UTN.BA FACULTAD REGIONAL BUENOS AIRES
SECRETARÍA DE EXTENSIÓN UNIVERSITARIA FRBA UTN

Centro de e-Learning

Javascript, JQuery y JSON avanzado.

Módulo 1

INTRODUCCIÓN A JQUERY

Pasos para la utilización del framework.

Función JQuery. Eventos, selectores y efectos.

Presentación de la Unidad:

En esta unidad aprenderán que es el framework JQuery y cuales son los beneficios de su utilización.

Veremos paso a paso como instalarlo y comenzaremos a introducirnos en su sintaxis y modo de uso.

Veremos que es la función JQuery, que es y para que nos sirve el document.ready. Utilizaremos eventos, selectores y efectos.

Objetivos:

- Aprender a seleccionar los elementos HTML utilizando los selectores de JQuery.
- Aprender a utilizar eventos y efectos y como combinarlos.
- Aprender que es y para que nos sirve el document.ready.

Temario:

Consignas para el aprendizaje colaborativo

En esta Unidad los participantes se encontrarán con diferentes tipos de consignas que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

1. Los foros asociados a cada una de las unidades.
2. La Web 2.0.
3. Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen las actividades sugeridas y compartan en los foros los resultados obtenidos.

Qué es JQuery

Para simplificar, podríamos decir que jQuery es un framework Javascript, pero quizás muchos de los lectores se preguntarán qué es un framework. Pues es un producto que sirve como base para la programación avanzada de aplicaciones, que aporta una serie de funciones o códigos para realizar tareas habituales. Por decirlo de otra manera, framework son unas librerías de código que contienen procesos o rutinas ya listos para usar. Los programadores utilizan los frameworks para no tener que desarrollar ellos mismos las tareas más básicas, puesto que en el propio framework ya hay implementaciones que están probadas, funcionan y no se necesitan volver a programar.

Por ejemplo, en el caso que nos ocupa, jQuery es un framework para el lenguaje Javascript, luego será un producto que nos simplificará la vida para programar en este lenguaje. Como probablemente sabremos, cuando un desarrollador tiene que utilizar Javascript, generalmente tiene que preocuparse por hacer scripts compatibles con varios navegadores y para ello tiene que incorporar mucho código que lo único que hace es detectar el browser del usuario, para hacer una u otra cosa dependiendo de si es Internet Explorer, Firefox, Opera, etc. jQuery es donde más nos puede ayudar, puesto que implementa una serie de clases (de programación orientada a objetos) que nos permiten programar sin preocuparnos del navegador con el que nos está visitando el usuario, ya que funcionan de exacta forma en todas las plataformas más habituales.

Así pues, este framework Javascript, nos ofrece una infraestructura con la que tendremos mucha mayor facilidad para la creación de aplicaciones complejas del lado del cliente. Por ejemplo, con jQuery obtendremos ayuda en la creación de interfaces de usuario, efectos dinámicos, aplicaciones que hacen uso de Ajax, etc. Cuando programemos Javascript con jQuery tendremos a nuestra disposición una interfaz para programación que nos permitirá hacer cosas con el navegador que estemos seguros que funcionarán para todos nuestros visitantes. Simplemente debemos conocer las librerías del framework y programar utilizando las clases, sus propiedades y métodos para la consecución de nuestros objetivos.

Además, todas estas ventajas que sin duda son muy de agradecer, con jQuery las obtenemos de manera gratuita, ya que el framework tiene licencia para uso en cualquier tipo de plataforma, personal o comercial. Para ello simplemente tendremos que incluir en nuestras páginas un script Javascript que contiene el código de jQuery, que podemos descargar de la propia página web del producto y comenzar a utilizar el framework.

El archivo del framework ocupa unos 56 KB, lo que es bastante razonable y no retrasará mucho la carga de nuestra página (si nuestro servidor envía los datos comprimidos, lo que es bastante normal, el peso de jQuery será de unos 19 KB). Además, nuestro servidor lo enviará al cliente la primera vez que visite una página del sitio. En siguientes páginas el cliente ya tendrá el archivo del framework, por lo que no

necesitará transferirlo y lo tomará de la caché. Con lo que la carga de la página sólo se verá afectada por el peso de este framework una vez por usuario. Las ventajas a la hora de desarrollo de las aplicaciones, así como las puertas que nos abre jQuery compensan extraordinariamente el peso del paquete.

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.

Pasos para la instalación

El sitio oficial de JQuery es jquery.com. Ingresamos al sitio y vemos a la izquierda un botón para descargar el framework. Si lo presionamos vamos a la página desde la cual se puede descargar JQuery en la versión que nos interese.

Dan dos posibilidades en cuanto a la versión, una es la versión 1.x y la otra es la 2.x. La diferencia entre ellas es que la 2.x no funciona en Internet Explorer 6, 7 y 8. A su vez dentro de estas dos versiones tenemos otras dos posibilidades para descargar, una que le llaman PRODUCTION, que es la adecuada para páginas web en producción, puesto que está minimizada y ocupa menos espacio, con lo que la carga de nuestro sitio será más rápida. La otra posibilidad es descargar la versión que llaman DEVELPOMENT, que está con el código sin comprimir, con lo que ocupa más espacio, pero se podrá leer la implementación de las funciones del framework, que puede ser interesante en etapa de desarrollo, porque podremos bucear en el código de jQuery por si tenemos que entender algún asunto del trabajo con el framework.

Nosotros vamos a utilizar la versión 1.x compressed, o production JQuery.

Bajamos entonces dicha versión, que como verán, no es mas que un archivo con extensión .js que deberemos incluir en nuestro HTML a través de una etiqueta <script> para poder hacer uso de las distintas clases y métodos del framework. Nos quedará de la siguiente manera:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Inclusion de JQuery en HTML</title>
<script type="text/javascript" src="jquery-1.10.2.min.js"></script>
</head>

<body>
</body>
</html>
```

Ejecutar código una vez que la página esté lista

El paso que vamos a explicar ahora es importante que se entienda. Se trata de detectar el momento en que la página está lista para recibir comandos Javascript que hacen uso del DOM.
Cuando hacemos ciertas acciones complejas con Javascript tenemos que estar seguros que la página haya terminado de cargar y esté lista para recibir comandos Javascript que utilicen la estructura del documento con el objetivo de cambiar cosas, como crear elementos, quitarlos, cambiar sus propiedades, etc. Si no esperamos a que la página esté lista para recibir instrucciones corremos un alto riesgo de obtener errores de Javascript en la ejecución.
Cuando queremos hacer acciones con Javascript que modifiquen cualquier cosa de la página, tenemos que esperar a que la página esté lista para recibir esos omandos.
Generalmente, cuando se desea ejecutar Javascript después de la carga de la página, si no utilizamos ningún framework, lo más normal será utilizar un código como este:

```
window.onload = function () {
alert("cargado...");
}
```

Pero esta sentencia, que carga una funcionalidad en el evento onload del objeto window, sólo se ejecutará cuando el navegador haya descargado completamente TODOS los elementos de la página, lo que incluye imágenes, iframes, banners, etc. lo que puede tardar bastante, dependiendo de los elementos que tenga esa página y su peso.
Pero en realidad no hace falta esperar todo ese tiempo de carga de los elementos de la página
para poder ejecutar sentencias Javascript que alteren el DOM de la página. Sólo habría que hacerlo cuando el navegador ha recibido el código HTML completo y lo ha procesado al renderizar la página. Para ello, jQuery incluye una manera de hacer acciones justo cuando ya está lista la página, aunque haya elementos que no hayan sido cargados del todo. Esto se hace con la siguiente sentencia.

```
$(document).ready(function(){
//código a ejecutar cuando el DOM está listo para recibir instrucciones.
});
```

Por dar una explicación a este código, digamos que con $(document) se obtiene una referencia al documento (la página web) que se está cargando. Luego, con el método ready() se define un evento, que se desata al quedar listo el documento para realizar acciones sobre el DOM de la página.

Función JQuery() o $()

La función jQuery, también conocida como $() es fundamental para comenzar a trabajar con JQuery, el funcionamiento del Core de jQuery se basa en ella. Como dicen en la propia documentación del framework, "Todo en jQuery está basado en esta función o la usa de alguna forma".

La función jQuery sirve para hacer varias cosas, según los parámetros que le pasemos. El uso más simple es seleccionar elementos o grupos de elementos de una página web, pero también sirve para crear elementos sobre la marcha o para hacer un callback de funciones, que veremos mas adelante.

Esta función se invoca también con el símbolo pesos $(), lo que sería una manera resumida de utilizarla, así es como la verán generalmente y así es como la utilizaremos.

Veamos los distintos usos de esta función, según los parámetros que reciba.

Función jquery enviando un selector y un contexto

Este uso de la función sirve para seleccionar elementos de la página. Recibe una expresión y se encarga de seleccionar todos los elementos de la página que corresponden con ella, devolviendo un **objeto jQuery** donde se encuentran todos los elementos de la página seleccionados y extendidos con las funcionalidades del framework. Adicionalmente, podemos pasarle un contexto de manera opcional. Si no se le envía un contexto, selecciona elementos del documento completo, si indicamos un contexto conseguiremos seleccionar elementos sólo dentro de ese contexto.

La expresión que debemos enviar obligatoriamente como primer parámetro sirve de selector. En ese parámetro tenemos que utilizar una notación especial para poder seleccionar elementos de diversas maneras y la verdad es que las posibilidades de selección con jQuery son muy grandes, como veremos en el tema de "Selectores".

Este paso de selección de elementos es básico en cualquier script jQuery, para poder actuar en cualquier lugar de la página y hacer nuestros efectos y utilidades Javascript

sobre el lugar que deseemos.

Veamos un uso de esta función:

```
var elem1 = $("#elem1");
```

Con esta línea de código habremos seleccionado un elemento de la página que tiene el identificador (atributo id del elemento HTML) "elem1" y lo hemos guardado en una nueva variable llamada elem1. La variable elem1 guardará entonces lo que se llama el objeto jQuery con el cual podremos trabajar, solicitando métodos o funciones de dicho objeto, para trabajar con el elemento seleccionado.

Luego, podríamos hacer cualquier cosa con ese elemento seleccionado, como lo siguiente:

```
elem1.css("background-color", "#ff9999");
```

Este método css() sirve para cambiar atributos CSS de un elemento, entre otras cosas. Así pues, con esa línea cambiaríamos el color de fondo del elemento seleccionado anteriormente, que habíamos guardado en la variable elem1.

Ahora veamos otro ejemplo de la selección de elementos con la función dólar.

```
var divs = $("div");
divs.css("font-size", "32pt");
```

Aquí seleccionamos todas las etiquetas DIV de la página, y les colocamos un tamaño de letra de 32pt.

El código de esta página sería el siguiente:

ejemplo1.html

```
<html>
<head>
  <title>función jquery</title>
  <script type="text/javascript" src="jquery-1.10.2.min.js"></script>
<script>
$(document).ready(function(){
  var elem1 = $("#elem1");
  //podríamos haber escrito: var elem1 = jQuery("#elem1");
  elem1.css("background-color", "#ff9999");

  var divs = $("div");
  //podríamos haber escrito: var elem1 = jQuery("#elem1");
  divs.css("font-size", "32pt");
});
```

```
</script>
</head>
<body>
<div id="elem1">Este elemento se llama elem1</div>
<div id="elem2">Este otro elemento se llama elem2</div>
</body>
</html>
```

Me gustaría aclarar dos cosas importantes de este código:

La primera es que JQuery trabaja con el concepto de funciones anónimas, es decir una función que se define y ejecuta al mismo tiempo. Nosotros estamos acostumbrados a definir primero una función con su código y luego hacer el llamado a la misma para que se ejecute.

La otra es la función ready() en la línea:

```
$(document).ready(function(){
```

ésta hace referencia a una función a ejecutar una vez que el DOM (Document Object Model) de la página está listo. Esto es así porque desde JQuery voy a manipular los elementos del html y para poder hacerlo necesito que el árbol de nodos que vimos en la unidad I esté terminado.

Es por esto que comúnmente todo el código de JQuery que tengamos en nuestra página lo vamos a colocar dentro de esta función:

```
$(document).ready(function(){
……….
}
```

Otros usos de la función $()

Función jquery pasando un html

Una posibilidad de trabajar con la función jQuery es enviarle un string con un HTML. Esto crea esos elementos en la página y les coloca los contenidos que se indique en el string. Ojo, que el HTML tiene que estar bien formado para que funcione en cualquier navegador, esto es, que se coloquen etiquetas que se puedan meter en el BODY de la página y que todas las etiquetas tengan su cierre.

```
var nuevosElementos = $("<div>Elementos que creo en <b>tiempo de ejecución</b>.<h1>En ejecución...</h1></div>");
```

Esto nos creará en la variable nuevosElementos los elementos HTML que hemos especificado en el string. Luego podríamos hacer lo que queramos con ellos, como colocarlos en la página con el método appendTo(), por ejemplo de esta manera:

```
nuevosElementos.appendTo("body");
```

Veamos el código completo de una página que hace uso de este ejemplo:

ejemplo2.html

```
<html>
<head>
  <title>función jquery</title>
  <script type="text/javascript" src="jquery-1.10.2.min.js"></script>
<script>
$(document).ready(function(){
  var nuevosElementos = $("<div>Estos elementos ..</b>.<h1>Título...</h1></div>");
  nuevosElementos.appendTo("body");
});

</script>
</head>
<body>
<p>Esto es el cuerpo de la página, que tiene poca cosa...</p>
</body>
</html>
```

Función jquery pasando un elemento

Otro posible valor que se le puede enviar a la función jQuery es un elemento o una jerarquía de elementos del DOM, para extenderlos con las funcionalidades que aporta el framework para los elementos.

Por ejemplo:

```
var documento = $(document.body);
documento.css("background-color", "#ff8833");
```

Con la primera línea creamos una variable llamada documento, a la que asignamos el valor que devuelve el método $() enviando el parámetro document.body.

Con ello obtenemos un objeto que es el cuerpo de la página (document.body) al que le hemos agregado todas las funcionalidades del framework jQuery para los elementos.

Así pues, en la línea siguiente, invocamos el método css() sobre la variable "documento", que es el propio documento de la página extendido. Por eso el método css(), que es de jQuery(), funciona sobre ese objeto.

El código completo sería:

ejemplo3.html

```
<html>
<head>
  <title>función jquery</title>
  <script type="text/javascript" src="jquery-1.10.2.min.js"></script>
<script>
$(document).ready(function(){
  var documento = $(document.body);
  documento.css("background-color", "#ff8833");
});

</script>
</head>
<body>
<p>Página 3 </p>
</body>
</html>
```

Función jquery pasando una función

En la función $() una última posibilidad es pasarle como parámetro una función y entonces lo que tenemos es una función callback que se invoca automáticamente cuando el DOM está listo.

Selectores

Como la propia palabra indica, los selectores son un mecanismo, disponible en jQuery, para seleccionar determinados elementos de la página. El selector no es más que una cadena de caracteres, creada bajo unas normas que veremos a continuación, con la que podemos referirnos a cualquiera o cualesquiera de los elementos que hay en una página.

Todo en jQuery pasa por utilizar los selectores, para acceder a los elementos de la página que deseamos alterar dinámicamente con Javascript.

Una de las cosas que más potentes de jQuery son los selectores, veremos a continuación cómo utilizarlos y aprovecharnos de su potencia.

Para empezar, veamos un selector, cuando utilizamos la función jQuery (o función pesos) lo que pasamos como parámetro es el selector. La función jQuery devuelve justamente los elementos de la página que concuerdan con el selector enviado por parámetro.

```
$("p");
```

En esa llamada a la función jQuery, estamos pasando por parámetro una cadena "p" y como decía, esa misma cadena es el selector. En este caso, "p" es un selector que sirve para seleccionar todas las etiquetas P de la página, es decir, los párrafos.

Selectores básicos en jquery

Los selectores, al menos los más básicos, son parecidos, o iguales, a los que se utilizan en CSS para seleccionar los elementos a los que se desean aplicar ciertos estilos.

Selector de etiquetas:
Simplemente indicamos la etiqueta a la que deseamos referirnos, es decir, la etiqueta que queremos seleccionar. Obtendremos con él todas las etiquetas de la página indicada en el selector.

```
$("h1") //selecciona todos los encabezados de nivel 1
```

Selector por identificador:
Sirven para seleccionar los elementos que tengan un identificador dado, que se asigna a las etiquetas a través del atributo id del HTML. Para utilizar este selector se indica primero el carácter "#" y luego el identificador de cuyo elemento se desee seleccionar.

```
$("#idelemento") //selecciona una etiqueta que tiene el atributo id="idelemento"
```

Selector por clase:
Podemos indicar el nombre de una clase (class de CSS) y seleccionar todos los elementos a los que se ha aplicado esta clase. Para ello, como en CSS, comenzamos colocando el carácter "." y luego el nombre de la clase que deseamos seleccionar.

```
$(".miclase") //selecciona todos los elementos que tienen el atributo class="miclase"
```

Selector por varias clases:
Si lo deseamos, podemos indicar varias clases CSS, para obtener todos los elementos que tienen esas clases aplicadas: todas al mismo tiempo. Esto se consigue comenzando por un ".", igual que los selectores de clases, y luego otro "." para separar las distintas clases que queremos utilizar en el selector.

```
$(".clase1.clase2") //selecciona los elementos que tienen class="clase1 clase2"
```

Selector asterisco "*":
Nos sirve para seleccionar todos los elementos de la página.

```
$("*") //selecciona todos los elementos que tiene la página
```

Concatenar varios selectores distintos:
Por último, podemos utilizar varios selectores, para obtener todas las etiquetas que cumplen uno de ellos. No hace falta que cumplan todos los selectores a la vez, sino con

que uno de ellos concuerde es suficiente. Para ello colocamos todos los selectores que deseamos, separados por una coma ",".

```
$("div,p") //selecciona todos los elementos división y párrafo
$(".clase1,.clase2") //selecciona los elementos que tienen la clase "clase1" o "clase2"
$("#miid,.miclase,ul) //selecciona el elemento con id="miid", los elementos con class="miclase" y todas las listas UL
```

Hasta este punto hemos visto los selectores básicos de jQuery, que nos servirán para hacer la mayoría de nuestros ejemplos y resolver también la mayor parte de las necesidades de selección de elementos que nos podamos encontrar en ejemplos reales. Sin embargo, el framework Javascript incluye una buena gama de selectores adicionales que pueden venirnos bien en algunos casos más concretos.

Si quieren profundizar sobre el tema de selectores pueden visitar la documentación de JQuery donde encontrarán toda la gama de selectores posibles en http://api.jquery.com/category/selectors/

Selectores de jerarquía en JQuery

Existen varios otros tipos de selectores, junto con algunos filtros, que hacen todavía más potente el framework de cara a acceder a las etiquetas o elementos que deseamos seleccionar.

Sabemos que la página está compuesta por etiquetas HTML que se meten unas dentro de otras, formando una jerarquía de etiquetas o de elementos. Los selectores de Jerarquía permiten utilizar la propia estructura de la página para acceder a unos elementos dados, que se seleccionan a través de la jerarquía existente de etiquetas en la página. Dentro de éstos, existen a su vez varias posibilidades, que hacen uso de criterios de descendencia, ascendencia, siguiente, anterior, etc.

Selector ancestor descendant:
Sirve para seleccionar elementos de la página que son descendientes de otro y que además se corresponden con un selector dado. Para este selector se indican dos datos, separados por un espacio. Primero el selector para definir el elemento o elementos antecesores y el segundo selector para definir el tipo de elementos que se tienen que seleccionar de entre los descendientes.

```
$("p b") //selecciona todas las etiquetas B que hay dentro de las etiquetas P
$("p.parraforojo i") //selecciona todas las etiquetas I que hay dentro de los párrafos con clase "parraforojo".
$("table.mitabla td") //selecciona todas las etiquetas TD que hay en las tablas que tienen class="mitabla"
```

Selector parent > child:
Con el selector parent > child podemos acceder a elementos que sean hijos directos de

otros. Para ello indicamos un selector como "parent" y un selector como "child". Nos seleccionará todos los elementos que son hijos directos de parent y que concuerdan con el selector child.

$("p > b") //selecciona todas las etiquetas B que son hijas directas de los párrafos.
$("#capa > *") //selecciona todas las etiquetas que son hijas directas del elemento con id="capa"

Selector prev + next:
Con este selector conseguimos acceder a las elementos que están después de otros, es decir, a las etiquetas que concuerdan con el selector "next", que se abren después de cerrar las etiquetas que concuerdan con el selector "prev".

$("p.parraforojo + p") //Esto selecciona los párrafos que están después de cualquier párrafo que tenga la clase "parraforojo"
$("i + b") //selecciona todas las negritas (etiqueta B) que hay después de una itálica (etiqueta I)

Selector prev ~ siblings:
Selecciona los elementos hermanos que hay a continuación de los elementos que concuerden con el selector "prev", que son del tipo que se especifica con el selector "siblings". Los elementos hermanos son los que están en el mismo contenedor y se encuentran en el mismo nivel de jerarquía.

$("#miparrafo ~ table") //selecciona los elementos TABLE que son hermanos del elemento con id="miparrafo"
$("#a2 ~ div.clase") //selecciona los elementos hermanos del que tiene el id="a2" que sean etiquetas DIV con la class="clase".

Manipular atributos de HTML

Mediante el método attr() podremos trabajar con los atributos de los elementos de la página. Este método, como muchos otros en jQuery tiene diferentes usos, dependiendo de los parámetros que le pasemos, pero siempre sirve para trabajar con los atributos HTML, como pueden ser title, height, width, href, value, etc.

El uso es bien simple. Dado un objeto jQuery, invocando el método attr() sobre él, podemos acceder a sus atributos, para recuperar sus valores, modificarlos o eliminarlos. Veremos los distintos usos conforme los parámetros que le pasemos.

El primer uso de attr() es para recuperar el valor de un atributo. En este caso, el método debe recibir una cadena con el nombre del atributo que queremos recuperar.

Ahora podríamos acceder a lo que hay escrito en el campo de texto de la siguiente manera:

```
$("#campotexto").attr("value")
```

Pero atención, en caso que el elemento no tenga definido ese atributo al que se pretenda acceder, devolvería undefined.

Ahora vamos a ver un uso de attr() en el que no leemos el atributo, sino que lo modificamos. En este caso la función recibe dos cadenas de texto, la primera con el nombre del atributo y la segunda con el nuevo valor que queremos asignar. Por ejemplo:

```
$('li').attr("type", "square");
```

Esto haría que todos los elementos de lista tengan un bullet de tipo cuadrado.

También podemos utilizar el método attr() pasando un objeto con pares atributo/valor. Esto sirve para modificar de una sola vez varios atributos sobre los elementos que haya en un objeto jQuery y si esos atributos no existían, simplemente los crea con los valores enviados en el objeto.

Imaginemos que tenemos varios enlaces en la página, y que queremos modificar sus atributos, para todos los enlaces a la vez.

```
$('a').attr({
'title': 'Title modificado por jQuery',
'href': 'http://www.google.com',
'style': 'color: #f80'
});
```

A partir de la ejecución de la sentencia anterior todos los title de los enlaces tendrán el valor "Title modificado por jQuery". Las URLs a las que enlazarán los link serán siempre la home de Google y además se les creará un estilo CSS para que sean de color naranja.

Podemos también enviar una función para procesar el valor que queremos asignar a un atributo. Para ello enviamos a attr() dos parámetros, el primero con el nombre del atributo y el segundo con la función que debe devolver el valor a asignar a dicho atributo.

Para ilustrar este uso de attr() mostraremos un ejemplo en el que desde jQuery accedemos a los elementos INPUT de la página que tienen la clase CSS "fecha" y le insertamos como texto a mostrar la fecha de hoy. Para obtener el día actual necesitamos procesar cierto código Javascript y para ello crearemos una función que devuelve la cadena de texto con la fecha.

```
$('input.fecha').attr("value", function(indiceArray){
//indiceArray tiene el índice de este elemento en el objeto jQuery
var f = new Date();
return f.getDate() + "/" + (f.getMonth() +1) + "/" + f.getFullYear();
```

```
});
```

Por último vamos a ver otro método distinto de los objetos jQuery, que sirve para borrar un atributo. Este sencillo método, llamado removeAttr(), simplemente recibe una cadena con el nombre del atributo que queremos eliminar y lo borra del elemento. Es decir, no es que se asigne un nuevo valor a un atributo, como ocurría con el método attr(), sino que ese atributo se borra por completo de la etiqueta, con lo cual no existirá en ningún caso, tomando el valor por defecto, si es que existe, que tenga configurado el navegador.

Para mostrarlo vamos a hacer un ejemplo en el que tenemos una celda de una tabla con nowrap, con lo que el texto de esa celda aparece todo en la misma línea. Luego quitamos el atributo y veremos que el texto de la celda se partirá en varias líneas. Esto lo hacemos simplemente enviando el valor "noWrap" al método removeAttr().

El código de este ejemplo es el siguiente.

```
<html>
<head>
<title>método removeAttr</title>
<script src="../jquery-1.3.2.min.js" type="text/javascript"></script>
<script>
$(document).ready(function(){
$("#boton").click(function(i){
$("td").removeAttr("noWrap");
});
});
</script>
</head>
<body>
<table width="50">
<tr>
<td nowrap>
Esta celda tiene un nowrap, con lo que todo el texto se muestra en la misma línea!
Pero realmente la tabla mide 50 pixeles de anchura, luego tendrían que aparece varias líneas!
</td>
</tr>
</table>
<input type="Button" id="boton" value="Quitar nowrap">
</body>
</html>
```

Un detalle es que en la línea que se hace la llamada al método removeAttr("noWrap"), el nombre del atributo "noWrap" tiene que estar escrito con la "W" mayúscula para que funcione en Explorer.

Método css() para modificar atributos de css

El método más importante para trabajar con las CSS de manera dinámica es justamente css() vamos a dedicarle un poco de tiempo para aprender todas sus posibilidades.

El método css() sirve tanto para recibir el valor de un atributo CSS como para asignarle un nuevo valor y su funcionamiento depende de los parámetros que podamos enviarle. Así que, para hablar sobre este método veremos cada uno de los posibles juegos de parámetros que podemos enviarle, explicando cada una de las opciones disponibles y ofreciendo diversos ejemplos.

.css(nombre_propiedad_css)

Si enviamos un solo parámetro al método CSS estamos indicando que queremos recibir el valor de una propiedad CSS. En este caso la función devolverá el valor del atributo CSS que le hayamos indicado.

Si tenemos un elemento en la página como este, al que le hemos colocado un identificador, atributo id="micapa":

```
<div id="micapa" style="color: red;">hola!</div>
```

Podremos acceder a alguna de sus propiedades css de la siguiente manera:

```
$("#micapa").css("color");
```

Esto nos devolverá el atributo "color" de ese elemento, que en este caso valía "color".

Como podemos suponer, el método CSS enviando un solo parámetro puede servir de mucha utilidad para obtener datos sobre los estilos actuales de nuestros elementos, no obstante, todavía es más utilizada la siguiente opción, en la que enviamos dos parámetros.

.css(nombre_propiedad_css, valor)

En este segundo caso, aparte del nombre de una propiedad CSS estamos enviando un segundo parámetro con un valor y nos servirá para asignar un nuevo estado a dicho atributo. Esta segunda manera de invocar al método CSS tiene además algunas variantes.

Cambiar un único atributo CSS: podemos enviar el nombre de un único atributo CSS y su nuevo valor.

```
$("#micapa").css("color", "green");
```

Con esto estaríamos cambiando el color del texto del elemento con id="micapa" y asignando el color verde ("green").

Cambiar varios atributos CSS al mismo tiempo: Podemos enviar todos los atributos CSS que deseemos y sus nuevos valores, en notación de objeto. Con esto conseguimos que, en una única llamada a css() se cambien varias propiedades a la vez.

```
$("#micapa").css({
  "background-color": "#ff8800",
  "position": "absolute",
  "width": "100px",
  "top": "100px",
  "left": "200px"
})
```

Como se puede ver, se estarían actualizando con la anterior llamada a css() varios atributos CSS, como el color de fondo, la posición del elemento, su anchura, etc.

Sobre este punto vamos a dar un ejemplo adicional que puede estar bien para aprender a variar un atributo CSS teniendo en cuenta el valor anterior que tuviera.

```
$("#micapa").mouseover(function(){
  antiguoLeft = parseInt($(this).css("left"));
  //alert (antiguoLeft);
  $(this).css("left", antiguoLeft + 10 + "px");
})
```

Con esto estamos definiendo un evento onmouseover sobre la capa con id="micapa", por lo que estas instrucciones se pondrán en ejecución cuando se pase el mouse por encima de la capa. Dentro del método estamos haciendo un par de cosas. Como primer paso estamos extrayendo el valor de la propiedad CSS "left" y convirtiéndola en un entero. Como segundo paso estamos actualizando ese valor de "left" y asignando un nuevo valor que sería 10 píxeles más que el valor antiguo. Para ello sumamos 10 al valor antiguo de "left" y lo concatenamos con la unidad de medida "px".

Cambiar un único atributo y colocar el valor según el resultado de una función: Este tercer uso es un poco más avanzado y está disponible sólo a partir de jQuery 1.4. Consiste en enviarle una función como segundo parámetro, en vez del valor directamente, para asignar al atributo el valor devuelto por esa función.

Esto es tan sencillo de poner en marcha como pasar una función que simplemente tenga un return. Pero hay un detalle y es que esa función recibe dos valores. El primero es el índice del elemento dentro del objeto jQuery que recibe el método y el segundo, más útil, sirve para obtener el valor actual que hay en el atributo que queremos cambiar.

Para ver este uso del método jQuery utilizaremoa el siguiente ejemplo.

```
$("#micapa").click(function(){
  $(this).css("width", function(index, value){
    //alert (value);
    var aumento = prompt("cuanto quieres aumentar?", "25");
    return (parseInt(value) + parseInt(aumento)) + "px";
  });
})
```

Como se puede ver, se define un evento clic sobre una capa. Luego utilizamos el método css() sobre el elemento, para cambiar el atributo width. El valor de width que se colocará será lo que devuelva la función indicada como segundo parámetro en el método css(). Si nos fijamos, la función devuelve un valor, que es lo que se colocará en el atributo width.

Como dijimos al comienzo css() es uno de los métodos mas utilizados para manipular el css de la página, además de éste en la documentación de JQuery se definen también otros métodos dentro de la categoría CSS. Algunos de estos métodos son los siguientes:

.addclass()

Añade la clase especificada (s) a cada uno del conjunto de elementos seleccionados.

.hasclass()

Determinar si a alguno de los elementos seleccionados se le asigna a la clase dada.

.height()

Obtener la altura computada actual para el primer elemento del conjunto de elementos seleccionados o establecer la altura de cada elemento seleccionado.

. InnerHeight ()

Obtener la altura computada actual para el primer elemento del conjunto de elementos seleccionados, incluyendo *padding* pero no *border*.

. InnerWidth ()

Obtener el ancho computada actual para el primer elemento del conjunto de elementos seleccionados, incluyendo *padding* pero no *border*.

.outerHeight()

Recibe un objeto jQuery y devuelve las dimensiones externas del primer elemento de dicho objeto jQuery recibido por parámetro, esto es, la altura del elemento contando

el padding del elemento y su borde.

.outerWidth()

Recibe un objeto jQuery y devuelve las dimensiones externas del primer elemento de dicho objeto jQuery recibido por parámetro, esto es, el ancho del elemento contando el padding del elemento y su borde.

.offset() y .position()

Ambos métodos devuelven la posición de un elemento en la página. Reciben un objeto jQuery y devuelven la localización del primer elemento que haya en ese objeto jQuery. La posición siempre se indica como valor de retorno del método por medio de un objeto que tiene dos atributos, "top" y "left", indicando los píxeles que está separado de la esquina superior izquierda del documento. La diferencia entre estos dos métodos es que offset() indica la posición del elemento real, teniendo en cuenta los márgenes del elemento, lo que suele ser más útil. Por su parte, position() indica la posición donde habría sido posicionado el elemento si no tuviera márgenes, lo que a menudo no es la posición real.

.removeClass()

Quitar una sola clase, varias clases, o todas las clases de cada elemento en el conjunto de elementos seleccionados.

.width()

Obtener el ancho computado actual para el primer elemento del conjunto de elementos seleccionados o configurar el ancho de cada elemento seleccionado.

Cada uno de estos métodos se encuentra muy bien explicado y ejemplificado en http://api.jquery.com/category/css/

Ejemplos

Veamos algunos ejemplos del uso de algunos métodos definidos dentro de la categoría CSS.

Ejemplo5.html

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Mostrar Ocultar</title>
<script type="text/javascript" src="jquery-1.10.2.min.js"></script>
<script>
```

```
$(document).ready(function(){
  $("#mayoria_edad").click(function(){
    if ($("#mayoria_edad").prop("checked")){
      $("#formulariomayores").css("display", "block");
    }else{
      $("#formulariomayores").css("display", "none");
    }

  });
});
</script>

</head>

<body>
<form>
Nombre: <input type="text" name="nombre">
<br>
<input type="checkbox" name="mayor_edad" value="0" id="mayoria_edad"> Soy mayor de edad
<br>
<div id="formulariomayores" style="display: none;">
Dato para mayores de edad: <input type="text" name="mayores_edad">
</div>
</form>

</body>
</html>
```

Nótese que se utiliza el método prop en reemplazo del método attr en la siguiente línea $("#mayoria_edad").prop("checked") A partir de la versión 1.6 para algunos atributos se reemplazó el método attr() por el método prop(). Para mas información consultar la documentación oficial http://api.jquery.com/attr/

Ejemplo6.html

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Añadir y quitar clases CSS a elementos</title>
<script type="text/javascript" src="jquery-1.10.2.min.js"></script>
<style type="text/css">
        .clasecss{
          background-color: #ff8800;
          font-weight: bold;
        }
</style>
```

```
<script>
        $(document).ready(function(){
          $("a").mouseover(function(event){
                $("#capa").addClass("clasecss");
          });
          $("a").mouseout(function(event){
                $("#capa").removeClass("clasecss");
          });
        });
</script>

</head>

<body>
  <div id="capa">
  Esta capa es independiente y voy a añadir y eliminar clases css sobre ella
  </div>

  <br />
  <br />

  <a href="http://www.google.com">Añadir y quitar clase en la capa de arriba</a>

</body>
</html>
```

Ejemplo7.html

```
<html>
<head>
<title>Funciones CSS en jQuery</title>
<script type="text/javascript" src="jquery-1.10.2.min.js"></script>
<script type="application/x-javascript">
function dimensionCapa(capa){
capa = $(capa);
var dimensiones = "";
dimensiones += "Dimensiones internas: " + capa.innerWidth() + "x" +
capa.innerHeight();
dimensiones += "\nDimensiones externas: " + capa.outerWidth() + "x" +
capa.outerHeight();
alert(dimensiones);
}
function posicionCapa(capa){
capa = $(capa);
var posicion = "";
posicion += "Posición relativo al documento:\nLEFT: " + capa.offset().left + "\nTOP:" +
capa.offset().top;
```

```
posicion += "\n\nPosición si no tuviera margen:\nLEFT: " + capa.position().left + "\nTOP:" + capa.position().top;
alert(posicion);
}
$(document).ready(function(){
$("#botondimensiones").click(function(){
dimensionCapa("#capa1");
});
$("#botonposicion").click(function(){
posicionCapa("#capa1");
});
$("#botontamano").click(function(){
$("#capa1").css("width", 200);
});
$("#botonmargen").click(function(){
$("#capa1").css("margin", 20);
});
$("#botondimensionesc2").click(function(){
dimensionCapa("#capa2");
});
$("#botonposicionc2").click(function(){
posicionCapa("#capa2");
});

});

</script>

</head>
<body>
<h1>Funciones CSS en jQuery de dimensiones y posición</h1>
<p>Probando funciones de localización de elementos en la página...</p>
<div id="capa1" style="padding: 24px; background-color: #ffccdd; float: left; border: 2px dotted #666;">
<h2>capa1:</h2>
Voy a crear esta capa para ver lo que mide y donde está posicionada.
</div>
<br style="clear: both;">
<div style="margin: 10px;">
<button id="botondimensiones" type="button">Dimensiones de capa1</button>
<button id="botonposicion" type="button">Posicion de capa1</button>
<button id="botontamano" type="button">Cambiar tamaño capa1</button>
<button id="botonmargen" type="button">Cambiar margen capa1</button>
</div>

<div style="margin: 10px;">
<button id="botondimensionesc2" type="button">Dimensiones de capa2</button>
```

```
<button id="botonposicionc2" type="button">Posicion de capa2</button>
</div>

<br>
Desplaza la página hacia abajo para ver la capa2...
<br>
<br>
...
<br>
<div id="capa2" style="background-color:#ccc; border-bottom: 5px solid #999; margin-left: 10px;">
Esta capa está muy hacia abajo!!
</div>
</body>
</html>
```

Callback de funciones en JQuery

A menudo cuando hacemos aplicaciones enriquecidas del lado del cliente con jQuery nos vemos en la necesidad de encadenar varias llamadas a funciones, para que una se ejecute detrás de otra, creando un efecto más elaborado. En este artículo veremos lo sencillo que es realizar lo que en inglés se llama "callback", es decir una función que se ejecuta después de otra.

Apilar funciones, para que se ejecuten una detrás de otra, nos servirá para hacer muchas cosas. En nuestro día a día con jQuery iremos encontrando la utilidad, pero de momento para explicar un caso en el que nos resultará imprescindible, se me ocurre que deseemos hacer una secuencia de efectos y cambios dinámicos en un elemento.

Por ejemplo imaginemos que se desea ocultar una capa con un efecto de fundido (de opaco a transparente), luego moverla a otra posición y volverla a mostrar (ya en la nueva posición) con otro efecto de fundido (en este caso de transparente a opaco). En principio podríamos pensar en hacer un código como este:

```
$("#micapa").fadeOut(2000);
$("#micapa").css({top: 300, left:200});
$("#micapa").fadeIn(2000);
```

En este caso estamos alterando las propiedades de una capa con id="micapa". Primero llamamos a fadeOut() para ocultarla con un fundido, que durará 2 segundos (véase el parámetro 2000, que son los milisegundos que durará el efecto). Luego alteramos la posición de la capa, cambiando sus atributos CSS. Para acabar la volvemos a mostrar con un fundido de otros 2000 milisegundos.

Si lanzamos la ejecución de estas sentencias, tal como aparece en el código, será como si se ejecutasen todas a la vez. Como los fadeOut y fadeIn tardarán 2 segundos en

ejecutarse y los cambios de las propiedades CSS top y left son inmediatos, lo que ocurrirá será que primero veremos la capa moverse a la nueva posición y luego veremos los dos efectos de fundido.

Ahora que ya hemos visto uno de los casos en los que necesitaríamos ejecutar funciones en una pila, una después de otra, esperando a que termine completamente la ejecución de cualquier efecto o acción antes de comenzar con la siguiente. Vamos a ver cómo hacerlo con jQuery.
Simplemente tenemos que saber que todas las funciones o métodos de jQuery pueden recibir un parámetro adicional con el nombre de la función que se tiene que ejecutar después que termine el procesamiento de la primera. Esa segunda función que se ejecuta después de la primera es la que se conoce en inglés por callback. Un ejemplo sencillo para entenderlo.

```
miFuncion ("parametros de la función", funcionCallback);
```

En ese esquema de llamada a miFuncion(), se le están pasando dos parámetros. El primero sería un supuesto parámetro que necesitase miFuncion() y el segundo, que es le que nos interesa en este caso, el nombre de la función que se tiene que ejecutar después que acabe.

Con este código, primero se ejecuta miFuncion() y cuando acaba completamente, se ejecuta funcionCallback(). Pero atención que este ejemplo lo hemos simplificado para que se pueda entender fácilmente y esta sintaxis sólo valdrá si funcionCallback no recibe parámetros, porque no los podemos indicar con el nombre de la función.
Veamos entonces una forma de hacer este callback que funcione siempre:

```
miFuncion ("parametros de la funcion", function(){
funcionCallback();
});
```

Con este código, que funcionaría igual que el anterior, lo bueno es que sí podemos indicar los
parámetros que se necesiten para la llamada a funcionCallback().

Ahora que hemos aprendido toda la teoría, veamos un ejemplo práctico que solucionaría el problema comentado anteriormente sobre el procesamiento de diversos efectos y cambios en las propiedades de los objetos, para que se hagan siempre en la secuencia que deseamos. Se trata simplemente de aplicar las llamadas con callback que hemos antes.

Ejemplo8.html

```
$("#micapa").fadeOut(1000, function(){
$("#micapa").css({'top': 300, 'left':200});
$("#micapa").fadeIn(1000);
});
```

Como se puede ver, en la llamada a fadeOut() estamos pasando como parámetros el valor 1000, que son los milisegundos tiene que durar el efecto fade out (fundido hacia transparente), y luego la función callback, que se ejecutará después de que fadeOut() haya terminado.

Eventos

Los eventos son uno de los elementos más importantes en el desarrollo de aplicaciones web enriquecidas del lado del cliente, puesto que sirven para realizar acciones en la página a medida que el usuario realiza cosas con la página. Es decir, son la base para crear la interacción con el usuario, algo tan importante en las páginas que usan jQuery.

Ya hemos trabajado en algunos casos con eventos, ya que es complicado realizar ejemplos en páginas web que no tengan aunque sea una mínima interacción con el cliente. Casi siempre nos hemos limitado al evento click, pero hay muchos más.

A la vista de este código que trabaja con eventos podemos entender un poco mejor cómo funcionan en jQuery:

```
$(".mienlace").click(function(mievento){
  mievento.preventDefault();
  alert("Has hecho click Como he hecho preventDefault, no te llevaré al href");
});
```

1. El evento se define sobre todos los elementos de un objeto jQuery. En este ejemplo se define sobre el objeto jQuery obtenido al invocar el selector ".mienlace"), que devolvería todos los elementos que tienen el atributo class como "mienlace". Por tanto definiré un evento sobre un número variable de elementos de la página que concuerden con ese selector.
2. El tipo de evento se define a partir de una función click() o similares. Existen diferentes tipos de funciones que implementan cada uno de los eventos normales, como dblclick(), focus(), keydown(), etc.
3. Como parámetro en la función click() o similares tenemos que enviar una función, con el código que pretendemos ejecutar cuando se produzca el evento en cuestión.
4. La función que enviamos por parámetro con el código del evento, en este caso la función a ejecutar al hacer click, tiene a su vez otro parámetro que es el propio evento que estamos manejando. En el código anterior tenemos la variable "mievento", que es el evento que se está ejecutando y a través de esa variable tenemos acceso a varias propiedades y métodos para personalizar aun más nuestros eventos.
5. Como decimos, existen diversos tipos de propiedades y métodos sobre el evento que recibo por parámetro. En este caso utilizamos

mievento.preventDefault() para evitar el comportamiento por defecto de un enlace. Como sabemos, al pulsar un enlace el navegador nos lleva al href definido en la etiqueta A correspondiente, algo que evitamos al invocar a preventDefault() sobre nuestro evento.

Podemos ver el código completo del ejemplo anterior en el ejemlo9.html de la carpeta ejemplos.

Vamos a ver ahora un ejemplo con el evento dblclick. El evento doble-clic se produce cuando se realizan dos clic seguidos sobre un mismo elemento. Todos conocemos lo que es un doble clic, por lo que no necesitamos muchas más explicaciones, no obstante, tenemos que saber que cuando se produce un evento doble-clic al mismo tiempo se están produciendo eventos clic (uno por cada uno de los 2 clic del doble-clic). Para aclarar este asunto hemos hecho el siguiente ejemplo.

Tenemos una capa, en la que se puede hacer doble-clic, pero que también tiene definido un evento clic. Entonces, al hacer un doble clic podremos comprobar que se producen dos eventos clic y después un doble-clic.

Este es el código HTML con el que vamos a trabajar:

```
<div id="micapa" style="padding: 10px; background-color: #ffcc99; width: 150px; float: left;">Hazme dobleclick</div>

<div id="mensaje" style="padding: 10px; margin-left: 180px;">Aquí voy a colocar mensajes para que los leas...</div>
```

Para poder saber cuántos clics y dobles clic que se realizan, vamos a crear un par de variables Javascript para contarlos.

```
var numClics = 0;
var numDobleClics = 0;
```

Ahora veamos la programación del evento clic:

```
$("#micapa").click(function(e){
  numClics++;
  $("#mensaje").html("Clic " + numClics);
});
```

Con $("#micapa") obtenemos el objeto jQuery de la capa donde hay que hacer clic. Con el método click() sobre ese objeto jQuery creamos el evento clic y la función que pasamos como parámetro contiene el código a ejecutar cuando se hace clic. Se trata simplemente acumular 1 en la variable que cuenta los clics y luego se muestra un texto en la capa de los mensajes.

La programación del evento para el doble clic se puede ver a continuación:

```
$("#micapa").dblclick(function(e){
  numDobleClics++;
  $("#mensaje").html("Doble Clic " + numDobleClics);
});
```

Como se puede ver, es un código muy similar al anterior. Simplemente que se define el evento con el método dblclick(). En el código del evento acumulamos esta vez 1 en la variable que cuenta el número de dobles clic. Luego en el mensaje mostramos el número de doble-clic.

Con ello, al hacer clic o doble-clic se mostrará el mensaje para ver la cuenta de clics y dobles clic realizados y podremos comprobar que siempre se producen dos clics antes de cualquier doble clic.

Eso es todo, aunque para completar esta información, puedes encontrar a continuación el código completo de este ejemplo de uso de eventos en jQuery.

Ejemplo10.html

```
<html>
<head>
<title>Trabajando con eventos</title>
<script type="text/javascript" src="jquery-1.10.2.min.js"></script>
<script>
var numClics = 0;
var numDobleClics = 0;

$(document).ready(function(){

  $("#micapa").dblclick(function(e){
    numDobleClics++;
    $("#mensaje").html("Has hecho doble-clic<br>" + "Número de clics: " + numClics +
"<br>Número de doble clics: " + numDobleClics);
  });
  $("#micapa").click(function(e){
    numClics++;
    $("#mensaje").html("Has hecho doble-clic<br>" + "Número de clics: " + numClics +
"<br>Número de doble clics: " + numDobleClics);
  });
})
</script>

</head>
<body>
<h1>Trabajando con eventos en jQuery</h1>
  <div id="micapa" style="padding: 10px; background-color: #ffcc99; width: 150px;
float: left;">Hazme dobleclick</div>
  <div id="mensaje" style="padding: 10px; margin-left: 180px;">Aquí voy a colocar
```

```
mensajes para que los leas...</div>
</body>
</html>
```

Hata aquí hemos conocido los manejadores de eventos clic y doble-clic, pero hay muchos más. A continuación presentamos un listado de los tipos de manejadores de eventos disponibles en jQuery.

Eventos relacionados con el mouse

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

mouseup()
Para generar un evento cuando el usuario ha hecho clic y luego suelta un botón del mouse. El evento mouseup se produce sólo en el momento de soltar el botón.

mouseenter()
Este evento se produce al situar el mouse encima de un elemento de la página.

mouseleave()
Este se desata cuando el mouse sale de encima de un elemento de la página.

mousemove()
Evento que se produce al mover el mouse 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 mouse de la superficie de un elemento.

mouseover()

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

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.

Nota: a través del objeto evento, que reciben las funciones que indiquemos como parámetro de estos métodos, podemos saber qué tecla se está pulsando, aparte de otras muchas informaciones.

Eventos combinados teclado o mouse

Ahora mostramos varios eventos que pueden producirse tanto por el mouse como por el teclado, es decir, como resultado de una acción con el mouse o como resultado de presionar teclas en el teclado.

focusin()
Evento que se produce cuando el elemento gana el foco de la aplicación, que puede producirse al hacer clic sobre un elemento o al presionar el tabulador y situar el foco en ese elemento.

focusout()
Ocurre cuando el elemento pierde el foco de la aplicación, que puede ocurrir cuando el foco está en ese elemento y pulsamos el tabulador, o nos movemos a otro elemento

con el mouse.

focus()
Sirve para definir acciones cuando se produce el evento focus de Javascript, cuando el elemento gana el foco de la aplicación.

El objeto evento en JQuery

Vamos a hacer un inciso para dar una breve introducción al objeto evento en jQuery y ofrecer un ejemplo bastante práctico, para saber cuál es la posición del mouse al producirse un evento. Vamos a dar algunas nociones que deberemos conocer para poder acompañar los siguientes ejemplos sobre eventos.

Lo que ya hemos visto es que, al definir un evento con jQuery, tenemos que escribir una función con el código a ejecutar cuando se produzca el evento. Esa función recibe un parámetro, que es el **objeto evento**, que podemos utilizar dentro de la función del evento y que contiene diversas utilidades que pueden ser esenciales a la hora de codificar el evento.

En uno de los ejemplos anteriores hemos utilizado el método preventDefault() cuando definíamos un evento clic sobre un enlace. Cuando se hace clic sobre un enlace, el navegador se mueve a la dirección del href de ese enlace y con preventDefault() podemos evitar ese comportamiento por defecto de los enlaces. A continuación vamos a ver un ejemplo distinto de uso de las propiedades del objeto evento.

Vamos a ver como averiguar la posición del mouse al hacer click. En el objeto evento, entre otras muchas cosas, existen dos propiedades que nos informarán sobre la posición del mouse al producirse ese evento:

- pageX: que nos informa sobre el número de píxeles desde el lateral izquierdo de la página.
- pageY: con el número de píxeles desde la parte de arriba de la página.

Veamos el siguiente ejemplo:

```
$("#mielemento").click(function(e){
  $("#mielemento").html("X: " + e.pageX + " - Y: " + e.pageY)
});
```

Al hacer clic en el elemento con id="mielemento" se mostrarán las coordenadas X e Y del lugar de la página donde se hizo clic. Las coordenadas se mostrarán como texto en la propia capa sobre la que se ha hecho clic.

Este código se puede modificar fácilmente para que se muestre las coordenadas del mouse al hacer clic en la página, independientemente de donde se haga clic y no sólo

en un elemento en concreto.

```
$(document).click(function(e){
  alert("X: " + e.pageX + " - Y: " + e.pageY)
});
```

Como se puede ver, se ha indicado el evento "click" sobre el objeto document, que existe en Javascript y que hace referencia a todo el documento que se está visualizando.

El código completo de una página que define este evento y utiliza las mencionadas propiedades del objeto evento es el siguiente.

Ejemplo11.html

```
<html>
<head>
<title>Trabajando con el objeto evento</title>
<script type="text/javascript" src="jquery-1.10.2.min.js"></script>
<script>
$(document).ready(function(){
  $(document).click(function(e){
    alert("X: " + e.pageX + " - Y: " + e.pageY)
  });
})
</script>

</head>
<body>
<h1>Trabajando con el objeto evento</h1>
  Haz clic en cualquier parte de la página...

</body>
</html>
```

Nota: en los ejemplos anteriores hemos visto cómo calcular la posición del ratón al hacer clic. Sin embargo, nosotros podemos calcular la posición del ratón al producirse cualquier evento, ya que el objeto evento de jQuery está disponible para cualquier evento.

Por ejemplo, con este código mostramos la posición del mouse al moverlo por la página, mostrando las coordenadas en el texto de los titulares h1 que pueda haber en la página:

```
$(document).mousemove(function(e){
  $("h1").html("X: " + e.pageX + " - Y: " + e.pageY)
});
```

Eventos de mouse en JQuery mouseenter y mouseleave

A continuación veremos un ejemplo de página sencilla que utiliza eventos de mouse, para la construcción de un sistema de tip muy simple, es decir, tendremos una serie de áreas en la página, sobre las que situando el mouse por encima, aparecerá un mensaje explicativo que tenemos en otra capa.

Los eventos mouseenter y mouseleave, son los eventos más interesantes y útiles si queremos detectar el momento en el que entramos con el puntero del mouse sobre un elemento o salimos de su superficie.

Además, utilizaremos el objeto evento, que recibe la función con la que implementamos el manejador del evento, que tiene diversos datos útiles sobre el evento que se acaba de ejecutar. En este artículo mostraremos cómo averiguar la posición del mouse en el momento de producirse el evento, que podemos extraer con las propiedades pageX y pageY del objeto evento.

Anteriormente vimos como averiguar la posición del mouse al hacer click en un elemento. Así que ahora vamos a utilizar esos conocimientos para hacer un sencillo ejemplo de eventos donde crearemos un típico efecto de tip. Para realizar este efecto tendremos dos elementos, el primero será un elemento visible en todo momento y el segundo será un elemento oculto, el tip, que se mostrará sólo al pasar el ratón sobre el primer elemento.

Para realizar cosas cuando el ratón entra y sale de un elemento, utilizaremos los manejadores de eventos de jQuery mouseenter y mouseleave, que se producen al entrar con el mouse sobre un elemento y al salir del elemento respectivamente. Así pues, los eventos mouseenter y mouseleave los tendremos que crear sobre el elemento que permanece siempre visible, mostrando y ocultando la capa que contiene el tip.

Veamos antes que nada el HTML que tendremos, con el elemento visible y su tip.

```
<div id="elemento1" style="background-color: #ccccff; padding: 5px;">Pasa el ratón por encima de este "elemento1".</div>

<div class="tip" id="tip1">Esto es para explicar algo sobre el elemento1</div>
```

Además, al tip le hemos aplicado estilos por medio de CSS:

```
background-color: #ffcc99;
padding: 10px;
display: none;
position: absolute;
```

Los estilos importantes aquí son display: none; (para que el elemento esté oculto inicialmente) y position: absolute; (para que lo podamos posicionar libremente por la página y sin afectar a otros elementos).

Veamos ahora el código del evento mouseenter:

```
$("#elemento1").mouseenter(function(evento){
  $("#tip1").css("left", evento.pageX + 5);
  $("#tip1").css("top", evento.pageY + 5);
  $("#tip1").css("display", "block");
});
```

Simplemente cambiamos las propiedades de CSS "left" y "top" de la capa del tip, asignando valores a través de evento.pageX y evento.pageY, las propiedades del objeto evento que nos dan la posición del ratón. Con esto situamos la capa del tip en un lugar próximo a donde estaba el mouse.

Luego se cambia el atributo de CSS display de la capa del tip, al valor "block", que sirve para que ese elemento se vea en la página.

Ahora veamos el evento mouseleave, para realizar acciones cuando sacamos el mouse de encima de un elemento.

```
$("#elemento1").mouseleave(function(e){
  $("#tip1").css("display", "none");
});
```

Simplemente cambiamos la propiedad CSS display del tip, para el valor "none", que hace que esa capa desaparezca de la página.

Veamos el código completo de una página que implementa este mecanismo para producir tips en jQuery.

Ejemplo12.html

```
<html>
<head>
<title>Trabajando con eventos - Tip simple</title>
<style type="text/css">
  .tip{
    background-color: #ffcc99;
    padding: 10px;
    display: none;
    position: absolute;
  }
</style>
<script type="text/javascript" src="jquery-1.10.2.min.js"></script>
<script>
```

```
$(document).ready(function(){
  $("#elemento1").mouseenter(function(e){
    $("#tip1").css("left", e.pageX + 5);
    $("#tip1").css("top", e.pageY + 5);
    $("#tip1").css("display", "block");
  });
  $("#elemento1").mouseleave(function(e){
    $("#tip1").css("display", "none");
  });

  $("#elemento2").mouseenter(function(e){
    $("#tip2").css("left", e.pageX + 5);
    $("#tip2").css("top", e.pageY + 5);
    $("#tip2").css("display", "block");
  });
  $("#elemento2").mouseleave(function(e){
    $("#tip2").css("display", "none");
  });
})
</script>

</head>
<body>
<h1>Trabajando con eventos en jQuery</h1>

  <div id="elemento1" style="background-color: #ccccff; padding: 5px;">Pasa el ratón
por encima de este "elemento1".</div>
  <p>
    Este texto es para poner <a id="elemento2" href="#">otro elemento con tip</a>.
  </p>

  <div class="tip" id="tip1">Esto es para explicar algo sobre el elemento1</div>
  <div class="tip" id="tip2">Explico mejor este otro elemento con tip!!</div>
</body>
</html>
```

Eventos de teclado en JQuery

Vamos a hacer una práctica con los eventos de teclado, es decir, con la definición de acciones cuando el usuario presiona las teclas. La manera de trabajar con eventos de teclado no difiere mucho de la que ya hemos visto en otros eventos, pero con los eventos de teclado hay algo mas y es que cuando se produce el mismo, en el objeto evento de jQuery tenemos una propiedad que nos sirve para saber cuál es la tecla pulsada, para hacer cosas en nuestros scripts personalizadas en función de la tecla presionada por el usuario.

Los eventos de teclado, en principio, son tres, keydown, keypress y keyup. Realmente no actúan por separado, sino que se produce una combinación de éstos al ir presionando y soltando las teclas.

Por ejemplo si pulsamos y soltamos una tecla, primero se produce un evento keydown, al presionar la tecla, luego un keypress y por último un keyup al soltarla.

Si hacemos una pulsación prolongada de una tecla este esquema varía, pues se produce un keydown y luego un keypress. Mientras se mantiene pulsada la tecla en bucle se van produciendo eventos keydown y keypress, repetidas veces hasta que finalmente se suelta la tecla y se produce un keyup.

En el caso de las teclas CTRL, Mayúsculas o ALT, se producen múltiples keydown hasta que se suelta la tecla y se produce un keyup. Es decir, al pulsar una de estas teclas no se produce el evento keypress.

Vamos a aprender cuál es la secuencia con la que se producen los eventos de teclado, con un pequeño ejemplo práctico.

Se trata de hacer una función que detecte cualquier evento de teclado, muestre el tipo de evento que ha ocurrido y lo muestre en la página. Así podremos ver los eventos que se producen, sean cuales sean, y en qué orden.

Primero podríamos definir la función que va a procesar los eventos:

```
function operaEvento(evento){
  $("#loescrito").html($("#loescrito").html() + evento.type + ", ")
}
```

Esta función recibe el evento y escribe en una capa el tipo de evento, que se consigue con la propiedad type del objeto evento.

Ahora podríamos hacer que cualquier evento de teclado invoque esta función con el código:

```
$(document).keypress(operaEvento);
$(document).keydown(operaEvento);
$(document).keyup(operaEvento);
```

Como hemos asociado los eventos al objeto document de Javascript, estos eventos se pondrán en marcha cada vez que se pulse una tecla, independientemente de dónde esté el foco de la aplicación (o donde esté escribiendo el usuario).

El código completo del ejemplo es el siguiente:

Ejemplo13.html

```
<html>
```

```
<head>
<title>Trabajando con eventos de teclado en jQuery</title>
<script type="text/javascript" src="jquery-1.10.2.min.js"></script>
<script>
function operaEvento(evento){
  $("#loescrito").html($("#loescrito").html() + evento.type + ", ")
}
$(document).ready(function(){
  $(document).keypress(operaEvento);
  $(document).keydown(operaEvento);
  $(document).keyup(operaEvento);
})
</script>

</head>
<body>
<h1>Eventos de teclado en jQuery</h1>
  <div id="loescrito"></div>
</body>
</html>
```

Para averiguar que tecla fue pulsada utilizamos la propiedad which del objeto evento de jQuery, con ella podemos saber qué tecla ha sido pulsada cuando se produce el evento de teclado. Esta propiedad contiene un número entero con el código Unicode de la tecla pulsada. Haremos un ejemplo para explicarlo.

Tenemos un textarea y escribiendo algo en él, mostraremos la tecla pulsada en una capa, independiente del textarea. Este será el código HTML que necesitaremos para el ejemplo:

```
<form>
  <textarea cols=300 rows=2 id="mitexto">Escribe algo aquí!</textarea>
  <br>
  <b>Tecla pulsada:</b>
  <br>
  <div id="loescrito"></div>
</form>
```

Ahora definiremos con jQuery el evento keypress, para mostrar la tecla pulsada.

```
$("#mitexto").keypress(function(e){
  e.preventDefault();
  $("#loescrito").html(e.which + ": " + String.fromCharCode(e.which));
});
```

Con e.preventDefault(); hacemos que no se escriba nada en el textarea, osea, estamos inhibiendo el comportamiento habitual del evento, que es escribir las teclas en el textarea, que no tiene mucho que ver con nuestro ejemplo, pero que está bien para

ver cómo funciona.

Luego escribimos en la capa con id "loescrito" el código de Unicode de esa tecla y luego su conversión a un carácter normal, a través de la función estática de la clase String fromCharCode().

El código completo del ejercicio es el siguiente.

Ejemplo14.html

```
<html>
<head>
<title>Trabajando con eventos de teclado en jQuery</title>
<script type="text/javascript" src="jquery-1.10.2.min.js"></script>
<script>
$(document).ready(function(){
  $("#mitexto").keypress(function(e){
    e.preventDefault();
    $("#loescrito").html(e.which + ": " + String.fromCharCode(e.which))
  });
})
</script>

</head>
<body>
<h1>Eventos de teclado en jQuery</h1>
  <h2>Averiguar qué tecla se está pulsando</h2>
  <form>
    <textarea cols=300 rows=2 id="mitexto">Escribe algo aquí!</textarea>
    <br>
    <b>Tecla pulsada:</b>
    <br>
    <div id="loescrito"></div>
  </form>
</body>
</html>
```

Efectos

Ha llegado el momento de dedicarnos a mostrar las maneras con las que podemos crear efectos para adornar nuestras páginas y hacer que la experiencia de uso sea más atractiva. En adelante vamos a explicar el funcionamiento de algunos métodos sencillos y otros más complejos pero también más versátiles.

El método animate()

En el presente artículo vamos a comenzar a aprender cosas sobre el método animate(), uno de los más interesantes para hacer efectos en jQuery a partir de la modificación de propiedades CSS. Este método, como veremos, resulta bastante polivalente, pues con él podemos crear muchos tipos de animaciones, tantos como combinaciones de atributos CSS podemos tener. Sirve básicamente para ofrecer un listado de atributos CSS, con los nuevos valores a los que deseamos actualizarlos y jQuery se encargará de hacer esa modificación de manera que sea bastante suave.

Por ejemplo, tenemos un elemento con los atributos CSS width y height con valores "X e Y" y queremos animarlos para que esos atributos pasen a tener valores "Z y T" Con el método animate() podemos conseguir que esos atributos pasen de unos valores a otros sin cambios bruscos, y en lugar de ello lo hagan con una animación suavizada desde uno a otro valor.

Para invocar al método animate() tenemos que indicar una serie de parámetros, aunque sólo uno de ellos será obligatorio. La lista es la siguiente:

.animate(Propiedades, [Duración], [Función de animación], [Callback])

Propiedades: Este es el único parámetro que se debe pasar obligatoriamente y es para indicar qué atributos CSS queremos actualizar, con sus nuevos valores. Se tiene que indicar en notación de objeto, de manera similar a como se puede indicar en el método css() de jQuery y sólo permite el cambio de propiedades CSS que tengan valores numéricos. Por ejemplo, podríamos cambiar la anchura de un borde, pero no el tipo de borde (si queremos que sea sólido, con línea de puntos, etc.) porque no tiene valores numéricos. Generalmente, si no especificamos otra cosa los valores se entienden en píxeles. Los nuevos valores se pueden indicar de manera absoluta, o incluso de manera relativa, con un string del tipo "+=50", que indica que se debe aumentar en 50 ese atributo.

Duración: Sirve para indicar la duración de la animación, en un valor numérico en milisegundos, o en un valor de cadena de caracteres como "fast" o "slow".

Función de animación: Esta función sirve para indicar cómo se realizará la animación, si más suave al principio y rápida al final, o igual de rápida todo el tiempo. Es decir, la velocidad con la que se realizará el cambio de valores en diferentes puntos de dentro de la animación. En principio, los dos posibles valores son "swing" (por defecto) y "linear".

Callback: Ofrece la posibilidad de indicar una función a ejecutarse cuando se ha terminado totalmente de producir el efecto. Es decir, una función que se invoca cuando se ha llegado al valor final de los atributos CSS que se solicitaron cambiar.

Para acabar vamos a ver un ejemplo del método animate(), pero bastante simplificado.

En realidad sólo vamos a utilizar el primero de los parámetros, para indicar las propiedades CSS que deseamos animar.

Tendremos un titular en la página H1 con algunos atributos de estilos:

```
<h1 style="border-bottom: 1px solid #ff8800; font-size: 15pt;">Animacion jQuery</h1>
```

Nuestra animación hará que el borde del elemento pase a tener 20 píxeles de anchura y que el tamaño de la fuente suba para 25pt. Para ponerla en marcha utilizaríamos un código como el siguiente:

```
$("h1").animate({
  'border-bottom-width': "20px",
  'font-size': '25pt'
});
```

Como se puede ver, en notación de objeto indicamos dos atributos CSS y los dos valores a los que queremos animarlos. El primero de los valores, que no tiene unidades, es considerado como píxeles. El segundo valor, que se indica en puntos (pt), hará que jQuery utilice ese tipo de unidades en vez de los píxeles.

Además, podemos fijarnos que en este caso a animate() sólo le hemos pasado un parámetro, con la lista de las propiedades CSS a animar. Por tanto, dejamos a jQuery que utilice los valores por defecto de tiempo de animación y función de animación.

Veamos el código completo.

Ejemplo15.html

```
<html>
<head>
<title>Método animate jQuery</title>
<script type="text/javascript" src="jquery-1.10.2.min.js"></script>
<script>
$(document).ready(function(){
  $("#animar").click(function(e){
    e.preventDefault()
    $("h1").animate({
      'border-bottom-width': "20px",
      'font-size': '25pt'
    });
  });

  $("#restaurar").click(function(e){
    e.preventDefault()
    $("h1").css({
      'border-bottom-width': "1px",
      'font-size': '15pt'
```

```
    });
  });
})
</script>
</head>
<body>
<h1 style="border-bottom: 1px solid #ff8800; font-size: 15pt;">Animacion jQuery</h1>

Trabajando con el método animate:
<a href="#" id="animar">Animar</a>

<br>
<br>

Vuelvo a lo que había antes:
<a href="#" id="restaurar">Restaurar</a>

</body>
</html>
```

Fading en JQuery

Vamos a conocer otra manera de aplicar efectos a elementos de la página, a través de los métodos de fading de jQuery. Son métodos muy sencillos de aplicar y que sirven para crear efectos bastante atractivos, donde se produce un fundido a través del cambio de la propiedad opacity de CSS.

Recordemos que CSS tiene una propiedad para alterar la opacidad de los elementos. Todos los valores de Opacity se expresan con números de 0 al 1. Con un valor de cero haría que el elemento fuera totalmente transparente y opacity con un valor de 1 sería totalmente opaco.

Con los métodos de fading de jQuery se puede cambiar esa propiedad. Existen tres métodos para crear efectos de fundido, los siguientes:

Método fadeOut()
Este método hace que el elemento que lo recibe desaparezca de la página a través del cambio de su opacidad, haciendo una transición suavizada que acaba con el valor de opacity cero.

Método fadeIn()
El método fadeIn() hace que el elemento que lo recibe aparezca en la página a través del cambio de su opacidad, haciendo una transición suavizada que acaba con el valor de opacity 1. Este método sólo podremos observarlo si el elemento sobre el que lo invocamos era total o parcialmente transparente, porque si era opaco al hacer un

fadeIn() no se advertirá ningún cambio de opacidad.

Método fadeTo()

El tercer método para hacer efectos de fundidos es fadeTo() y es el más versátil de todos, puesto que permite hacer cualquier cambio de opacidad, a cualquier valor y desde cualquier otro valor. Este método recibe la duración deseada para el efecto, el valor de opacidad al que queremos llegar y una posible función callback.

Para ilustrar el modo en el que se pueden hacer efectos de fundido con el cambio de opacidad hemos hecho un ejemplo de página donde se pueden ver todos los métodos de fading en funcionamiento, con algunas variantes interesantes.

En el ejemplo vamos a tener una lista como esta:

```
<ul id="milista">
  <li id="e1">Elemento 1</li>
  <li id="e2">Segundo elemento</li>
  <li id="e3">Tercer LI</li>
</ul>
```

Como vemos, tanto la lista (etiqueta UL) como los elementos (etiquetas LI) tienen identificadores (atributos id) para poder referirnos a ellos desde jQuery.

Ahora veamos cómo hacer que la lista desaparezca con un fundido hacia transparente, a partir de una llamada a fadeOut().

```
$("#milista").fadeOut();
```

Como se puede ver, fadeOut() en principio no recibe ningún parámetro. Aunque luego veremos que le podemos pasar un parámetro con una función callback, con código a ejecutarse después de finalizado el efecto.

Este sería el código para que la lista vuelva a aparecer, a través de la restauración de su opacidad con una llamada a fadeIn().

```
$("#milista").fadeIn();
```

El método fadeTo es bastante más versátil, como ya se había adelantado. Para hacer un ejemplo interesante con este método tenemos que ver cómo se le pueden pasar distintos valores de opacidad y para ello hemos creado un campo select con distintos valores.

```
<select name="opacidad" id="selopacidad">
  <option value="0.2">20%</option>
  <option value="0.5">50%</option>
  <option value="0.8">80%</option>
  <option value="1">100%</option>
```

```
</select>
```

Como se puede ver, este SELECT tiene diferentes OPTION con algunos valores de opacidad. Los valores (atributos value de los OPTION) son números entre 0 y 1. Ahora vamos a mostrar el código de un evento que asociaremos a este campo SELECT, para ejecutar acciones cuando el usuario cambia el valor que aparece en él. Cuando el SELECT cambie, queremos actualizar el valor de opacity de los elementos H1 de la página.

```
$("#selopacidad").change(function(e){
  var opacidad_deseada = e.target.options[e.target.selectedIndex].value
  $("h1").fadeTo(1000,opacidad_deseada);
});
```

En este código estamos definiendo un evento "onchange" sobre el SELECT anterior. En la primera línea de la función se está extrayendo la opacidad deseada y para ello se accede a la propiedad target del objeto evento que se recibe en la función que enviamos al método change().

Nota: en el objeto evento, target es una referencia al objeto del DOM sobre el que se está codificando el evento. Es decir, en este ejemplo, e.target es una referencia al campo SELECT sobre el que estamos definiendo el evento. Con e.target.options[] tengo el array de options que hay dentro de ese SELECT. Con e.target.selectedIndex obtengo el índice del elemento seleccionado, para poder acceder a la opción seleccionada a través del array de options. Con e.target.options[e.target.selectedIndex].value estamos accediendo a la propiedad value del OPTION que se encontraba seleccionado. Así accedemos a la opacidad deseada que queríamos aplicar.

Una vez tenemos esa opacidad deseada, recogida del value del OPTION seleccionado, podemos ver la siguiente línea de código, en la que hacemos el fadeTo().

Veamos que fadeTo() recibe en principio dos métodos. El primero es la duración en milisegundos del ejemplo. El segundo es el valor de opacidad que queremos aplicar.

Los tres métodos que estamos viendo para hacer fading, como cualquiera de los existentes en jQuery, permiten el envío de un parámetro como función callback.

Con este código conseguimos que se ejecute un fadeIn() después de un fadeOut(), para conseguir un efecto de parpadeo, en el que primero se oculta el elemento y cuando desaparece se vuelve a mostrar restaurando su opacidad.

```
$("#milista").fadeOut(function(){
  $(this).fadeIn();
});
```

Como vemos, se está indicando una función callback y dentro de la misma, this es una referencia al objeto jQuery que recibió el anterior método. Osea, con $("#milista").fadeOut() se hace un efecto de fundido para que desaparezca el

elemento "#milista". Luego la función callback se ejecutará cuando ese elemento termine de desaparecer. Dentro de esa función callback se accede a $(this) para tener una referencia a "#milista" y sobre ese elemento invocamos al método fadeIn() para hacer que aparezca de nuevo la lista.

Ahora vamos a mostrar otro ejemplo de callback, en el que se encadenan varias funciones callback, que se ejecutarían una detrás de la otra.

```
var opacidad_deseada = $("#selopacidad").attr("value");
$("#e1").fadeTo(500, opacidad_deseada, function(){
  $("#e2").fadeTo(500, opacidad_deseada, function(){
    $("#e3").fadeTo(500, opacidad_deseada);
  });
});
```

En este código hacemos un efecto de fadeTo() sobre cada uno de los elemento de la lista. Para definir qué opacidad queremos aplicar a esos elementos utilizamos de nuevo el campo SELECT que habíamos visto anteriormente en este artículo. Pero en esta ocasión utilizamos una manera distinta de acceder al valor de opacidad que hay seleccionado, a través del método attr() de jQuery.

En el código anterior primero se ejecuta el cambio de opacidad en el primer elemento, luego en el segundo y por último en el tercero, siempre hacia la misma "opacidad_deseada" que se había recuperado en el SELECT.

A continuación podemos ver el código completo de trabajo con los métodos de fading disponibles en jQuery.

Ejemplo16.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd"
>
<html lang="es">
<head>
<title>Fading en jQuery</title>
  <script type="text/javascript" src="jquery-1.10.2.min.js"></script>
<script>
$(document).ready(function(){
  $("#ocultartoda").click(function(e){
    $("#milista").fadeOut();
  });
  $("#mostrartoda").click(function(e){
    $("#milista").fadeIn();
  });
  $("#ocultarmostrar").click(function(e){
    $("#milista").fadeOut(function(){
      $(this).fadeIn();
```

```
    });
  });
  $("#selopacidad").change(function(e){
    var opacidad_deseada = e.target.options[e.target.selectedIndex].value
    $("h1").fadeTo(1000,opacidad_deseada);
  });
  $("#pororden").click(function(e){
    var opacidad_deseada = $("#selopacidad").prop("value");
    $("#e1").fadeTo(500, opacidad_deseada, function(){
      $("#e2").fadeTo(500, opacidad_deseada, function(){
        $("#e3").fadeTo(500, opacidad_deseada);
      });
    });
  })
})
</script>
</head>
<body>
  <h1>Fading en jQuery</h1>
  <b>Mostrar y ocultar elementos de forma suavizada con fading</b>
  <p>
    <a href="#" id="ocultartoda">ocultar toda la lista</a> |
    <a href="#" id="mostrartoda">Mostrar toda la lista</a> |
    <a href="#" id="ocultarmostrar">Ocultar la lista y luego mostrarla</a>
  </p>
  <form name="f1">
    Cambia la opacidad del elemento H1 a: <select name="opacidad"
id="selopacidad">
      <option value="0.2">20%</option>
      <option value="0.5">50%</option>
      <option value="0.8">80%</option>
      <option value="1">100%</option>
    </select>
    <br>
    <a href="#" id="pororden">Cambiar la opacidad de los elementos de la lista por
orden</a>
  </form>

  <ul id="milista">
    <li id="e1">Elemento 1</li>
    <li id="e2">Segundo elemento</li>
    <li id="e3">Tercer LI</li>
  </ul>

</body>
</html>
```

Cola de efectos en JQuery

Todos estos métodos tratados anteriormente, y algunos más que no hemos visto todavía como sliceUp() o sliceDown(), que funcionan de manera similar a los ya vistos métodos fadeIn() o fadeOut(), sirven para realizar efectos variados en páginas web y son tan sencillos de usar como invocarlos sobre el objeto jQuery que contiene al elemento que deseamos animar. Ahora que ya sabemos hacer toda una gama de efectos simples, vamos a aprender a encadenar varios efectos a ejecutar uno detrás de otro.

Veremos que encadenar efectos es tan sencillo como llamar a todos los métodos de efecto que queremos realizar. Todos esos métodos se incluirán automáticamente en una cola y serán ejecutados uno detrás del otro, sin que tengamos que hacer nada por nuestra cuenta, aparte de la propia invocación de los métodos.

Para esto es necesario conocer el concepto de funciones de efectos. Las funciones de efectos son los métodos jQuery que realizan un cambio en los elementos de la página de manera suavizada, es decir, que alteran las propiedades de uno o varios elementos progresivamente, en una animación a lo largo de un tiempo.

Por poner un ejemplo, tenemos el método fadeOut(), que realiza un efecto de opacidad sobre uno o varios elementos, haciendo que éstos desaparezcan de la página con un fundido de opaco a transparente. El complementario método fadeIn() hace un efecto de fundido similar, pero de transparente a opaco. Como éstos, tenemos en jQuery numerosos métodos de efectos adicionales como animate(), sliceUp() y sliceDown(), etc. En la propia documentación del framework, en el apartado Effects de la referencia del API, podremos ver una lista completa de estas funciones de efectos.

Cuando invocamos varias funciones de efectos de las disponibles en jQuery, éstas se van introduciendo en una cola de efectos predeterminada, llamada "fx". Cada elemento de la página tiene su propia cola de efectos predeterminada y funciona de manera automática. Al invocar los efectos se van metiendo ellos mismos en la cola y se van ejecutando automáticamente, uno detrás de otro, con el orden en el que fueron invocados.

```
capa = $("#micapa");
capa.fadeOut();
capa.fadeIn();
capa.slideUp();
capa.slideDown();
```

Las funciones de efectos, una detrás de otra, se invocan en un instante, pero no se ejecutan todas a la vez, sino que se espera que acabe la anterior antes de comenzar la siguiente. Por suerte, jQuery hace todo por su cuenta para gestionar esta cola.

Como decimos, cada elemento de la página tiene su propia cola de efectos y, aunque

incluso podríamos crear otras colas de efectos para el mismo elemento, en la mayoría de los casos tendremos suficiente con la cola por defecto ya implementada .

Veamos un ejemplo Vamos lanzar varios efectos sobre una capa y veremos como ellos mismos se ejecutan en el orden como los hemos invocado.

Tendremos un elemento DIV, como este:

```
<div id="micapa">Esta capa que se va a animar, en un bucle infinito...</div>
```

Ahora podemos ver una función que realiza la invocación a varios efectos jQuery:

Ejemplo17.html

```
function colaEfectos(){
  capa = $("#micapa");
  capa.animate({
    "font-size": "1.5em"
  }, 2000);
  capa.hide(1000);
  capa.show(1000);
  capa.animate({
    "left": "350px",
    "top": "50px"
  },1500);
  capa.animate({
    "font-size": "0.75em"
  }, 2000);
  capa.animate({
    "left": "100px",
    "top": "20px"
  }, 1500, colaEfectos);
}
```

Habría que fijarse que la última de las funciones de efecto invocadas hace una llamada a esta misma función, por medio de un callback, por lo que, cuando terminen de ejecutarse todos los efectos, se volverá a invocar a la función y se producirá así un bucle infinito, donde se repetirá todo el tiempo la misma secuencia de efectos.

Y ahora podemos poner en marcha esta función cuando la página esté lista:

```
$(document).ready(function(){
  colaEfectos();
});
```

Con esto hemos hecho nuestro primer ejemplo de cola de efectos. Ha sido fácil, no? Pero claro que a partir de aquí la cosa se puede complicar todo lo que deseemos, o

necesitemos.

A continuación empezaremos a explicar el modo existente en jQuery para alterar las colas de efectos, para hacer cosas como detenerlas, analizarlas, cargar funciones de otros tipos para ejecutar en la cola, etc.

Método queue() para de alterar una cola de efectos

Vimos que crear una cola de efectos y como jQuery la gestiona de manera automática. Ahora queremos comenzar a mostrar cómo podemos trabajar nosotros mismos con estas colas de efectos y modificar su comportamiento.

Para ello vamos a ver el método más importante que tenemos que conocer para trabajar con las colas de efectos de jQuery: queue(). Como muchos otros métodos de este framework Javascript, queue() permite su invocación con distintos juegos de parámetros, por lo que, dependiendo de los valores que le pasemos a la función hará unas cosas u otras. Comenzaremos con el uso más simple y luego iremos complicando los ejercicios.

Si llamamos al método queue() sin parámetros o pasándole una cadena con el nombre de una cola, nos devolverá un array con cada una de las funciones que están encoladas en ese momento.

Si no indicamos parámetros a queue() estamos indicando que nos pase la lista de eventos encolados en la cola predeterminada. Si se indica un parámetro de tipo cadena, que sería el nombre de la cola a examinar, lo que nos devuelve es el array de funciones de la cola con nombre indicado en el parámetro.

Nota: El nombre de la cola predeterminada es "fx", por lo que llamar a la función:

elemento.queue("fx");

Tendría el mismo efecto que llamarla sin parámetros.

elemento.queue();

Veremos un ejemplo sencillo de esta posible invocación del método queue() y además, aparte vamos a ver que se pueden encolar funciones en la cola tantas veces como queramos, aunque la cola esté en marcha.

El efecto es que esas funciones encoladas posteriormente se quedarán al final de la cola y se ejecutarán cuando el resto de la cola se haya ejecutado.

Tenemos el siguiente HTML, que incluye varios elementos:

```
<button id="botonfade">Muestra y luego oculta con fadeIn y fadeOut</button>
<button id="botonslide">Muestra y luego oculta con slideUp slideDown</button>
<button id="botontamanocola">Muestra el número de funciones en cola ahora mismo</button>
<div id="mensaje">
  En estos momentos no hay funciones de efectos en la cola por defecto.
  <br>
  <span class="notar">Pulsa repetidas veces los botones de arriba para ir metiendo funciones en la cola</span>
</div>
<div id="micapa"></div>
```

Como se puede ver tenemos tres botones. Uno sirve para agregar funciones en la cola para hacer efectos fadeIn() y fadeOut(), el segundo para agregar a la cola funciones de efectos slideUp() y slideDown() y el tercero para mostrar la longitud de la cola en un momento dado.

Luego tenemos una capa para mostrar mensajes y otra con id="micapa" que será el DIV que vamos a animar con los efectos.

Así podremos definir el evento onclick del primer botón:

```
$("#botonfade").click(function(){
  capa = $("#micapa");
  capa.fadeOut(500);
  capa.fadeIn(500);
  muestraRestantesCola();
});
```

Así podemos definir el evento onclick del segundo:

```
$("#botonslide").click(function(){
  capa = $("#micapa");
  capa.slideUp(500);
  capa.slideDown(500);
  muestraRestantesCola();
});
```

Estos dos botones, como se puede ver, ejecutan efectos sobre "micapa" y el resultado es que, a medida que pulsamos los botones repetidas veces, los efectos se van encolando. Podemos pulsar tantas veces como queramos y se irán encolando más y más efectos en la cola predeterminada.

Al ejecutar estos eventos click, como última sentencia hay una llamada a la función muestraRestantesCola(), que veremos ahora mismo. Pero antes veamos el tercer botón, que sirve para mostrar el número de elementos de la cola predeterminada.

```
$("#botontamanocola").click(function(){
```

```
  muestraRestantesCola();
});
```

Como se ve, se llama a la función muestraRestantesCola(), que simplemente accede a la cola para saber el número de funciones de efectos encoladas en un momento dado. Su código es el siguiente:

```
function muestraRestantesCola(){
  var numFuncionesEnCola = $("#micapa").queue().length;
  $("#mensaje").text("En el momento de hacer el último clic en los botones hay " +
numFuncionesEnCola + " funciones de efectos en cola");
}
```

En la primera sentencia se accede a la cola predeterminada del elemento con id="micapa", lo que nos devuelve un array, al que luego se accede a su propiedad "length" para saber el número de elementos que contiene. Con esto averiguamos el número de funciones encoladas en un momento dado. Luego se muestra ese número en la capa con id="mensaje".

Podemos ver el código completo de este ejercicio.

Ejemplo18.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd"
>
<html lang="en">
<head>
<title>Cola de efectos por defecto en jQuery</title>
<script src="../jquery-1.4.2.min.js" type="text/javascript"></script>
<style type="text/css">
body{
  font-size: 0.75em;
  font-family: tahoma, verdana, sans-serif;
}
.notar{
  color: #339;
}
#mensaje{
  margin: 20px 5px;
}
#micapa{
  left: 200px;
  top: 150px;
  position: absolute;
  width: 50px;
  height: 50px;
  background-color: #3d3;
```

```
}
</style>
<script languague="javascript">
function muestraRestantesCola(){
  var numFuncionesEnCola = $("#micapa").queue().length;
  $("#mensaje").text("En el momento de hacer el último clic en los botones hay " +
numFuncionesEnCola + " funciones de efectos en cola");
}
$(document).ready(function(){
  $("#botonfade").click(function(){
    capa = $("#micapa");
    capa.fadeOut(500);
    capa.fadeIn(500);
    muestraRestantesCola();
  });
  $("#botonslide").click(function(){
    capa = $("#micapa");
    capa.slideUp(500);
    capa.slideDown(500);
    muestraRestantesCola();
  });
  $("#botontamanocola").click(function(){
    muestraRestantesCola();
  });
});
</script>

</head>
<body>
  <button id="botonfade">Muestra y luego oculta con fadeIn y fadeOut</button>
  <button id="botonslide">Muestra y luego oculta con slideUp slideDown</button>
  <button id="botontamanocola">Muestra el número de funciones en cola ahora
mismo</button>
  <div id="mensaje">
    En estos momentos no hay funciones de efectos en la cola por defecto.
    <br>
    <span class="notar">Pulsa repetidas veces los botones de arriba para ir metiendo
funciones en la cola</span>
  </div>
<div id="micapa"></div>
</body>
</html>
```

A continuación aprenderemos a encolar funciones que no son las de efectos de jQuery, de modo que podamos meter nuestras propias funciones en la cola, con cualquier tipo de instrucción.

Meter cualquier función que no sea un efecto en una cola

¿Qué pasa si queremos encolar otro tipo de función Javascript o jQuery que no sea un efecto? Como sabemos hasta ahora, las funciones de efectos se encolan ellas mismas sin que tengamos que hacer nada, pero si se trata de otro tipo de función la situación cambia un poco, pues tendremos que encolarla nosotros mismos explícitamente y para ello tendremos que utilizar el método queue() de jQuery.

El método queue() funciona de maneras distintas dependiendo de los parámetros que le enviemos. E

El juego de parámetros con el que tenemos que llamar al método queue() para encolar cualquier tipo de función es el siguiente:

.queue([nombreCola],callback(continua))

- Primer parámetro nombreCola, que es opcional, se indica el nombre de la cola donde encolar una función. Si no se indica nada, o si se indica el nombre de la cola predeterminada "fx", se encola esa función en la cola por defecto que gestiona jQuery por nosotros. Si se indica cualquier valor distinto de "fx" se encolará en esa cola que estemos indicando.
- El segundo parámetro es la función que se desea encolar. Al encolarla se coloca como última de las funciones a ejecutar de la cola, por tanto, se tendrán que ejecutar todas las funciones encoladas anteriormente antes de llegar a ésta que estamos introduciendo.

A continuación podemos ver un código de ejemplo en el que encolamos una función, que no es de efectos, en la cola de efectos predeterminada.

```
capa = $("#micapa");
capa.queue(function(){
  $(this).css({
    "border": "3px solid #339",
  });
  //cualquier otro código jQuery....
  //llamamos al siguiente paso de la cola
  $(this).dequeue();
});
```

Como se puede ver, se llama a queue() indicando la función que deseamos encolar. Ésta tiene la llamada a un método, css(), que no es un método de efecto animado y que no se encolaba de manera predeterminada como sí lo hacían las funciones de efectos. Además, luego podríamos tener un número indeterminado de instrucciones jQuery, tantas como se desee.

Lo que es importante es que, al final del código de esta función, se debe invocar explícitamente al siguiente paso de la cola. Esto lo hacemos con una llamada al

método dequeue() que aun no habíamos visto. Si no llamamos a este método, ocurriría que la cola se detendría y no continuaría la ejecución de otras funciones encoladas en el caso que las hubiera.

Nota: El método dequeue() puede recibir un parámetro que es el nombre de la cola que se debe continuar ejecutándose. Si no indicamos ninguna cola o indicamos el valor "fx", la cola que seguirá procesándose es la cola por defecto.

A partir de jQuery 1.4 existe otra posibilidad de trabajo con las colas y es que a partir de esa versión del framework, la función que estamos encolando recibe un parámetro (que nosotros hemos llamado "continua") que es la función siguiente de la cola. Este parámetro nos serviría para continuar la cola sin tener que ejecutar el método dequeue(). Podemos ver un ejemplo a continuación.

```
capa.queue(function(continua){
  alert("Hola, esto es un código cualquiera");
  //el parámetro continua es una función para ir al siguiente paso de la cola
  continua();
});
```

Ahora podemos ver un ejemplo completo en el que encolamos varios tipos de funciones. Algunas son funciones de efectos, que no necesitamos que hacer nada para que se encolen y otras son funciones normales, que tenemos que encolar explícitamente.

Tenemos este código HTML:

Ejemplo19.html

```
<button id="botoncomenzar">Hacer una cola de ejecución con funciones que no son efectos</button>
<div id="micapa"></div>
```

Como se puede ver, hay un botón y una capa. La capa nos servirá para animarla y el botón para comenzar la animación en el momento que lo pulsemos. Veamos entonces el código del evento click que asociaremos a ese botón y que encolará varias funciones, unas de efectos y otras funciones normales.

```
$("#botoncomenzar").click(function(){
  capa = $("#micapa");
  //encolo directamente funciones que son efectos
  capa.animate({"width": "80px"}, 1000);
  //para encolar otras funciones utilizo queue()
  capa.queue(function(){
    $(this).css({
      "border": "3px solid #339",
    });
```

```
    $("#botoncomenzar").text("Acabo de ponerle el borde... ");
    $(this).dequeue();
  });
  capa.animate({"height": "200px"}, 2000);
  capa.queue(function(continua){
    $(this).css({
      "border": "0px"
    });
    $("#botoncomenzar").text("Quitado el borde... ");
    //el parámetro continua es una función que lleva al siguiente paso de la cola (jpara
Query 1.4)
    continua();
  });
  capa.animate({
    "height": "50px",
    "width": "400px"
  }, 1000);
});
```

Parar la ejecución de una cola

Vamos a aprender a detener colas de efectos, algo que más tarde o más temprano vamos a necesitar cuando hagamos scripts interactivos que incluyan funciones de efectos.

Comenzaremos, explicando el último uso que nos queda por revisar del método queue().

.queue([nombreCola], arrayFunciones)

En éste uso del método queue() enviamos dos parámetros:

- El primer parámetro opcional con el nombre de la cola (por defecto si no se indica nada se entiende la cola "fx" que es la predeterminada.
- El segundo parámetro es un array de funciones que se cargarán en la cola. Las funciones que pudiera haber en la cola se descartarán y se encolarán las nuevas funciones que se indican en el Array, cuya ejecución también será secuencialmente, tal como estén ordenadas en el array.

Un ejemplo de código donde hacemos este uso del método queue() es el que se puede ver a continuación:

```
$("#elemento").queue([function(){
  $(this).hide("slow");
  $(this).show("slow");
```

```
}]);
```

Como se puede comprobar, invocamos queue() indicando sólo un parámetro de tipo array, con lo cual estaremos modificando la cola predeterminada para asignarle las funciones del array. En este caso en el array hay una sola función que se ejecutará como siguiente paso de la cola.

El método stop() sirve para detener la animación actual en la cola de efectos, pero atención, con stop() en principio sólo detenemos el efecto actual de la cola de efectos, por lo que los siguientes pasos de la cola, si es que había, seguirán ejecutándose. Podemos enviarle dos parámetros, ambos booleanos y opcionales:

- El parámetro limpiarCola sirve para eliminar todas las funciones que pudieran haber en la cola de efectos todavía por ejecutar. Si indicamos true, esas funciones se retirarían de la cola de efectos y por tanto no se ejecutarían más. Si no indicamos nada o se indica un false, ocurrirá que sólo se detenga el efecto o función actual y automáticamente se continuará con la siguiente función que pudiera haber encolada.
- El segundo parámetro, saltarAlFinal, sirve para que se detenga el efecto de animación actual, pero que se coloque directamente el valor final al que tendía ese efecto. Por defecto, este parámetro toma el valor false, que indica que la animación se para y se queda donde se había detenido. Por ejemplo en el caso false, si estamos ocultando una capa por medio de un efecto animado y hacemos un stop() la capa quedaría ocultada a medias. Sin embargo, si indicamos true en este parámetro, al hacer stop(), aunque no se haya terminado la animación para ocultar la capa, ésta se ocultaría del todo repentinamente.

Podemos ver varios ejemplos:

```
$("#elemento").stop();
```

Esto terminaría con el efecto que se está ejecutando en la cola, pero continuaría con los siguientes efectos que pudiera haber encolados.

```
$("#elemento").stop(true);
```

Terminaría con el efecto que se esté realizando y limpiaría la cola, con lo que no se ejecutarían otras funciones que pudiera haber.

```
$("#elemento").stop(false, true);
```

Terminaría el efecto actual pero saltaría en la animación para mostrar directamente el estado al que se llegaría si el efecto huviese continuado hasta el final. Luego continuaría automáticamente con la siguiente función de la cola.

Ahora vamos a realizar un ejercicio completo con los métodos jQuery que acabamos de explicar. Se trata de hacer una animación, compuesta por varias funciones que se

insertarán en la cola y varias pruebas de uso del método stop().

Comenzamos mostrando el código HTML de los elementos de este ejemplo:

```
<button id="botoncomenzar">Comenzar animación</button>
<br>
<button id="botonparar" class="botondetener">Pasar a la siguiente etapa de la animación</button>
<br>
<button id="botonpararllegar" class="botondetener">Pasar a la siguiente etapa, pero llegar hasta el final de donde se planeaba la animación</button>
<br>
<button id="botonparartodo" class="botondetener">Parar todo!</button>
<div id="micapa">Hola a todos!!!</div>
```

Como se puede ver, hay 4 botones. El primero servirá para poner en marcha la animación y los otros 3 para parar las funciones de la cola de efectos, de diversos modos, para que podamos practicar. Luego tenemos un elemento DIV con la capa que pretendemos animar.

El primero de los tres botones siempre permanece visible, pero los otros en principio no se muestran en la página, gracias a este CSS:

```
button.botondetener{
  display: none;
}
```

Ahora vamos a ver cada una de las funciones que cargaremos como eventos click, a ejecutar en cada uno de los botones. Comenzaremos con el botón que produce la animación en una cola de efectos.

```
$("#botoncomenzar").click(function(){
  capa = $("#micapa");
  capa.queue(function(continua){
    $("button.botondetener").show(500);
    continua();
  });
  //2 animaciones que tardan mucho
  capa.animate({"left": "0px"}, 5000);
  capa.animate({"left": "200px"}, 5000);
  capa.queue(function(continua){
    $("button.botondetener").hide(500);
    continua();
  });
});
```

Como vemos, tenemos una serie de funciones que se van a encolar. Como primer paso de la animación se hace una instrucción para que se muestren los botones que no

estaban visibles.

Luego hacemos dos efectos animados, a los que les ponemos una duración de 5 segundos cada uno, lo suficiente para que nos dé tiempo a detener la animación antes que estos efectos se lleguen a completar.

Ahora veamos los distintos eventos click para los botones que pararán la animación, con varios comportamientos ligeramente distintos.

```
$("#botonparar").click(function(){
  $("#micapa").stop();
});
```

Con esta función conseguiremos que se pare el paso actual de la animación, pero se continuará con el siguiente paso que haya en la cola de efectos.

```
$("#botonpararllegar").click(function(){
  $("#micapa").stop(false, true);
});
```

Con esta función hacemos que se detenga el paso actual de la animación, pero llevamos el elemento al lugar donde hubiera llegado si la animación hubiese continuado hasta el final. Luego continúa con los siguientes efectos encolados.

```
$("#botonparartodo").click(function(){
  $("#micapa").queue([]);
  $("#micapa").stop();

  //Esto mismo podría haberse hecho también así:
  //$("#micapa").stop(true);

  alert("Lo he parado todo!, ni se ocultarán los botones de parar. Pasos encolados: " +
$("#micapa").queue().length)
});
```

Esta función es la más compleja de todas, veamos por qué.

```
$("#micapa").queue([]);
```

Con esto estoy cambiando la cola por defecto del elemento con id="micapa". Y estamos asignando una cola de efectos vacía, porque el array enviado como parámetro no tiene elementos. Con eso consigo quitar todas las funciones de la cola, pero hay que tener en cuenta que alguno de los efectos puede estar ejecutándose todavía y lo puedo parar con la sentencia:

```
$("#micapa").stop();
```

Con eso consigo que la animación se detenga tal como estuviera en ese mismo

instante. Esto, si lo preferimos, se podría haber conseguido con una simple llamada al método stop():

```
$("#micapa").stop(true);
```

Para acabar, lanzamos un mensaje al usuario, en una simple caja de alert(), para indicar los pasos que quedaron encolados en ese momento, que son cero, dado que hemos retirado todas las funciones de la cola.

El código completo del ejemplo se encuentra en el ejemplo20.html.

Método delay() para retrasar la ejecución de efectos de la cola

El método delay() es necesario aprender a utilizar para generar una pausa entre la ejecución de funciones que se encuentran en la cola de efectos.

Es tan sencillo como invocarlo indicando como parámetro el número de milisegundos que deseas que se espere entre una y otra llamada a las funciones encoladas en la cola de efectos, pero aunque sea bastante obvio, quizás estará bien ofrecer algunas notas sobre su funcionamiento.

Como sabemos, las funciones de la cola de efectos se ejecutan una detrás de la otra, sin que transcurra ningún tiempo entre los distintos efectos encolados. Es decir, en el instante que un efecto termina, comienza el siguiente efecto de la cola sin más demora. Pero esto es algo que podemos cambiar si usamos delay().

Nota: Ahora bien, cabe decir que delay() no reemplaza la función nativa setTimeout() de Javascript. el método delay() sólo sirve para generar una pausa entre efectos de la cola de efectos, pero no para producir tiempos de espera en general, que tendrán que realizarse como debemos de saber, con la función nativa setTimeout().

El método delay recibe dos parámetros, que explicamos a continuación:

.delay(duaracion, [nombreCola])

- El parámetro duración sirve para indicar los milisegundos de espera que tienen que producirse entre efecto y efecto.
- El parámetro nombreCola está para indicar en qué cola de efectos se desea realizar ese retardo. Este segundo parámetro es opcional y si no lo indicamos se realizará la espera en la cola de efectos predeterminada.

Veamos el ejemplo siguiente:

```
capa = $("#micapa");
capa.slideUp(1000);
capa.delay(2000)
```

```
capa.slideDown(1000);
```

En este caso estamos encolando dos efectos, con un retardo entre medias de 2 segundos. En total habremos encolado tres funciones, la primera un efecto slideUp(), la segunda un retardo de 2000 milisegundos y la tercera un efecto slideDown().

Esta carga de las tres funciones se podría resumir, concatenando llamadas a los métodos de la siguiente manera:

```
capa.slideUp(1000).delay(2000).slideDown(1000);
```

A continuación podemos ver una página de ejemplo completa en la que hacemos varios efectos y aplicamos varios retardos entre ellos.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd"
>
<html lang="en">
<head>
<title>Cola con delay()</title>
<script src="../jquery-1.4.2.min.js" type="text/javascript"></script>
<style type="text/css">
#micapa{
  left: 20px;
  top: 20px;
  position: absolute;
  font-size: 0.75em;
  font-family: tahoma, verdana, sans-serif;
  width: 740px;
  background-color: #ddf;
  padding: 10px;
  border: 1px solid #bbb;
}
</style>
<script languague="javascript">
function colaEfectos(){
  capa = $("#micapa");
  capa.slideUp(1000);
  capa.delay(2000)
  capa.slideDown(1000);

  capa.fadeTo(1500, 0.3).delay(3000).fadeTo(500, 1);

  capa.delay(500);
  capa.animate({
    "font-size": "+=0.5em"
  }, 1000, colaEfectos);
  //alert (capa.queue().length)
```

```
}
$(document).ready(function(){
  colaEfectos();
});
</script>

</head>
<body>
<div id="micapa">Vamos a ejecutar varios métodos para hacer una cola de efectos,
pero vamos a ponerles un retardo entre unos y otros.</div>
</body>
</html>
```