soluciones similares que también funcionan muy bien, que básicamente nos sirven para hacer lo mismo. Como es normal, cada uno de los frameworks tiene sus ventajas e inconvenientes, pero jQuery es un producto con una aceptación por parte de los programadores muy buena y un grado de penetración en el mercado muy amplio, lo que hace suponer que es una de las mejores opciones. Además, es un producto serio, estable, bien documentado y con un gran equipo de desarrolladores a cargo de la mejora y actualización del framework. Otra cosa muy interesante es la dilatada comunidad de creadores de plugins o componentes, lo que hace fácil encontrar soluciones ya creadas en jQuery para implementar asuntos como interfaces de usuario, galerías, votaciones, efectos diversos, etc.

Uno de los competidores de jQuery, del que hemos publicado ya en DesarrolloWeb.com un amplio manual para programadores, es Mootools, que también posee ventajas similares. Os dejo el enlace al Manual de Mootools, que también puede ser interesante, porque seguramente lo tengamos explicado con mayor detalle que jQuery.

3. Lo Básico

Como toda librería JavaScript debemos incluirla en nuestra página web. La podemos descargar desde [jQuery.com](http://code.jquery.com/jquery-latest.min.js), guardarlo en alguna carpeta e incluirlo localmente u otra manera más rápida es usando el hospedaje de terceros (CDN) como el mismo jQuery, Google, o Microsoft:

```
<script type="text/javascript" src="http://code.jquery.com/jquery-latest.min.js"></script>
<script type="text/javascript" src="https://ajax.googleapis.com/ajax/libs/jquery/1.4.4/jquery.min.js">
</script>
<script type="text/javascript" src="http://ajax.microsoft.com/ajax/jquery/jquery-1.4.4.min.js"></script>
```

Ya con la librería incluida dentro de nuestro documento podemos empezar a interactuar con jQuery abrimos un:

```
<script type="text/javascript">
// código JS o jQuery
</script>
```

4. Los selectores

La base de jQuery son los selectores, para los que están familiarizados con CSS, les será facilísimo ya que usa el mismo sistema de selectores que CSS para interactuar con los elementos HTML. Para seleccionar elementos se usa la función base que es representada con el símbolo `$` y encerrando entre paréntesis y comillas el elemento a seleccionar, ejemplo:

```
$("#table");  
$(".class");  
$("#id");  
$("a[href]");
```

4.1. ELECCIÓN DE SELECTORES

La elección de buenos selectores es un punto importante cuando se desea mejorar el rendimiento del código. Una pequeña especificidad — por ejemplo, incluir el tipo de elemento (como `div`) cuando se realiza una selección por el nombre de clase — puede ayudar bastante. Por eso, es recomendable darle algunas "pistas" a jQuery sobre en qué lugar del documento puede encontrar lo que desea seleccionar. Por otro lado, demasiada especificidad puede ser perjudicial. Un selector como `#miTabla thead tr th.especial` es un exceso, lo mejor sería utilizar `#miTabla th.especial`.

jQuery ofrece muchos selectores basados en atributos, que permiten realizar selecciones basadas en el contenido de los atributos utilizando simplificaciones de expresiones regulares.

```
// encontrar todos los <a> cuyo atributo rel terminan en "thinger"  
$("a[rel$='thinger']");
```

Estos tipos de selectores pueden resultar útiles pero también ser muy lentos. Cuando sea posible, es recomendable realizar la selección utilizando IDs, nombres de clases y nombres de etiquetas.

Si se desea conocer más sobre este asunto, Paul Irish realizó una gran [presentación sobre mejoras de rendimiento en JavaScript](#) (en inglés), la cual posee varias diapositivas centradas en selectores.

4.2. COMPROBAR SELECCIONES

Una vez realizada la selección de los elementos, se puede querer conocer si dicha selección entregó algún resultado. Para ello, se podría escribir:

```
if ($('#div.foo')) { ... }
```

Sin embargo esta forma no funcionará. Cuando se realiza una selección utilizando `$()`, siempre es devuelto un objeto, y si se lo evalúa, éste siempre devolverá `true`. Incluso si la selección no contiene ningún elemento, el código dentro del bloque `if` se ejecutará.

En lugar de utilizar el código mostrado, lo que se debe hacer es preguntar por la cantidad de elementos que posee la selección que se ejecutó. Esto es posible realizarlo utilizando la propiedad JavaScript `length`. Si la respuesta es 0, la condición evaluará falso, caso contrario (más de 0 elementos), la condición será verdadera.

Evaluar si una selección posee elementos

```
if ($('#div.foo').length) { ... }
```

4.3. GUARDAR SELECCIONES

Cada vez que se hace una selección, una gran cantidad de código es ejecutado. jQuery no guarda el resultado por sí solo, por lo tanto, si va a realizar una selección que luego se hará de nuevo, deberá salvar la selección en una variable. Contrariamente a otros lenguajes de programación, en JavaScript el signo `$` no posee ningún significado especial — es solamente otro carácter. Sin embargo, en jQuery, se suele utilizar para indicar que dicha variable posee un objeto jQuery.

Guardar selecciones en una variable

```
var $divs = $("div");
```

4.4. REFINAMIENTO Y FILTRADO DE SELECCIONES

A veces, puede obtener una selección que contiene más de lo que necesita; en este caso, es necesario refinar dicha selección. jQuery ofrece varios métodos para poder obtener exactamente lo que desea.

Refinamiento de selecciones

```
$('#div.foo').has('p');           // el elemento div.foo contiene elementos <p>  
$('h1').not('.bar');              // el elemento h1 no posee la clase 'bar'  
$('ul li').filter('.current');   // un item de una lista desordenada  
                                // que posee la clase 'current'  
$('ul li').first();              // el primer item de una lista desordenada  
$('ul li').eq(5);                // el sexto item de una lista desordenada
```

4.5. SELECCIÓN DE ELEMENTOS DE UN FORMULARIO

jQuery ofrece varios pseudo-selectores que ayudan a encontrar elementos dentro de los formularios, éstos son especialmente útiles ya que dependiendo de los estados de cada elemento o su tipo, puede ser difícil distinguirlos utilizando selectores CSS estándar.

- `:button` Selecciona elementos `<button>` y con el atributo `type='button'`
- `:checkbox` Selecciona elementos `<input>` con el atributo `type='checkbox'`
- `:checked` Selecciona elementos `<input>` del tipo `checkbox` seleccionados

- `:disabled` Selecciona elementos del formulario que están deshabilitados
- `:enabled` Selecciona elementos del formulario que están habilitados
- `:file` Selecciona elementos `<input>` con el atributo `type='file'`
- `:image` Selecciona elementos `<input>` con el atributo `type='image'`
- `:input` Selecciona elementos `<input>`, `<textarea>` y `<select>`
- `:password` Selecciona elementos `<input>` con el atributo `type='password'`
- `:radio` Selecciona elementos `<input>` con el atributo `type='radio'`
- `:reset` Selecciona elementos `<input>` con el atributo `type='reset'`
- `:selected` Selecciona elementos `<options>` que están seleccionados
- `:submit` Selecciona elementos `<input>` con el atributo `type='submit'`
- `:text` Selecciona elementos `<input>` con el atributo `type='text'`

Utilizando pseudo-selectores en elementos de formularios

```
// obtiene todos los elementos inputs dentro del formulario #myForm
$('#myForm :input');
```

4.6. GETTERS Y SETTERS

jQuery "sobrecarga" sus métodos, en otras palabras, el método para establecer un valor posee el mismo nombre que el método para obtener un valor. Cuando un método es utilizado para establecer un valor, es llamado método establecedor (en inglés *setter*). En cambio, cuando un método es utilizado para obtener (o leer) un valor, es llamado obtenedor (en inglés *getter*).

El método `$.fn.html` utilizado como establecedor

```
$('#h1').html('hello world');
```

El método `html` utilizado como obtenedor

```
$('#h1').html();
```

Los métodos establecedores devuelven un objeto jQuery, permitiendo continuar con la llamada de más métodos en la misma selección, mientras que los métodos obtenedores devuelven el valor por el cual se consultó, pero no permiten seguir llamando a más métodos en dicho valor.

Restablecer la selección original utilizando el método \$.fn.end

```
$('#content')  
  .find('h3')  
  .eq(2)  
    .html('nuevo texto para el tercer elemento h3')  
  .end() // resetea la selección anterior  
  .eq(0)  
    .html('nuevo texto para el primer elemento h3');
```

5. Eventos

jQuery permite enlazar o desenlazar manejadores de eventos a elementos del DOM sin tener que preocuparse por la implementación de cada explorador web.

Si se investiga un poco se puede comprobar que existen algunas funciones como `bind()`, `unbind()`, `delegate()`, `live()` que están en "desuso" y que han sido remplazadas por las funciones `on()` y `off()`.

La forma de declarar o enlazar eventos es:

```
$('#body').on('click', function(e) {  
    // Acciones al hacer "click" sobre el elemento body  
});
```

5.1. EL OBJETO EVENT

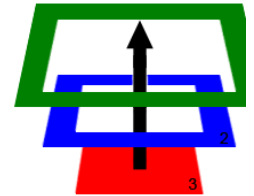
El objeto Event de jQuery complementa al de JavaScript y contiene varios métodos y propiedades interesantes.

Propiedad / Evento	Descripción
pageX, pageY	La posición del puntero del ratón en el momento que el evento ocurrió, relativo a las zonas superiores e izquierda de la página.
type	El tipo de evento (por ejemplo <code>click</code>).
which	El botón o tecla presionada.
data	Alguna información pasada cuando el evento es ejecutado.
target	El elemento DOM que inicializó el evento.
preventDefault()	Cancela la acción predeterminada del evento (por ejemplo: seguir un enlace).
stopPropagation()	Detiene la propagación del evento sobre otros elementos. En IE < 9 se sustituye por <code>cancelBubble = true false</code> .
isDefaultPrevented()	retorna <code>true</code> si <code>preventDefault()</code> ha sido invocado alguna vez por el objeto o selector del evento actual, <code>false</code> de lo contrario.
isPropagationStopped()	retorna <code>true</code> si <code>stopPropagation()</code> ha sido invocado alguna vez por el objeto o selector del evento actual, <code>false</code> de lo contrario.

5.2. EL PRINCIPIO DE PROPAGACIÓN (BUBBLING)

El principio fundamental de propagación o “burbujeo” en JavaScript afirma: **Después de que un evento se desencadene en el elemento más profundo posible, entonces se disparan en sus ancestros por orden de anidamiento.**

Dicho de otro modo, si tenemos 3 DIVs, cada uno dentro de otro, el primero en disparar un evento será el más profundo de los 3 (el nieto), después se disparará el evento superior (su padre) y después el siguiente superior (su abuelo).



Este proceso, normalmente, se realiza en 2 fases

- **CAPTURA** Hacia abajo (abuelo, su hijo, su nieto)
- **PROPAGACIÓN** Hacia arriba (nieto, su padre, su abuelo)

Este principio proporciona muchas ventajas y problemas a los desarrolladores ya que nos puede ayudar a controlar todo lo que sucede en la página pero también nos puede perjudicar. Por esta razón, existe un método para detener la propagación de eventos. Dependiendo del navegador lo tendremos que definir de una u otra manera:

PARA EL NAVEGADOR COMPATIBLE CON W3C:

```
event.stopPropagation();
```

PARA IE <9:

```
event.cancelBubble = true;
```

<pre><div class="parent"> <input type="text" /> </div> <div class="log"></div></pre>	HTML ⚙️
<pre>function log(str){ \$('<div/>').text(str); } \$('.parent') .focusin(function(){log('div focusin');}) .focusout(function(){log('div focusout');}) .focus(function(){log('div focus');}) .blur(function(){log('div blur');}); \$('input') .focusin(function(){log('input focusin');}) .focusout(function(){log('input focusout');}) .focus(function(){log('input focus');}) .blur(function(){log('input blur');});</pre>	JAVASCRIPT ⚙️ input focusin div focusin input focus input focusout div focusout input blur

5.3. EVENTOS GLOBALES O DEL DOM

5.3.1. Document Ready

Es una buena práctica cuando se trabaja con JavaScript esperar a que el DOM esté cargado para ejecutar nuestro código. jQuery tiene una función específica para esto:

```
$(document).ready(function(){  
    // código JS o jQuery  
});
```

ó

```
$(function(){  
    // código JS o jQuery  
});
```

5.3.2. Window load

Este evento se ejecuta cuando la página se ha cargado por completo, es decir, que se ha cargado todo incluyendo llamadas asíncronas, frames, imágenes,...

```
$(window).load(function(){  
    // código JS o jQuery  
});
```

5.3.3. Window resize

Este evento se ejecuta cuando se realiza un cambio en el tamaño del campo de visualización de la ventana del navegador.

```
$(window).resize(function(){  
    // código JS o jQuery  
});
```

5.4. EL SCROLL

5.4.1. Detectar si se está haciendo scroll en una página

Este evento se ejecuta cuando se realiza una acción de scroll en nuestra página.

```
$(window).scroll(function(){  
    console.log($(this).scrollTop());  
});
```

Este evento, en general, se aplica al window o document, aunque también se puede aplicar a cualquier selector que sea scrollable.

5.4.2. El método scrollTop()

El método scrollTop() nos permite establecer o leer el valor actual del scroll vertical de nuestra ventana. El siguiente ejemplo tiene un comportamiento no deseable pero nos sirve para entenderlo bastante bien:

```
$(window).scroll(function(){  
    console.log("La siguiente instrucción es inútil y muy molesta");  
    $(this).scrollTop(500);  
});
```

5.4.3. El método scrollLeft()

El método scrollLeft() nos permite establecer o leer el valor actual del scroll horizontal de nuestra ventana.

```
$(window).scroll(function(){  
    console.log("Posición scroll horizontal: " + $(this).scrollLeft());  
});
```

5.4.4. Animaciones suaves con el scroll

Si queremos realizar un desplazamiento suave del scroll hasta una posición concreta, aunque esto se verá más en detalle en un capítulo próximo, podemos hacer:

```
$('#arriba').click(function(){  
    // Id del elemento cliqueable  
    $('html, body').animate({scrollTop:0}, 'slow');  
    return false;  
});
```

5.5. MANEJADORES DE EVENTOS

5.5.1. Eventos relacionados con el ratón

A continuación podemos ver una lista de los eventos que se pueden definir en jQuery que tienen que ver con el ratón. Es decir, cómo definir eventos cuando el usuario realiza diferentes acciones con el ratón sobre los elementos de la página.

CLICK

Sirve para generar un evento cuando se produce un clic en un elemento de la página.

DBLCLICK

Para generar un evento cuando se produce un doble clic sobre un elemento.

HOVER

Esta función en realidad sirve para manejar dos eventos, cuando el ratón entra y sale de encima de un elemento. Por tanto espera recibir dos funciones en vez de una que se envía a la mayoría de los eventos.

MOUSEDOWN

Para generar un evento cuando el usuario hace clic, en el momento que presiona el botón e independientemente de si lo suelta o no. Sirve tanto para el botón derecho como el izquierdo del ratón.

MOUSEUP

Para generar un evento cuando el usuario ha hecho clic y luego suelta un botón del ratón. El evento

MOUSEUP

Este evento se produce sólo en el momento de soltar el botón.

MOUSEENTER

Este evento se produce al situar el ratón encima de un elemento de la página.

mouseleave

Este se desata cuando el ratón sale de encima de un elemento de la página.

MOUSEMOVE

Evento que se produce al mover el ratón sobre un elemento de la página.

MOUSEOUT

Este evento sirve para lo mismo que el evento mouseout de JavaScript. Se desata cuando el usuario sale con el ratón de la superficie de un elemento.

MOUSEOVER

Sirve para lo mismo que el evento `mouseover` de Javascript. Se produce cuando el ratón está sobre un elemento, pero tiene como particularidad que puede producirse varias veces mientras se mueve el ratón sobre el elemento, sin necesidad de haber salido.

TOGGLE

Sirve para indicar dos o más funciones para ejecutar cosas cuando el usuario realiza clics, con la particularidad que esas funciones se van alternando a medida que el usuario hace clics.

5.5.2. Eventos relacionados con el teclado

A continuación se muestran los eventos que pueden modelizarse como respuesta a la pulsación de teclas del teclado.

KEYDOWN

Este evento se produce en el momento que se presiona una tecla del teclado, independientemente de si se libera la presión o se mantiene. Se produce una única vez en el momento exacto de la presión.

KEYPRESS

Este evento ocurre cuando se digita un carácter, o se presiona otro tipo de tecla. Es como el evento `keypress` de Javascript, por lo que se entiende que `keypress()` se ejecuta una vez, como respuesta a una pulsación e inmediata liberación de la tecla, o varias veces si se pulsa una tecla y se mantiene pulsada.

KEYUP

El evento `keyup` se ejecuta en el momento de liberar una tecla, es decir, al dejar de presionar una tecla que teníamos pulsada.

5.5.3. Eventos relacionados con el foco

FOCUS

Este evento se ejecuta cuando el objeto o selector toma el foco.

BLUR

Este evento se ejecuta cuando el objeto o selector pierde el foco.

FOCUSIN

Este evento se ejecuta cuando el objeto o selector toma el foco. A diferencia del evento `focus`, el evento `focusin` cumple y realiza el principio de propagación de JavaScript.

FOCUSOUT

Este evento se ejecuta cuando el objeto o selector pierde el foco. A diferencia del evento `blur`, el evento `focusout` cumple y realiza el principio de propagación de JavaScript.

CLICK.HIDEMENU

jQuery nos permite espacio de nombres de controladores de eventos a través de este evento. Es como si fuese otro tipo de evento que únicamente se puede activar o desactivar de forma directa en el objeto que nosotros deseemos. Si asociamos este evento al BODY de nuestra página, se convertirá en un evento con un comportamiento global, es decir, que cualquier click que se realice dentro de nuestro espacio de nombres pasará por aquí. El ejemplo más típico es:

```
$('#body').on('click.hideMenu', function(e) {  
    // Check to see if the list is currently displayed.  
    if ($(e.target).parents('#main-menu').hasClass('menu-showed')) {  
        $('#main-menu').removeClass('menu-showed');  
    }  
});
```

5.6. EJEMPLOS

```
$('#a.button').click(function(event){
    console.log('Manejador para el evento click del enlace(a) con clase(.) button');
    event.preventDefault();
});

$('body').on('click.hideMenu', function(e) {
    if ($(e.target).prop("tagName") == "A") { ... }
});

if ( "onhelp" in window ) { window.onhelp = function () { return false; }; }

$(window).keydown(function(e){
    var keyCode    = e.keyCode || e.which;
    var keyPressed = String.fromCharCode(e.keyCode);
    var target      = $(e.target);
    var targetTag   = target.prop("tagName");

    // If the cursor is inside an input or textarea, we enable default behavior.
    // Otherwise, you override default behavior.
    if((keyPressed == 'C' && e.ctrlKey && !e.shiftKey) ||
        (keyPressed == 'V' || keyPressed == 'X') &&
        e.ctrlKey && !e.shiftKey && (target.is('input') || target.is('textarea'))){
        return true;
    }

    shiftPressed = false; if(e.shiftKey) shiftPressed = true;
    ctrlPressed  = false; if(e.ctrlKey) ctrlPressed  = true;
    lastKeyPressed = keyCode;
});

$('select, input, button').on('blur change click', function(e) {
    // Código jQuery
});
```

6. Funciones nominales y anónimas

6.1. FUNCIONES NOMINALES

Para ejecutar un bloque de instrucciones y realizar una tarea concreta, JavaScript utiliza las “funciones”, para agruparlas. Para definir cada función se le debe asignar un nombre para que, más tarde, pueda ser invocada en el momento que se desee.

Si la función necesita algún dato como parámetro, se indicará en la definición de la función y si, además, necesita devolver algo se indicará con la palabra clave `return`.

6.2. FUNCIONES ANONIMAS

Si una función nominal tiene un nombre, parámetros, ... la diferencia con las anónimas es que estas no poseen nombre.

```
function(){  
    //Grupo de instrucciones que realizará la función  
}
```

Para utilizar esta función, la clave está en utilizarla como si tuviese nombre, es decir, encerrándola entre paréntesis y colocando otro par de paréntesis al final. De esta manera, se definirá y autoejecutará a la vez.

```
(function(){ }());
```

Las ventajas para usar este tipo de funciones son que se puede pasar como parámetro a otra función y que, al ser anónima, todo lo que se defina o ejecute dentro de ella no interferirá con nuestra aplicación. Esto es lo que se denomina **ENCAPSULAMIENTO**.

El ejemplo más claro y conocido de función anónima es:

```
$(document).ready(function(){  
    //Aquí va todo el código  
});  
ó  
$(function(){  
    //Aquí va todo el código  
});
```

7. Métodos

7.1. EL DREAM-TEAM DE JQUERY

En jQuery, como en un partido de fútbol hay un conjunto de métodos que son el “dream-team” por su utilización. Si pensamos un poco en los métodos que más se utilizan en los proyectos seguro que llegaremos a la conclusión de que la mayoría de las necesidades de estos proyectos se podrían resolver con la siguiente lista:

Método	Descripción
append()	Inserta el contenido especificado por parámetro al final del elemento indicado por el selector.
after()	Inserta el contenido especificado por parámetro al después del elemento indicado por el selector.
next()	Obtiene el elemento inmediatamente posterior (el elemento siguiente) de cada elemento dentro del conjunto de los elementos coincidentes.
prev()	Obtiene el elemento inmediatamente anterior de cada elemento dentro del conjunto de los elementos coincidentes.
attr()	Obtiene el valor del atributo indicado en el primer parámetro del selector seleccionado. Si tiene establecido un segundo parámetro el método funciona como “setter” estableciendo el atributo pasado como primer parámetro al valor propuesto en el segundo parámetro.
css()	Obtiene o establece estilos al selector indicado. Se puede hacer a través de nombre – valor o a través de un JSON.
html()	Obtiene o establece el contenido HTML del selector indicado.
val()	Obtiene o establece el atributo value del selector indicado. Sólo funciona con elementos de tipo formulario (input, select, button, ...) y que tienen este atributo ya definido.
position()	Obtiene las coordenadas actuales del primer elemento coincidente con el selector indicado tomando como punto de referencia el último “padre” o “ancestro” que tenga su propiedad de “position: relative”.
offset()	Obtiene las coordenadas actuales del primer elemento coincidente con el selector indicado tomando como punto de referencia la posición del documento o etiqueta BODY.
parent()	Obtiene el elemento “padre” del selector indicado.

7.2. MÉTODOS “SUPLENTE” DEL DREAM-TEAM

La siguiente lista de métodos es la que complementa el resto de necesidades que se suelen dar en los diferentes proyectos.

Método	Descripción
parents()	Obtiene el elemento “ancestro” del selector indicado que contenga el selector pasado como parámetro.
prepend()	Inserta el contenido especificado por parámetro al principio del elemento indicado por el selector.
before()	Inserta el contenido especificado por parámetro al antes del elemento indicado por el selector.
find()	Obtiene el elemento “descendiente” que coincida con el selector propuesto como parámetro.
prop()	Obtiene el valor del atributo indicado en el primer parámetro del selector seleccionado. Si tiene establecido un segundo parámetro el método funciona como “setter” estableciendo el atributo pasado como primer parámetro al valor propuesto en el segundo parámetro. Esta se usa, sobretodo, con propiedades de carácter booleano pero en su mayoría puede servir igual que attr().
data()	Obtiene o establece el atributo “data-{id}” de HTML5 del selector indicado.
Efectos de animación	
show()	Muestra el elemento y su contenido del selector indicado. Este método nos permite especificar la duración de la animación.
hide()	Oculto el elemento y su contenido del selector indicado. Este método nos permite especificar la duración de la animación.
slideDown()	Muestra el elemento y su contenido del selector indicado con efecto de animación tipo “slide”. Este método nos permite especificar la duración de la animación.
slideUp()	Oculto el elemento y su contenido del selector indicado con efecto de animación tipo “slide”. Este método nos permite especificar la duración de la animación.
fadeIn()	Muestra el elemento y su contenido del selector indicado con efecto de animación tipo “desvanecimiento” o “fade”. Este método nos permite especificar la duración de la animación.
fadeOut()	Oculto el elemento y su contenido del selector indicado con efecto de animación tipo “desvanecimiento” o “fade”. Este método nos permite especificar la duración de la animación.
animation()	Realiza una animación personalizada a través de un conjunto de propiedades CSS que se le pasan por parámetro. Los parámetros CSS se establecen a través de JSON. Este método nos permite especificar la duración de la animación.

7.3. MANIPULACION DE ELEMENTOS

Una vez realizada la selección de los elementos que desea utilizar, *la diversión comienza*. Es posible cambiar, mover, eliminar y duplicar elementos. También crear nuevos a través de una sintaxis simple.

La documentación completa sobre los métodos de manipulación puede encontrarla en [la sección Manipulation](#).

7.3.1. Obtener y establecer información en elementos

Existen muchas formas por las cuales se puede modificar un elemento. Entre las tareas más comunes están las de cambiar el HTML interno o algún atributo del mismo. Para este tipo de tareas, jQuery ofrece métodos simples, funcionales en todos los navegadores modernos. Incluso es posible obtener información sobre los elementos utilizando los mismos métodos pero en su forma de método obtenedor.

Realizar cambios en los elementos, es un trabajo trivial, pero hay que recordar que el cambio afectará a todos los elementos en la selección, por lo que, si desea modificar un sólo elemento, tiene que estar seguro de especificarlo en la selección antes de llamar al método establecedor.

Cuando los métodos actúan como obtenedores, por lo general, solamente trabajan con el primer elemento de la selección. Además no devuelven un objeto jQuery, por lo cual no es posible encadenar más métodos en el mismo. Una excepción es el método `$.fn.text`, el cual permite obtener el texto de los elementos de la selección.

Método	Descripción
<code>\$.fn.html</code>	Obtiene o establece el contenido HTML de un elemento.
<code>\$.fn.text</code>	Obtiene o establece el contenido en texto del elemento; en caso se pasarle como argumento código HTML, este es despojado.
<code>\$.fn.attr</code>	Obtiene o establece el valor de un determinado atributo.
<code>\$.fn.width</code>	Obtiene o establece el ancho en pixeles del primer elemento de la selección como un entero.
<code>\$.fn.height</code>	Obtiene o establece el alto en pixeles del primer elemento de la selección como un entero.
<code>\$.fn.position</code>	Obtiene un objeto con información sobre la posición del primer elemento de la selección, relativo al primer elemento padre posicionado. Este método es solo obtenedor.
<code>\$.fn.val</code>	Obtiene o establece el valor (<i>value</i>) en elementos de formularios.

Cambiar el HTML de un elemento

```
$('#myDiv p:first').html('Nuevo <strong>primer</strong> párrafo');
```

7.3.2. Mover, copiar y eliminar elementos

Existen varias maneras para mover elementos a través del DOM; las cuales se pueden separar en dos enfoques:

- Querer colocar el/los elementos seleccionados de forma relativa a otro elemento
- Querer colocar un elemento relativo a uno o varios elementos seleccionados.

Por ejemplo, jQuery provee los métodos `$.fn.insertAfter` y `$.fn.after`. El método `$.fn.insertAfter` coloca al elemento seleccionado después del elemento que se haya pasado como argumento; mientras que el método `$.fn.after` coloca al elemento pasado como argumento después del elemento seleccionado. Otros métodos también siguen este patrón: `$.fn.insertBefore` y `$.fn.before`; `$.fn.appendTo` y `$.fn.append`; y `$.fn.prependTo` y `$.fn.prepend`.

La utilización de uno u otro método dependerá de los elementos que tenga seleccionados y el tipo de referencia que se quiera guardar con respecto al elemento que se esta moviendo.

Mover elementos utilizando diferentes enfoques

```
// hacer que el primer item de la lista sea el último
var $li = $('#myList li:first').appendTo('#myList');

// otro enfoque para el mismo problema
$('#myList').append($('#myList li:first'));

// debe tener en cuenta que no hay forma de acceder a la
// lista de items que se ha movido, ya que devuelve la lista en sí
```

Clonar elementos

Cuando se utiliza un método como `$.fn.appendTo`, lo que se está haciendo es mover al elemento; pero a veces en lugar de eso, se necesita mover un duplicado del mismo elemento. En este caso, es posible utilizar el método `$.fn.clone`.

Obtener una copia del elemento

```
// copiar el primer elemento de la lista y moverlo al final de la misma
$('#myList li:first').clone().appendTo('#myList');
```

Si se necesita copiar información y eventos relacionados al elemento, se debe pasar `true` como argumento de `$.fn.clone`.

Eliminar elementos

Existen dos formas de eliminar elementos de una página: Utilizando `$.fn.remove` o `$.fn.detach`. Cuando desee eliminar de forma permanente al elemento, utilice el método `$.fn.remove`. Por otro lado, el método `$.fn.detach` también elimina el elemento, pero mantiene la información y eventos asociados al mismo, siendo útil en el caso que necesite reinsertar el elemento en el documento.

El método `$.fn.detach` es muy útil cuando se esta manipulando de forma severa un elemento, ya que es posible eliminar al elemento, trabajarlo en el código y luego restaurarlo en la página nuevamente. Esta forma tiene como beneficio no tocar el DOM mientras se está modificando la información y eventos del elemento.

Por otro lado, si se desea mantener al elemento pero se necesita eliminar su contenido, es posible utiliza el método `$.fn.empty`, el cual "vaciará" el contenido HTML del elemento.

7.3.3. Crear nuevos elementos

jQuery provee una forma fácil y elegante para crear nuevos elementos a través del mismo método `$()` que se utiliza para realizar selecciones.

Crear nuevos elementos

```
$('#<p>Un nuevo párrafo</p>');  
$('#<li class="new">nuevo ítem de la lista</li>');
```

Crear un nuevo elemento con atributos utilizando un objeto

```
$('#<a/>', {  
  html : 'Un <strong>nuevo</strong> enlace',  
  'class' : 'new',  
  href : 'foo.html'  
});
```

Observa que en el objeto que se pasa como argumento, la propiedad `class` está entre comillas, mientras que la propiedad `href` y `html` no lo están. Por lo general, los nombres de propiedades no deben estar entre comillas, excepto en el caso que se utilice como nombre una palabra reservada (como es el caso de `class`).

Cuando se crea un elemento, éste no es añadido inmediatamente a la página, sino que se debe hacerlo en conjunto con un método.

Crear un nuevo elemento en la página

```
var $myNewElement = $('<p>Nuevo elemento</p>');
$myNewElement.appendTo('#content');

// eliminará al elemento <p> existente en #content
$myNewElement.insertAfter('ul:last');

// clonar al elemento <p> para tener las dos versiones
$('ul').last().after($myNewElement.clone());
```

Estrictamente hablando, no es necesario guardar al elemento creado en una variable — es posible llamar al método para añadir el elemento directamente después de `$()`. Sin embargo, la mayoría de las veces se deseará hacer referencia al elemento añadido, por lo cual, si se guarda en una variable no es necesario seleccionarlo después.

Crear y añadir al mismo tiempo un elemento a la página

```
$('ul').append('<li>item de la lista</li>');
```

Nota La sintaxis para añadir nuevos elementos a la página es muy fácil de utilizar, pero es tentador olvidar que hay un costo enorme de rendimiento al agregar elementos al DOM de forma repetida. Si esta añadiendo muchos elementos al mismo contenedor, en lugar de añadir cada elemento uno por vez, lo mejor es concatenar todo el HTML en una única cadena de caracteres para luego anexarla al contenedor. Una posible solución es utilizar un array que posea todos los elementos, luego reunirlos utilizando `join` y finalmente anexarla.

```
var myItems = [], $myList = $('#myList');

for (var i=0; i<100; i++) {
    myItems.push('<li>item ' + i + '</li>');
}

$myList.append(myItems.join(''));
```

7.3.4. Manipulación de atributos

Las capacidades para la manipulación de atributos que ofrece la biblioteca son extensos. La realización de cambios básicos son simples, sin embargo el método `$.fn.attr` permite manipulaciones más complejas.

Manipular un simple atributo

```
$('#myDiv a:first').attr('href', 'newDestination.html');
```

Manipular múltiples atributos

```
$('#myDiv a:first').attr({  
    href : 'newDestination.html',  
    rel : 'super-special'  
});
```

Utilizar una función para determinar el valor del nuevo atributo

```
$('#myDiv a:first').attr({  
    rel : 'super-special',  
    href : function(idx, href) {  
        return '/new/' + href;  
    }  
});  
  
$('#myDiv a:first').attr('href', function(idx, href) {  
    return '/new/' + href;  
});
```

7.4. EL MÉTODO DATA

A menudo nos encontraremos con que existe información acerca de un elemento que se necesita guardar. En JavaScript es posible hacerlo añadiendo propiedades al DOM del elemento, pero esta práctica conlleva enfrentarse a consumos excesivos de memoria (*memory leaks*) en algunos navegadores. jQuery ofrece una manera sencilla para poder guardar información relacionada a un elemento, y la misma biblioteca se ocupa de manejar los problemas que pueden surgir por falta de memoria.

Guardar y recuperar información relacionada a un elemento

```
$('#myDiv').data('keyName', { foo : 'bar' });  
$('#myDiv').data('keyName'); // { foo : 'bar' }
```

A través del método `$.fn.data` es posible guardar cualquier tipo de información sobre un elemento, y es difícil exagerar la importancia de este concepto cuando se está desarrollando una aplicación compleja.

Por ejemplo, si desea establecer una relación entre el ítem de una lista y el div que hay dentro de este ítem, es posible hacerlo cada vez que se interactúa con el ítem, pero una mejor solución es hacerlo una sola vez, guardando un puntero al div utilizando el método `$.fn.data`:

Establecer una relación entre elementos utilizando el método `$.fn.data`

```
$('#myList li').each(function() {  
    var $li = $(this), $div = $li.find('div.content');  
    $li.data('contentDiv', $div);  
});  
  
// luego, no se debe volver a buscar al div;  
// es posible leerlo desde la información asociada al ítem de la lista  
var $firstLi = $('#myList li:first');  
$firstLi.data('contentDiv').html('nuevo contenido');
```

Además es posible pasarle al método un objeto conteniendo uno o más pares de conjuntos palabra clave-valor.

7.6. EL METODO ATTR

Los atributos de los elementos HTML que conforman una aplicación pueden contener información útil, por eso es importante poder establecer y obtener esa información.

El método `$.fn.attr` actúa tanto como método de escritura como de lectura. Además, al igual que el método `$.fn.css`, cuando se utiliza como método de escritura, puede aceptar un conjunto de palabras clave-valor o un objeto conteniendo más elementos.

Establecer atributos

```
$('#a').attr('href', 'allMyHrefsAreTheSameNow.html');  
$('#a').attr({  
  'title' : 'all titles are the same too',  
  'href' : 'somethingNew.html'  
});
```

En el ejemplo, el objeto pasado como argumento está escrito en varias líneas. Como se explicó anteriormente, los espacios en blanco no importan en JavaScript, por lo cual, es libre de utilizarlos para hacer el código más legible. En entornos de producción, se pueden utilizar herramientas de minificación, las cuales quitan los espacios en blanco (entre otras cosas) y comprimen el archivo final.

Obtener atributos

```
$('#a').attr('href'); // devuelve el atributo href perteneciente  
                     // al primer elemento <a> del documento
```

7.7. OTROS METODOS UTILES

jQuery ofrece varios métodos utilitarios dentro del espacio de nombres \$. Estos métodos son de gran ayuda para llevar a cabo tareas rutinarias de programación. A continuación se muestran algunos ejemplos, para una completa documentación sobre ellos, visite api.jquery.com/category/utilities/.

7.7.1. \$.fn.CSS

El método css permite cambiar el estilo del selector a través de un string o a través de JSON

```
$('#div').css("background-color", "red");  
$('#div').css({backgroundColor: "red"});
```

7.7.2. \$.trim

Remueve los espacios en blanco del principio y final.

```
$.trim('   varios espacios en blanco   ');  
// devuelve 'varios espacios en blanco'
```

7.7.3. \$.each

Interactúa en arrays y objetos.

```
$.each([ 'foo', 'bar', 'baz' ], function(idx, val) {  
    console.log('elemento ' + idx + ' es ' + val);  
});  
  
$.each({ foo : 'bar', baz : 'bim' }, function(k, v) {  
    console.log(k + ' : ' + v);  
});
```

7.7.4. \$.inArray

Devuelve el índice de un valor en un array, o -1 si el valor no se encuentra en el array.

```
var myArray = [ 1, 2, 3, 5 ];  
if ($.inArray(4, myArray) !== -1) {  
    console.log('valor encontrado');  
}
```

7.7.5. \$.extend

Cambia las propiedades del primer objeto utilizando las propiedades de los subsecuentes objetos.

```
var firstObject = { foo : 'bar', a : 'b' };  
var secondObject = { foo : 'baz' };
```

```
var newObject = $.extend(firstObject, secondObject);  
console.log(firstObject.foo); // 'baz'  
console.log(newObject.foo);   // 'baz'
```

Si no se desea cambiar las propiedades de ninguno de los objetos que se utilizan en `$.extend`, se debe incluir un objeto vacío como primer argumento.

```
var firstObject = { foo : 'bar', a : 'b' };  
var secondObject = { foo : 'baz' };  
  
var newObject = $.extend({}, firstObject, secondObject);  
console.log(firstObject.foo); // 'bar'  
console.log(newObject.foo);   // 'baz'
```

7.7.6. \$.isFunction

Devuelve true si es una declaración de función.

```
var myValue = [1, 2, 3];  
jQuery.isFunction(myValue); // falso (false)
```

7.7.7. \$.isPlainObject

Devuelve true si es un objeto.

```
var myValue = [1, 2, 3];  
jQuery.isPlainObject(myValue); // falso (false)
```

7.7.8. \$.isArray

Devuelve true si el tipo de datos es un array.

```
var myValue = [1, 2, 3];  
jQuery.isArray(myValue); // verdadero (true)
```

7.7.9. \$.proxy

Devuelve una función que siempre se ejecutará en el alcance (*scope*) provisto — en otras palabras, establece el significado de `this` (incluido dentro de la función) como el segundo argumento.

```
var myFunction = function() { console.log(this); };  
var myObject = { foo : 'bar' };  
  
myFunction(); // devuelve el objeto window  
  
var myProxyFunction = $.proxy(myFunction, myObject);
```

```
myProxyFunction(); // devuelve el objeto myObject
```

Si se posee un objeto con métodos, es posible pasar dicho objeto y el nombre de un método para devolver una función que siempre se ejecuta en el alcance de dicho objeto.

```
var myObject = {  
  myFn : function() {  
    console.log(this);  
  }  
};
```

```
$('#foo').click(myObject.myFn); // registra el elemento DOM #foo  
$('#foo').click($.proxy(myObject, 'myFn')); // registra myObject
```

8. Efectos

8.1. EFECTOS DE TRANSPARENCIA (SHOW Y HIDE)

jQuery puede mostrar u ocultar el contenido instantáneamente con los métodos `show()` y `hide()` .

```
// Ocultar instantáneamente todos los elementos p (párrafos)
$("p").hide();

// Mostrar instantáneamente todos los divs que tienen la clase .hidden
$("div.hidden").show();
```

Cuando jQuery oculta un elemento, que establece su CSS `display` propiedad a `none` . Esto significa que el contenido tendrá las propiedades `width` y `height` establecidas a cero.

jQuery también puede mostrar y ocultar contenidos a través de animaciones mediante los parámetros `slow` o `fast`:

```
// Ocultar poco a poco todos los elementos p (párrafos)
$("p").hide("slow");

// Mostrar rápidamente todos los divs que tienen la clase .hidden
$("div.hidden").show("fast");
```

Aunque, si uno lo prefiere, puede ejecutarse el proceso de mostrado u ocultado estableciendo tiempos específicos (en milisegundos):

```
// Ocultar todos los elementos p (párrafos) en 0.25 segundos
$("p").hide(250);

// Mostrar todos los divs que tienen la clase .hidden en 1 segundo
$("div.hidden").show(1000);
```

8.2. EFECTOS DE ATENUACIÓN (FADEIN Y FADEOUT)

jQuery no sólo provee de los efectos `show()` y `hide()` que muestran u ocultan los elementos de nuestra páginas. También nos provee de otro tipo de efectos de animación como son los efectos de atenuación o de deslizamiento.

Si se desea hacer que un contenido o elemento se desvanezca o aparezca podríamos cambiar las palabras clave `show()` y `hide()` por `fadeIn()` y `fadeOut()`:

```
// Desvanecer todos los elementos p (párrafos) en 0.25 segundos
$("p").fadeOut(250);

// Desatenuar todos los divs que tienen la clase .hidden en 1 segundo
$("div.hidden").fadeIn(1000);
```

Los métodos `fadeIn()` y `fadeOut()` admiten los mismos parámetros que los efectos anteriores.

8.3. EFECTOS DE DESLIZAMIENTO (SLIDEIN Y SLIDEOUT)

Como se mencionaba en el apartado anterior, jQuery nos provee de varios tipos de efectos de animación como son los efectos de atenuación o de deslizamiento.

Si se desea hacer que un contenido o elemento se desvanezca o aparezca podríamos cambiar las palabras clave `show()` y `hide()` por `fadeIn()` y `fadeOut()`:

```
// Deslizar hacia abajo (aparecer) todos los elementos p (párrafos) en 0.25 segundos
$("p").slideDown(250);

// Deslizar hacia arriba (desaparecer) todos los divs que tienen la clase .hidden en 1 segundo
$("div.hidden").slideUp(1000);
```

Los métodos `slideDown()` y `slideUp()` admiten los mismos parámetros que los efectos anteriores.

8.4. CAMBIAR DE EFECTO BASADO EN ESTADOS (TOGGLE)

jQuery también le permite cambiar la visibilidad de un contenido en función de su estado de visibilidad actual. El método `toggle()` mostrará el contenido que se encuentra oculto en ese momento y ocultará el contenido que esté visible en ese momento. El método `toggle()` admite los mismos parámetros que los efectos anteriores.

Si se desea hacer que un contenido o elemento se desvanezca o aparezca podríamos cambiar las palabras clave `show()` y `hide()` por `fadeIn()` y `fadeOut()`:

```
// Cambiar instantáneamente el estado de visualización de todos los elementos p (párrafos)
$("p").toggle();

// Cambiar lentamente el estado de visualización de todos los elementos img (imágenes)
$("img").toggle("slow");

// Cambiar en 1,8 segundos el estado de visualización de todos los elementos div (capas)
$("div").toggle(1800);
```

8.5. AÑADIENDO FUNCIONALIDADES DESPUES DEL EFECTO

Es importante darse cuenta de que los efectos anteriormente comentados sólo inician la animación. Una vez iniciada, la animación es implementada a través de propiedades CSS encapsuladas dentro de un bucle `setInterval` de JavaScript y ejecutado de forma asíncrona.

Por esta razón, si necesitamos realizar acciones o ejecutar funcionalidades después de ejecutar un efecto de animación, es necesario utilizar una función de devolución de llamada de animación. Esta función de devolución se puede especificar como segundo argumento en cualquiera de los métodos de animación discutidos anteriormente. Veamos un ejemplo:

```
// Añadir clase oculto una vez terminado el efecto de desvanecimiento de salida
$("#div").fadeOut(250, function(){
    $(this).addClass("oculto")
});

// Añadir clase visible una vez terminado el efecto de desvanecimiento de entrada
$("#div").fadeIn(250, function(){
    $(this).addClass("visible")
});
```

8.6. OTROS METODOS PARA GESTIONAR ANIMACIONES

El método `stop()` permite parar las animaciones en curso y así evitar anidamientos.

```
// Antes de iniciar el cambio, parar todas las animaciones en imágenes
$("#img").stop().toggle("slow");
```

El método `delay()` se utiliza para introducir un retardo entre animaciones sucesivas.

```
// Antes de iniciar el cambio, esperar 0,3 segundos
$("#img").delay(300).toggle("slow");
```

8.7. EFECTOS PERSONALIZADOS CON ANIMATE

Gracias a jQuery es posible animar las propiedades CSS a nuestra elección a través del método `animate()`. El método `animate()` permite animar un valor determinado, o un valor en relación con el valor actual.

```
// Efecto personalizado con .animate()
$("#div.funtimes").animate({left: "+=50", opacity: 0.25}, 300, function(){ console.log("hecho!"); });
```

El primer parámetro es un JSON que contiene las propiedades CSS a animar. El segundo parámetro es la duración de la animación y el tercer parámetro es el conjunto de acciones que se llevarán a cabo una vez termine la animación.

8.7.1. Flexibilización

La flexibilización describe la manera en que se produce un efecto (si la velocidad de cambio es constante o varía a lo largo de la duración de la animación). Para definir el comportamiento lineal o variante jQuery incluye dos propiedades que son `swing` y `linear`. Para realizar las animaciones más “naturales” existen plugins ya creados y disponibles para su descarga.

```
// Efecto personalizado con .animate()  
$("div.funtimes").animate({left: ["+=50", "swing"], opacity: [0.25, "linear"]}, 300);
```

9. Recorrer el DOM

Una vez obtenida la selección, es posible encontrar otros elementos utilizando a la misma selección.

Moverse a través del DOM utilizando métodos de recorrido

```
// seleccionar el inmediato y próximo elemento <p> con respecto a H1
$('h1').next('p');

// seleccionar el elemento contenedor al primer div visible
$('div:visible').parent();

// seleccionar el elemento <form> más cercano al primer input de tipo texto
$('input[type="text"]').closest('form');

// seleccionar todos los elementos hijos de #myList
$('#myList').children();

// seleccionar todos los items hermanos del elemento li con clase "selected"
$('li.selected').siblings();

// Recibir los elementos del DOM que coincidan con el selector li.selected
$('li.selected').get();

// Conseguir el índice asociado a li.selected
$('li.selected').index();
```

También es posible interactuar con la selección utilizando el método `$.fn.each`. Dicho método interactúa con todos los elementos obtenidos en la selección y ejecuta una función por cada uno. La función recibe como argumento el índice del elemento actual y al mismo elemento. De forma predeterminada, dentro de la función, se puede hacer referencia al elemento DOM a través de la declaración `this`.

Interactuar en una selección

```
$('#myList li').each(function(idx, el) {
    console.log('El elemento ' + idx + ' contiene el siguiente HTML: ' + $(el).html());
});
```

10. Observadores de Mutación

Muchas aplicaciones y sitios crean sus propios controles personalizados con HTML y JavaScript. Estos controles no son intrínsecos como los que proporciona el elemento input. Para que estos controles funcionen bien con el resto de la plataforma web, deben ser capaces de adaptarse a los cambios de contenido, responder a los eventos y administrar la interacción del usuario. Las aplicaciones y los sitios necesitan supervisar las mutaciones en el modelo de objetos de documento (DOM) y responder de manera adecuada.

En Internet Explorer 11, los desarrolladores web obtienen compatibilidad para los observadores de mutación DOM del W3C DOM4 que les ayuda a controlar qué mutaciones tienen que observar y seguir. Por ejemplo, quizás se quiera recibir notificaciones después de que se ejecute un script o antes de que se dibuje un nuevo marco en la pantalla.

Los observadores de mutación permiten realizar sustituciones con un rápido rendimiento en los mismos escenarios que los eventos de mutación, y suponen una alternativa a los escenarios admitidos por los eventos de cambio de propiedad.

10.1. EL OBJETO MUTATIONOBSERVER

Un objeto **MutationObserver** sirve para supervisar y registrar los cambios especificados en uno o varios nodos del DOM. **MutationObserver** trabaja de forma asíncrona, lo que significa que dejará los demás procesos trabajar. En vez de realizar una llamada a un evento por cada cambio ejecuta una llamada con una colección de los cambios realizados a través de un "callback".

10.1.1. Compatibilidad en navegadores

Solo tiene soporte en los siguientes navegadores:

- Chrome 26+ (Chrome 18-25 con el prefijo WebKit)
- Firefox 14+
- Internet Explorer 11+
- Opera 15+
- Safari 6.0 con el prefijo WebKit

De todas formas, si se desea detectar si el navegador acepta o no el objeto mutationobserver se puede consultar escribiendo:

```
var ObserveMutationDisponible = 'MutationObserver' in window;
```

10.1.2. Tipos de cambios del objeto mutationobserver

Entre los tipos de cambio que se pueden observar nos encontramos:

- Gestión de elementos secundarios (Add, Remove).
- Gestión de atributos (Add, Remove, Change).
- Cambios en valores de nodo de texto secundario.

Opción	Uso / Utilización
childList	Observar cambios en los hijos.
attributes	Observar los cambios en los atributos.
characterData	Observar cambios en los datos.
subtree	Observar todos los descendientes, no sólo los hijos directos.
attributeOldValue	Conocer el valor anterior del atributo antes de ser cambiado. <i>Para utilizar esta opción 'attributes' debe estar activada.</i>
characterDataOldValue	Conocer los datos antes de que fueran cambiados, <i>Para utilizar esta opción 'characterData' debe estar activada.</i>
attributeFilter	Indica el listado de atributos a observar. Por defecto está vacío y significa que se observan todos los cambios de todos los atributos.

Todas las opciones son de carácter booleano, es decir, que sólo puede adquirir el valor `true` o `false`. Si se desea utilizar `mutationobserver` se debe activar, al menos, `childList`, `attributes`, o `characterData`.

10.1.3. Cómo usar el objeto `mutationobserver`

Lo primero es crear el constructor:

```
// Create an observer object and assign a callback function
var observerObject = new MutationObserver(mutationObjectCallback);
```

Cuando se haya creado el objeto **MutationObserver**, se puede utilizar el método **observe** para registrar los nodos que se quiere supervisar y especificar los tipos de información que se desean recuperar (como se indicó en el apartado 7.3).

10.1.4. Detener la observación

Para detener la observación del elemento se debe utilizar:

```
observer.disconnect();
```

10.2. EL OBJETO MUTATIONRECORD

MutationRecord es un objeto que se envía como parametro al callback del observador. Se caracteriza por tener las siguientes propiedades:

Propiedad	Descripción
type	Devuelve el tipo mutación, attributes, characterData o childList.
target	Devuelve el nodo afectado por la mutación.
addedNodes	Devuelve una lista de nodos agregados o null si ninguno fue agregado.
removedNodes	Devuelve una lista de nodos eliminados o null si ninguno fue eliminado.
previousSibling	Devuelven el pariente anterior del nodo eliminado o agregado o null si no existe.
nextSibling	Devuelve el pariente siguiente del nodo eliminado o agregado o null si no existe.
attributeName	Devuelve el nombre del atributo cambiado o null si no existe cambio.
attributeNamespace	Devuelve el namespace del atributo cambiado o null si no existe.
oldValue	Devuelve el valor anterior a su modificación, ya sea el atributo, los datos o los nodos.

Estamos observando una etiqueta de parrafo, ahora vamos a saber que cambio en la etiqueta.

10.3. EJEMPLO BASICO

```
var observer, callback;
callback = function( listaDeCambios ) {
    listaDeCambios.forEach(function(mutation){
        console.log( 'El ' + mutation.type + ' fue afectado en el nodo ' + mutation.target );
        console.log( 'Antes su valor era: ' + mutation.oldValue)
    });
}
// creamos el objeto que va a observar todos los cambios.
observer = new MutationObserver(callback);

var options1 = {'attributes': true};
var options2 = {'attributes': true, 'childList': true, 'subtree': true, 'attributeOldValue': true,
'attributeFilter': ['class']},

var element1 = document.querySelector('p');
var element2 = document.querySelector('#container');

observer.observe(element1, options1);
observer.observe(element2, options2);
```

11. Peticiones Ajax

jQuery posee varios métodos para trabajar con Ajax. Sin embargo, todos están basados en el método `$.ajax`, por lo tanto, si se consigue entender este método, se entenderán todos los demás. A continuación se explica el método `$.ajax` y luego se indicará un breve resumen sobre los demás métodos.

Generalmente, es preferible utilizar el método `$.ajax` en lugar de los otros, ya que ofrece más características y su configuración es muy comprensible.

11.1. \$.AJAX

El método `$.ajax` es configurado a través de un objeto, el cual contiene todas las instrucciones que necesita jQuery para completar la petición. Este método es particularmente útil debido a que ofrece la posibilidad de especificar acciones en caso que la petición haya fallado o no. Además, al estar configurado a través de un objeto, es posible definir sus propiedades de forma separada, haciendo que sea más fácil la reutilización del código. Se puede visitar api.jquery.com/jquery.ajax para consultar la documentación sobre las opciones disponibles en el método.

11.1.1. Utilizar el método `$.ajax`

```
$.ajax({  
    // la URL para la petición  
    url : 'post.php',  
    // la información a enviar (también es posible utilizar una cadena de datos)  
    data : { id : 123 },  
    // especifica si será una petición POST o GET  
    type : 'GET',  
    // el tipo de información que se espera de respuesta  
    dataType : 'json',  
    success : function(json) {  
        // la respuesta es pasada como argumento a la función  
        $('<h1/>').text(json.title).appendTo('body');  
        $('<div class="content"/>').html(json.html).appendTo('body');  
    }, error : function(xhr, status) {  
        // el objeto de la petición en crudo y código de status de la petición  
        alert('Disculpe, existió un problema');  
    }, complete : function(xhr, status) {  
        // tanto si falla como si no  
        alert('Petición realizada');  
    }  
});
```

En el ejemplo se ha utilizado el parámetro `dataType`. Si el servidor devuelve información que es diferente al formato especificado por el parámetro `dataType`, el código fallará, y la razón de porque lo hace no siempre quedará clara debido a que la respuesta HTTP no mostrará ningún tipo de error. Cuando se esté trabajando con peticiones Ajax, se debe estar seguro que el servidor esta enviando el tipo de información que se esta requiriendo y verificar que la cabecera `Content-type` es exacta al tipo de dato. Por ejemplo, para información en formato JSON, la cabecera `Content-type` debería ser `application/json`.

11.1.2. Opciones del método \$.ajax

El método `$.ajax` posee muchas opciones de configuración, y es justamente esta característica la que hace que este método sea realmente potente. Para ver la lista completa de las opciones disponibles, se puede consultar api.jquery.com/jquery.ajax. Las más comunes son:

Opción	Descripción
<code>async</code>	Establece si la petición será asíncrona o no. De forma predeterminada el valor es <code>true</code> . Se debe tener en cuenta que si la opción se establece en <code>false</code> , la petición bloqueará la ejecución de otros códigos hasta que dicha petición haya finalizado.
<code>cache</code>	Establece si la petición será guardada en la cache del navegador. De forma predeterminada es <code>true</code> para todos los <code>dataType</code> excepto para <code>script</code> y <code>jsonp</code> . Cuando posee el valor <code>false</code> , se agrega una cadena de caracteres anti-cache al final de la URL de la petición.
<code>complete</code>	Establece una función de devolución de llamada que se ejecuta cuando la petición esta completa, aunque haya fallado o no. La función recibe como argumentos el objeto de la petición en crudo y el código de estatus de la misma petición.
<code>context</code>	Establece el alcance en que la/las funciones de devolución de llamada se ejecutaran (por ejemplo, define el significado de <code>this</code> dentro de las funciones). De manera predeterminada <code>this</code> hace referencia al objeto originalmente pasado al método <code>\$.ajax</code> .
<code>data</code>	Establece la información que se enviará al servidor. Esta puede ser tanto un objeto JSON como una cadena de datos (por ejemplo <code>foo=bar&baz=bim</code>)
<code>dataType</code>	Establece el tipo de información que se espera recibir como respuesta del servidor. Si no se especifica ningún valor, de forma predeterminada, jQuery revisa el tipo de MIME que posee la respuesta.
<code>error</code>	Establece una función de devolución de llamada a ejecutar si resulta algún error en la petición. Dicha función recibe como argumentos el objeto de la petición en crudo y el código de estatus de la misma petición.
<code>jsonp</code>	Establece el nombre de la función de devolución de llamada a enviar cuando se realiza una petición JSONP. De forma predeterminada el nombre es <code>callback</code>
<code>success</code>	Establece una función a ejecutar si la petición a sido satisfactoria. Dicha función recibe como argumentos la información de la petición (convertida a objeto JavaScript en el caso que <code>dataType</code> sea JSON), el estatus de la misma y el objeto de la petición en crudo.
<code>timeout</code>	Establece un tiempo en milisegundos para considerar a una petición como fallada. <code>traditional</code> Si su valor es <code>true</code> , se utiliza el estilo de serialización de datos utilizado antes de jQuery 1.4. Para más detalles puede visitar api.jquery.com/jquery.param .

type	De forma predeterminada su valor es GET. Otros tipos de peticiones también pueden ser utilizadas (como PUT y DELETE), sin embargo pueden no estar soportados por todos los navegadores.
url	Establece la URL en donde se realiza la petición. La opción url es obligatoria para el método \$.ajax;

11.2. METODOS CONVENIENTES

En caso que no quiera utilizar el método \$.ajax, y no necesite los controladores de errores, existen otros métodos más convenientes para realizar peticiones Ajax (aunque, como se indicó antes, estos están basados el método \$.ajax con valores pre-establecidos de configuración).

Los métodos que provee la biblioteca son:

\$.get	Realiza una petición GET a una URL provista.
\$.post	Realiza una petición POST a una URL provista.
\$.getScript	Añade un script a la página.
\$.getJSON	Realiza una petición GET a una URL provista y espera que un dato JSON sea devuelto.

Los métodos deben tener los siguientes argumentos, en orden:

Parámetro	Descripción
url	La URL en donde se realizará la petición. Su valor es obligatorio.
data	La información que se enviará al servidor. Su valor es opcional y puede ser tanto un objeto como una cadena de datos (como foo=bar&baz=bim).Nota: esta opción no es valida para el método \$.getScript.
success	callback Una función opcional que se ejecuta en caso que petición haya sido satisfactoria. Dicha función recibe como argumentos la información de la petición y el objeto en bruto de dicha petición.
data type	El tipo de dato que se espera recibir desde el servidor. Su valor es opcional. Nota: esta opción es solo aplicable para métodos en que no está especificado el tipo de dato en el nombre del mismo método.

11.2.1. Ejemplos

```
// obtiene texto plano o html
$.get('/users.php', { userId : 1234 }, function(resp) { console.log(resp); });

// añade un script a la página y luego ejecuta la función especificada
$.getScript('/static/js/myScript.js', function() { functionFromMyScript(); });
```

```
// obtiene información en formato JSON desde el servidor
$.getJSON('/details.php', function(resp) {
    $.each(resp, function(k, v) {
        console.log(k + ' : ' + v);
    });
});
```

11.3. \$.FN.LOAD

El método `$.fn.load` es el único que se puede llamar desde una selección. Dicho método obtiene el código HTML de una URL y rellena a los elementos seleccionados con la información obtenida. En conjunto con una URL, es posible especificar opcionalmente un selector, el cual obtendrá el código especificado en dicha selección.

11.3.1. Ejemplos

```
$('#container').load('./tabla.html');
$('#newContent').load('/titulo.html #myDiv h1:first', function(html) {
    alert('Contenido actualizado');
});
```

11.4. LA ESPECIFICACION CORS

Al crear aplicaciones complejas, en algún momento, por lo general se hace necesario hacer peticiones Ajax a dominios diferentes al actual. Hasta hace poco, esto era posible debido a las políticas de seguridad de los navegadores y JavaScript. Diversas técnicas se han empleado para evitar este tipo de restricciones de seguridad, por ejemplo, la utilización de proxies del lado del servidor, JSONP, y proxies iframe con mensajes.

La CORS (Cross Origin Resource Sharing) es una especificación que dice como realizar peticiones entre dominios y es la especificación recomendada por la W3C ya que, ahora sí, esta especificación es compatible con todos los navegadores.

11.4.1. Cómo funciona

La especificación CORS funciona añadiendo cabeceras HTTP a las solicitudes HTTP entre dominios y respuestas. Estas cabeceras indican el origen de la solicitud y el servidor debe indicar a través de otras cabeceras en la respuesta si va a servir a los recursos para este origen. Este intercambio de cabeceras es lo que hace CORS un mecanismo seguro. El servidor debe soportar CORS e indicar que se permite el dominio del cliente que realiza la solicitud en ese sentido. Lo interesante de esta especificación es que es

manejada automáticamente por los desarrolladores y las aplicaciones web no tienen que preocuparse por los detalles.

11.4.2. Cabeceras para la CORS

Cabecera	Descripción
Access-Control-Allow-Origin	Especifica el dominio o dominios permitidos. Si se establece a * se admite todo.
Access-Control-Allow-Methods	Especifica los métodos aceptados. POST, GET, OPTIONS, PUT, DELETE, ...
Access-Control-Allow-Credentials	Especifica si admite credenciales para su utilización.
Access-Control-Allow-Headers	Es una lista separada por comas de los encabezados personalizados que se utilizan. Es opcional y lo más frecuentes son origin, content-type, accept.
Access-Control-Max-Age	Es el tiempo en segundos que tienen que esperar las peticiones antes de ser cacheadas.
Access-Control-Expose-Headers	Es una lista separada por comas de nombres de encabezado. Puede añadir esta cabecera manualmente a la respuesta, o en función de la biblioteca que está usando, se podría añadir de forma diferente. Es opcional y, normalmente, no tiene mucho uso.

11.4.3. Ejemplo

```
$ .get (url: 'http://otherdomain.com/some-resource').done SuccessFn
```

El navegador emitirá una petición con los siguientes encabezados

GET http://otherdomain.com/some-resource/ HTTP/1.1

Referer: http://mydomain.com/myapp/

Origin: http://mydomain.com

Para que esta petición pueda ser atendida el servidor deberá tener una cabecera como:

Access-Control-Allow-Origin: http://mydomain.com

Content-Type: application / json

Si establecemos algún parámetro como headers a un valor determinado, lo deberemos poner también en el servidor como permitido para que la petición se resuelva.

12. Evitar conflictos con otras bibliotecas JavaScript

Si esta utilizando jQuery en conjunto con otras bibliotecas JavaScript, las cuales también utilizan la variable \$, pueden llegar a ocurrir una serie de errores. Para poder solucionarlos, es necesario poner a jQuery en su modo "no-conflicto". Esto se debe realizar inmediatamente después que jQuery se cargue en la página y antes del código que se va a ejecutar.

Cuando se pone a jQuery en modo "no-conflicto", la biblioteca ofrece la opción de asignar un nombre para reemplazar a la variable \$.

Poner a jQuery en modo no-conflicto

```
// la biblioteca prototype también utiliza $
<script src="prototype.js"></script>

// se carga jquery en la página
<script src="jquery.js"></script>

// se inicializa el modo "no-conflicto"
<script>var $j = jQuery.noConflict();</script>
```

También es posible seguir utilizando \$ conteniendo el código en una función anónima autoejecutable. éste es un patrón estándar para la creación de extensiones para la biblioteca, ya que \$ queda encerrada dentro del alcance de la misma función anónima.

Utilizar \$ dentro de una función anónima autoejecutable

```
<script src="prototype.js"></script>
<script src="jquery.js"></script>
<script>
jQuery.noConflict();

(function($) {
    // el código va aquí, pudiendo utilizar $
})(jQuery);
</script>
```

13. Creación de Plugins

Algunas veces necesitamos funciones específicas que sean capaces de interactuar con varios elementos del DOM y a su vez con otras funciones, ya sean similares o distintas. JQuery, a parte de ser una de los mejores frameworks de JavaScript, nos permite extenderlo para así poder crear nuevas funcionalidades. Con la creación de plugins se empieza a “crear” el concepto de reutilizable aunque, no hay que olvidar que para que un software, aplicación, plugin,... para que sea reutilizable primero, debe ser utilizable.

13.1. OBJETOS A EXTENDER

Para la creación de plugins con JQuery, hay dos objetos básicos que se extenderán: El **objeto JQuery**, que se encarga de prácticamente todo el procesamiento interno y el objeto **JQuery.fn**, que es el que maneja la interacción con elementos.

Para que este concepto se entienda mejor expondremos un par de ejemplos:

13.1.1. Ejemplo de definición JQuery

JQuery puede ser tratado como un objeto cualquiera y, por ello, se le puede añadir una función más:

```
jQuery.alert = function(mensaje){
    alert(mensaje)
}
```

La forma de llamar a nuestro plugin de JQuery será a través de \$.alert() o a través de jQuery.alert().

13.1.2. Ejemplo de definición JQuery.fn

```
/** jQuery Check Only Numbers Plugin
 * @version: 1.00
 * @author: Pablo E. Fernández (islavisual@gmail.com).
 * Copyright 2016 Islavisual.
 * Licensed under MIT (https://github.com/islavisual/checkOnlyNumbers/blob/master/LICENSE).
 * Last update: 31/03/2016
 */
jQuery.fn.checkOnlyNumbers = function(){
    return this.each(function() {
        if(('ontouchstart' in document.documentElement)){
            $(this).prop("type", 'number'); return;
        };

        $(this).keydown(function(e){
            var key = e.which || 0;
            return (key == 8 || key == 9 || key == 13 || key == 46 || key == 110 || key == 190 || (key
            >= 35 && key <= 40) || (key >= 48 && key <= 57) || (key >= 96 && key <= 105) || (key >= 112 && key <=
            123));
        });
    });
};
```

La forma de llamar a nuestro plugin de JQuery será a través de \$(selector).checkOnlyNumbers() o a través de jQuery(selector).checkOnlyNumbers().

13.2. EL METODO EXTEND()

Este es el método que nos va a ayudar a incrementar funcionalidad a jQuery, pues todo el código de nuestro plugin debe de ir definido dentro de él. Por lo tanto si queremos añadir un método llamado **red()** que cambie el color de los elementos, esta sería la estructura:

```
jQuery.fn.extend(  
  red: function() {  
    /*Recorre todos los elementos encapsulados*/  
    this.each(function()) {  
      /*Aquí se cambia el contexto, por lo que 'this' se refiere al elemento  
      DOM por el que se está pasando*/  
      jQuery(this).css("color", "#F00");  
    }  
  }  
);
```

13.3. ESTABLECIENDO PARAMETROS POR DEFECTO

Si se desea establecer unos parámetros por defecto sólo debemos añadir una instrucción.

```
var settings = $.extend({  
  duration: 500,  
  booleano: true,  
  param01: "valor 01",  
  param02: "valor 02"  
}, options );
```

Esta instrucción lo que hace es mezclar las opciones los parámetros por defecto de jQuery con los definidos por nosotros. De esta manera podremos utilizar más tarde tanto los parámetros que nos brinda jQuery como los nuestros personalizados para el plugin actual.

13.4. CREANDO UN PLUGIN CON PARÁMETROS Y VALORES POR DEFECTO

```
jQuery.fn.textTransform = function() {  
  var settings = $.extend({  
    transform: "uppercase"  
  }, options);  
  this.each(function()) {  
    jQuery(this).css("text-transform", settings.transform);  
  }  
};
```

Si no ponemos un parámetro se establecerá todo a mayúsculas, de lo contrario, se establecerá lo que enviemos en el parámetro (capitalize, lowercase,...).

14. Mejorar el rendimiento

14.1. GUARDAR LA LONGITUD DE LOS BUCLES

En un bucle, no es necesario acceder a la longitud de un vector cada vez que se evalúa la condición; dicho valor se puede guardar previamente en una variable y evitar su evaluación en cada iteración.

```
var myLength = myArray.length;
for (var i = 0; i < myLength; i++) {
    // do stuff
}
```

14.2. AÑADIR NUEVO CONTENIDO POR FUERA DE UN BUCLE

Si se va a insertar muchos elementos en el DOM, debe hacerse de una sola vez.

```
// En vez de añadir elementos al DOM uno a uno...
$.each(myArray, function(i, item) {
    var newListItem = '<li>' + item + '</li>';
    $('#ballers').append(newListItem);
});

// Es mejor convertirlo en anidamiento y asignación final
var myHtml = '';

$.each(myArray, function(i, item) {
    myHtml += '<li>' + item + '</li>';
});
$('#ballers').html(myHtml);
```

14.3. UTILIZAR LA DELEGACIÓN DE EVENTOS

La delegación de eventos permite vincular un controlador de evento a un elemento contenedor (por ejemplo, una lista desordenada) en lugar de múltiples elementos contenidos (por ejemplo, los ítems de una lista). Es mejor utilizar la delegación de eventos a través de “on” en vez de a través de “live”, “bind”, o “delegate”. Si seguimos esta metodología, conseguiremos hasta un 80% más de rendimiento en el proceso de carga.

```
$("#lista li").on("click", function () { ... });
```

14.4. ASIGNAR OBJETOS A VARIABLES

Cuando se está programando en jQuery es muy habitual utilizar expresiones del tipo:

```
$(e.target).val();  
$(this).val('...');
```

Esta forma de manipular los elementos en nuestra página puede bajar mucho el rendimiento ya que, cada vez que ponemos estas expresiones, se debe re-evaluar la función \$ y más si utilizamos este tipo de expresiones muchas veces. Es mejor introducirlo en una variable.

```
var $val = $(e.target).val();  
var $this = $(this).val('...');
```

14.5. NO ABUSAR DE \$.FN.CSS

En la medida de lo posible, no se debe abusar de la asignación de estilos a través de jQuery ya que esta funcionalidad baja notablemente el rendimiento de la página. Es mejor, si se puede, realizarlo desde CSS.

```
$('#name').css('display', 'none');  
$('#code').css('visibility', 'hidden');
```

14.6. NO ABUSAR DE \$(THIS)

El uso de \$(this) puede significar retrasos y bajadas de rendimiento importantes ya que cada vez que se utiliza este objeto se debe de evaluar.

14.7. UTILIZAR \$.DATA EN LUGAR DE \$.FN.DATA

Utilizar \$.data en un elemento del DOM en lugar de \$.fn.data en una selección puede ser hasta 10 veces más rápido. Antes de realizarlo, este seguro de comprender la diferencia entre un elemento DOM y una selección jQuery.

```
// En vez de utilizar esta expresión...  
$(elem).data(key,value);  
  
// utilizar esta...  
$.data(elem,key,value);
```

14.8. EVITAR INTERACCIONES CON ELEMENTOS VACÍOS

Cuando se trabaja con objetos dinámicos es frecuente encontrarse con selecciones que dan como resultado el conjunto vacío. Esto puede llegar a provocar excepciones inesperadas y/o retrasos en la ejecución del código. Para evitarlo es adecuado utilizar:

```
if(typeof item != "undefined") { ... }  
if($list.length != 0) { ... }
```

14.9. EVITAR EL USO DE \$.EACH

Es muy frecuente que los programadores recorran los objetos con el método “each”. Esto produce una bajada del rendimiento importante ya que, cada vez que empieza el bucle, hay mapear todo el objeto contenedor, buscar el elemento y reasignarlo a la variable. Es mucho más eficiente utilizar la clausula “for”

```
// En vez de utilizar esta expresión...  
$('div a').each(function(){  
    ...  
});  
  
// utilizar esta ...  
var items = $('div a');  
for(var x = 0; x < items.length; ++x){  
    var item = $(items[x]);  
    ...  
}
```

15. Referencias

<http://librosweb.es/>
<http://librojquery.com/>
<https://es.wikipedia.org/>