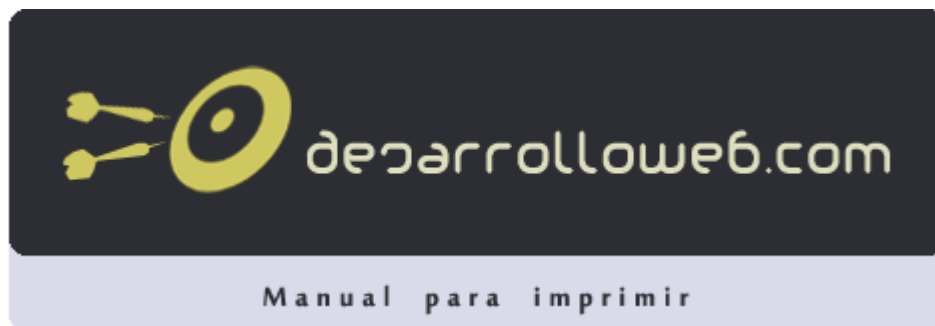


Manual de jQuery

El manual de jQuery pretende explicar el popular framework Javascript jQuery, con el que podremos hacer aplicaciones web avanzadas del lado del cliente.



Autores del manual

Este manual ha sido realizado por los siguientes colaboradores de DesarrolloWeb.com:

Miguel Angel Alvarez
Director de DesarrolloWeb.com
<http://www.desarrolloweb.com>
(43 capítulos)

Parte 1:

Introducción a jQuery

Comenzamos por los capítulos más básicos sobre jQuery, que sirven para introducirnos en el desarrollo de una manera sencilla. Hablaremos sobre la metodología de trabajo con el framework Javascript de manera general.

1.1.- Introducción a jQuery

Qué es jQuery, para qué sirve y qué ventajas tiene el utilizar este framework Javascript.

Bienvenidos al [manual sobre jQuery](#) que vamos a publicar en DesarrolloWeb.com, con el que pretendemos clarificar a los usuarios el método de trabajo y programación de aplicaciones del lado del cliente, compatibles con todos los navegadores más comunes.

1.1.1.- 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.

Nota: si no sabes lo que es Javascript seguramente no te interesará este artículo, pero puedes aprenderlo también en DesarrolloWeb.com: [Qué es Javascript](#)

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.

1.1.2.- Ventajas de jQuery con respecto a otras alternativas

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

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

1.1.3.- jQuery, es para mí?

Si estás interesado en enriquecer tu página web con componentes de la llamada Web 2.0, como efectos dinámicos, Ajax, interacción, interfaces de usuario avanzadas, etc., jQuery es una herramienta imprescindible para desarrollar todas estas cosas sin tener que complicarte con los niveles más bajos del desarrollo, ya que muchas funcionalidades ya están implementadas, o bien las librerías del framework te permitirán realizar la programación mucho más rápida y libre de errores.

Ahora bien, todas estas mejoras de la web 2.0, que en un principio puede ser muy atractivas, también tienen un coste en tiempo de desarrollo de los proyectos. Sin un framework como jQuery, el tiempo de creación y depuración de todos esos componentes dinámicos sería mucho mayor, pero aun así nadie dice que todo sea instalar el sistema y empezar correr. Sin embargo, lo más complicado de jQuery es aprender a usarlo, igual que pasa con cualquier otro framework Javascript. Requerirá del desarrollador habilidades avanzadas de programación, así como el conocimiento, al menos básico, de la [programación orientada a objetos](#). Una vez aprendido las ventajas de utilizarlo compensarán más que de sobra el esfuerzo. Esperamos que con este [Manual de jQuery](#), que vamos a publicar en DesarrolloWeb.com puedas aprender lo necesario para desarrollar tus propios componentes dinámicos en Javascript con los que enriquecer tus aplicaciones.

Por otra parte publicaremos artículos con ejemplos prácticos de JQuery que iremos colocando en nuestro [taller de JQuery](#), para aquellos que ya tengan conocimientos en esta materia.

Además tenemos un [Videotutorial de jQuery](#) con una colección de vídeos para aprender paso a paso el popular framework Javascript.

Podemos conocer jQuery accediendo a la página de inicio del framework Javascript: <http://jquery.com/>

Artículo por Miguel Angel Alvarez

1.2.- Demo muy simple de uso de jQuery

Vamos a hacer nuestro primer script en jQuery, con unas funcionalidades muy simples, para que sirva de demo de uso de este framework.

Con objetivo de que los lectores puedan hacerse una rápida idea de las posibilidades de jQuery, escribiendo unas brevísimas

líneas de código Javascript, vamos a publicar un par de ejemplos bien simples que nos ilustren, pero sin complicarnos demasiado. Nos servirán para la introducción a jQuery que estamos publicando en el [Manual de jQuery](#).

La idea de este artículo no es explicar las funcionalidades que vamos a demostrar, sino ver el poco código que hemos tenido que escribir para realizar unos scripts con dinamismos sencillos. Quizás los scripts en si no digan mucho a un lector poco experimentado, pero los que ya han tenido contacto con los pormenores que hay que seguir para hacer estos efectos, de manera que sean compatibles con todos los navegadores, sabrán que jQuery nos ha simplificado mucho nuestra tarea.

Así pues, no te preocupes demasiado con los detalles de estos códigos, que los explicaremos en DesarrolloWeb.com más adelante con detalle.

1.2.1.- Demo 1 de jQuery

Para empezar vamos a ver este ejemplo, donde tenemos dos botones y un texto. Al pulsar un botón, cambiaremos el texto y al pulsar el otro pondremos otro texto distinto.

Podemos ver el [ejemplo en marcha en una página aparte](#).

En este ejemplo tenemos una capa que tiene este código

```
<div id="capa" style="padding: 10px; background-color: #ff8800">Haz clic en un botón</div>
```

Luego tenemos dos botones con estos códigos:

```
<input type="button" value="Botón A" onclick="$('#capa').html('Has hecho clic en el botón <b>A</b>')">
<input type="button" value="Botón B" onclick="$('#capa').html('Recibido un clic en el botón <b>B</b>')">
```

Como se puede ver, en los botones hay definidos un par de eventos onclick (uno en cada uno) que ejecutan una instrucción Javascript cuando se hace clic sobre ellos. La instrucción está en Javascript, pero hace uso de algunas herramientas disponibles en jQuery para trabajo con los elementos de la página. En este caso, por explicarlo rápidamente, se hace una selección del elemento DIV de la capa y luego se ejecuta un método sobre él para cambiar el contenido HTML del elemento.

1.2.2.- Demo 2 de jQuery

En este otro ejemplo vamos a ver algo un poquito más complejo. Bueno, realmente no tiene mucha mayor complejidad, pero estamos utilizando jQuery de una manera un poco distinta, que requiere algo más de código por nuestra parte. Ahora vamos a tener dos capas en nuestra página. Una capa estará siempre visible y otra capa aparecerá inicialmente oculta y la vamos a mostrar u ocultar dependiendo de si el usuario coloca el ratón encima de la capa que está siempre visible.

Para hacerse una idea exacta de nuestro ejemplo puedes [verlo en una ventana aparte](#).

En este segundo ejemplo tenemos este código HTML, con las dos capas.

```
<div id="capa" style="padding: 10px; background-color: #ff8800;">Pon el ratón encima de esta capa</div>
<br>
<div id="mensaje" style="display: none;">Has puesto el ratón encima!! <br>(Ahora quítalo de la capa)</div>
```

Ahora vamos a tener un código Javascript con jQuery que defina los eventos del usuario, para cuando sitúa el ratón dentro o fuera de la primera capa.

```
$("#capa").mouseenter(function(evento){
    $("#mensaje").css("display", "block");
});
$("#capa").mouseleave(function(evento){
    $("#mensaje").css("display", "none");
});
```

De esta sencilla manera, hemos creado dos eventos en el DIV con id="capa". Además, hemos definido el código de los eventos por medio de funciones, que se encargarán de mostrar o ocultar la segunda capa con id="mensaje".

```
$("#mensaje").css("display", "block");
```

Esto nos selecciona la capa con id "mensaje" y con el método `css()` indicamos que queremos cambiar el atributo "display" al valor "block" de ese elemento.

```
$("#mensaje").css("display", "none");
```

Esta otra línea muy similar, simplemente cambia el "display" a "none" para ocultar el elemento.

Esperamos que estos dos ejemplos te hayan servido para ver rápidamente algunas cosas que se pueden hacer con jQuery con muy poco esfuerzo y que funcionan en todos los navegadores.

Artículo por *Miguel Angel Alvarez*

1.3.- Pasos para utilizar jQuery en tu página web

Una descripción de la puesta en marcha de tu primer script jQuery, en tu propia página web, paso a paso.

En este artículo te vamos a explicar cómo comenzar a utilizar jQuery en tus páginas web, paso a paso, para que puedas hacer tu primer script jQuery y así comprender los fundamentos de uso de este framework Javascript. En este punto estaría bien que sepas lo que es jQuery, lo que ha sido explicado ya en el [Manual de jQuery](#) que venimos publicando en DesarrolloWeb.com.

La idea es que puedas ir haciendo tú mismo los pasos que vamos a relatar, para que compruebes tú mismo lo sencillo que es comenzar a utilizar jQuery. Este artículo sigue el esquema con el que los propios creadores de jQuery enseñan a utilizar su producto, así que está directamente inspirado en la [documentación de jQuery](#).

Vídeo: Todo el proceso de creación de un primer ejemplo con jQuery relatado en este artículo lo hemos grabado en el [videotutorial comenzar a programar con jQuery](#).

1.3.1.- Descarga la última versión del framework

Accede a la página de [jQuery](#) para descargar la última versión del framework. En el momento de escribir este artículo la release actual es la 1.3.2, y con la que hemos realizado estos ejemplos. Sin embargo, puede que haya publicado una nueva versión que mejore la actual.

Dan 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 DEVELOPMENT, 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.

Verás que la descarga es un archivo js que contiene el código completo del framework. Coloca el archivo en una carpeta en tu ordenador para hacer las pruebas.

1.3.2.- Crea una página HTML simple

Ahora, en el mismo directorio donde has colocado el archivo js, coloca un archivo html con el siguiente código.

```
<html>
  <head>
    <script src="jquery-1.3.2.min.js" type="text/javascript"></script>
    <script>

    </script>
  </head>
```

```
<body>
<a href="http://www.desarrolloweb.com/">DesarrolloWeb.com</a>
</body>
</html>
```

Como podrás ver, es una página bien simple, en la que tenemos una llamada a un script externo llamado jquery-1.3.2.min.js. Este es el código de jQuery que hemos descargado de la página del framework. Si has descargado una versión distinta, quizás el archivo se llame de otra manera, así que es posible que tengas que editar ese nombre de archivo para colocar el que tengas en el directorio.

Con ese script ya hemos incluido todas las funciones de jQuery dentro de nuestra página. Sólo tienes que prestar atención a que tanto el archivo .html de esta página, como el archivo .js del framework estén en el mismo directorio. Y, como decía, que el archivo que incluimos con la etiqueta SCRIPT sea el .js que nosotros hemos descargado.

Además, como se puede ver, hemos dejado dentro del HEAD una etiqueta SCRIPT de apertura y cierre que está vacía. Todo el código que vamos a explicar a continuación tendrás que colocarlo en dentro de esa etiqueta.

1.3.3.- Ejecutar código cuando la página ha sido cargada

El paso que vamos a explicar ahora es importante que se entienda, aunque sin lugar a dudas a lo largo del manual publicado en DesarrolloWeb.com, lo veremos hasta la saciedad. 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.

En el taller de programación con el [DOM de Javascript](#) hemos podido explicar que es el DOM y la importancia de realizar acciones sólo cuando el DOM está listo. Pero si no queremos entretenernos con la lectura de este texto, sirve con saber que, 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 comandos.

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.

1.3.4.- Insertar un manejador de evento a la etiqueta A (enlace) que hay en la página

Ahora que ya sabemos cómo y cuando debemos ejecutar las acciones de jQuery que alteren la funcionalidad, contenidos o aspecto de la página, podemos insertar un poco de código para demostrar el método de trabajo con jQuery.

Para este primer ejemplo vamos a crear un evento click en el enlace, para mostrar un mensaje cuando se haga clic sobre el

link. Para crear un evento click sobre un elemento tenemos que invocar al método click sobre ese elemento y pasarle como parámetro una función con el código que queremos que se ejecute cuando se hace clic.

```
$("#a").click(function(evento){
    //aquí el código que se debe ejecutar al hacer clic
});
```

Como vemos en el código anterior, con \$("#a") obtenemos una referencia al enlace. Bueno, en realidad con ello estamos seleccionando todas las etiquetas A (enlaces) del documento, pero como sólo hay un enlace, en realidad nos vale. Luego, el método click() del sobre \$("#a") estamos definiendo un evento, que se ejecutará cuando se haga clic sobre el enlace. Como se puede ver, al método click se le pasa una función donde se debe poner el código Javascript que queremos que se ejecute cuando se haga clic sobre el enlace.

Ahora veamos la definición del evento clic completa, colocada dentro del evento ready del document, para que se asigne cuando la página está lista.

```
$(document).ready(function(){
    $("#a").click(function(evento){
        alert("Has pulsado el enlace...nAhora serás enviado a DesarrolloWeb.com");
    });
});
```

Por líneas, esto es una recapitulación de lo que hemos hecho:

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

Esta línea sirve para hacer cosas cuando la página está lista para recibir instrucciones jQuery que modifiquen el DOM.

```
$("#a").click(function(evento){
```

Con esta línea estamos seleccionando todas las etiquetas A del documento y definiendo un evento click sobre esos elementos.

```
alert("Has pulsado el enlace...nAhora serás enviado a DesarrolloWeb.com");
```

Con esta línea simplemente mostramos un mensaje de alerta informando al usuario que se ha hecho clic sobre el enlace.

1.3.5.- Guarda el archivo html y ábrelo en un navegador

Una vez que tenemos hecho nuestra primera página con jQuery, la puedes guardar y ejecutarla en un navegador. Prueba hacer clic en el enlace y debería salirte la ventana de alerta.

Puedes ver este [script en funcionamiento en una página aparte](#).

Ya está hecho y funcionando tu primer script con jQuery!

Por si acaso quedaron dudas, dejamos aquí el código completo que deberías tener:

```
<html>
<head>
    <title>Primer script con jQuery</title>
<script src="jquery-1.3.2.min.js" type="text/javascript"></script>
<script>
$(document).ready(function(){
    $("#a").click(function(evento){
        alert("Has pulsado el enlace...nAhora serás enviado a DesarrolloWeb.com");
    });
});
</script>
</head>

<body>

<a href="http://www.desarrolloweb.com">Ir a DesarrolloWeb.com</a>

</body>
</html>
```


1.3.6.- Extra: Bloquear el comportamiento normal de un enlace

Ahora veamos una pequeña modificación para alterar el comportamiento por defecto de los enlaces. Como sabemos, cuando se pulsa un enlace nos lleva a una URL. Luego al hacer click, primero se ejecuta lo que hayamos colocado en el evento click del enlace y luego el enlace lleva al navegador a una nueva URL.

Este comportamiento se puede bloquear también desde jQuery, añadiendo una llamada al método `preventDefault()` sobre el evento. Si te fijas, la función definida para marcar el comportamiento del evento click sobre el enlace recibía un parámetro. Ese parámetro es un manejador de evento. Y tiene sus propios métodos y propiedades, como este `preventDefault()` que comentábamos. Su uso es el siguiente:

```
$(document).ready(function() {
    $("a").click(function(evento){
        alert("Has pulsado el enlace, pero vamos a cancelar el evento...\nPor tanto, no vamos a llevarte a DesarrolloWeb.com");
        evento.preventDefault();
    });
});
```

Como hemos podido ver invocando a `evento.preventDefault()` lo que conseguimos en este caso es que el link no lleve a ningún sitio, simplemente se ejecutará el código Javascript contenido para el evento click.

Artículo por *Miguel Angel Alvarez*

1.4.- Básicos jQuery: añadir y quitar clases CSS sobre elementos

Otro ejemplo básico con jQuery, para mostrar cómo se pueden añadir y quitar clases CSS a elementos de la página, bajo respuesta de eventos de usuario.

Para ir acostumbrándonos al trabajo con el framework Javascript jQuery vamos a seguir haciendo ejemplos simples de funcionamiento, que vamos a explicar en la medida de las posibilidades con lo que hemos conocido hasta ahora en el [Manual de jQuery](#).

Claro que estos ejercicios son un poco especiales, dado que sirven para ilustrar el modo de trabajo con jQuery, pero sin explicar todos los detalles relacionados con el uso del framework. Por que de momento lo que queremos es mostrar una introducción al sistema y mostrar por encima algunas de sus posibilidades. En el futuro publicaremos artículos que irán poco a poco explicando todos los detalles de trabajo con jQuery.

En el caso que nos ocupa, queremos demostrar el uso de jQuery para alterar elementos de una página web, añadiendo y quitando clases CSS. Esto es bien simple, porque en jQuery los elementos tienen dos clases llamadas `addClass()` y `removeClass()`, que sirven justamente para que el elemento que recibe el método se le aplique una clase CSS o se le elimine. Así que lo que vamos a aprender, con respecto al artículo anterior -[Pasos para utilizar jQuery](#)-, es utilizar esos nuevos métodos de los elementos.

Para complicarlo sólo un poco más, vamos a añadir y quitar clases del elemento con respuesta a acciones del usuario, para aprender también nuevos eventos de usuario.

En nuestro ejemplo vamos a tener dos elementos. Primero una capa DIV con un texto. Después tendremos un enlace que estará fuera de la capa DIV. Al situar el usuario el ratón sobre un enlace añadiremos una clase CSS a la capa DIV y al retirar el ratón del enlace eliminaremos la class CSS que habíamos añadido antes. Si se desea, para aclarar el caso de nuestro ejemplo, podemos [ver el ejercicio en marcha en una página aparte](#).

Nota: Se supone que ya hemos leído el artículo anterior, en el que explicamos paso por paso [hacer tu primera página con jQuery](#), pues necesitaremos conocer algunas cosas que ya se han comentado allí.

1.4.1.- Crear la página con una capa, un enlace y la definición de una clase CSS

El primer paso sería construir una página con todos los elementos que queremos que formen parte de este primer ejemplo: la capa DIV, el enlace y la definición de la class CSS.

Además, ahora también vamos a incluir el script de jQuery, que lo necesitaremos para acceder a las funciones del framework Javascript.

```
<html>
<head>
  <title>Añadir y quitar clases CSS a elementos</title>
  <script src="jquery-1.3.2.min.js" type="text/javascript"></script>
<style type="text/css">
.clasecss{
  background-color: #ff8800;
  font-weight: bold;
}
</style>
</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.desarrolloweb.com">Añadir y quitar clase en la capa de arriba</a>
</body>
</html>
```

Perfecto, ahora ya tenemos la infraestructura necesaria para nuestro ejemplo, con todos los integrantes del mismo. Sólo nos faltaría hacer el siguiente paso, que es añadir los comportamientos dinámicos con jQuery.

1.4.2.- Recordar: añadir siempre los scripts jQuery cuando el documento está "ready"

Ahora vamos a empezar a meter el código Javascript, pero quiero comenzar por recordar a los lectores que cualquier funcionalidad que queramos agregar a la página por medio de jQuery, debe ser incluida cuando el documento está listo para recibir acciones que modifiquen el DOM de la página.

Para esto tenemos que incluir siempre el código:

```
$(document).ready(function() {
  //aquí meteremos las instrucciones que modifiquen el DOM
});
```

1.4.3.- Añadir los eventos mouseover y mouseout para añadir y quitar una clase CSS

En este paso vamos a crear un par de eventos que asociaremos a los enlaces. Este par de eventos serán lanzados cuando el usuario coloque el puntero del ratón sobre el enlace (mouseover) y cuando el usuario retire el ratón del enlace (mouseout).

Por ejemplo, para definir un evento mouseover se tiene que llamar al método mouseover() sobre el elemento que queremos asociar el evento. Además, al método mouseover() se le envía por parámetro una función con el código que se quiere ejecutar como respuesta a ese evento.

En el caso de añadir una clase tenemos que utilizar el método addClass(), que se tiene que invocar sobre el elemento al que queremos añadirle la clase. A addClass() tenemos que pasarle una cadena con el nombre de la clase CSS que queremos añadir.

Para seleccionar el elemento que queremos añadir la clase hacemos \$("#idElemento"), es decir, utilizamos la función dólar pasándole el identificador del elemento que queremos seleccionar, precedida del carácter "#". Por ejemplo, con \$("#capa") estamos seleccionando un elemento de la página cuyo id="capa".

```
$("#a").mouseover(function(event) {
  $("#capa").addClass("clasecss");
});
```

```
});
```

De manera análoga, pero con el método `mouseout()`, definimos el evento para cuando el usuario saca el puntero del ratón del enlace.

```
$("#a").mouseout(function(event) {  
    $("#capa").removeClass("clasecss");  
});
```

1.4.4.- Código completo del ejemplo jQuery

Ahora veamos el código completo de este ejercicio.

```
<html>  
<head>  
    <title>Añadir y quitar clases CSS a elementos</title>  
    <script src="jquery-1.3.2.min.js" type="text/javascript"></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.desarrolloweb.com">Añadir y quitar clase en la capa de arriba</a>  
</body>  
</html>
```

Podemos ver el [ejemplo desarrollado en el artículo en una página aparte](#).

Artículo por *Miguel Angel Alvarez*

1.5.- Mostrar y ocultar elementos de la página con jQuery

Con el método `css()` de jQuery podemos aplicar cualquier estilo css a elementos de la página. Veremos cómo aplicarlo para mostrar y ocultar elementos de la página.

Para aprender rápidamente cómo hacer cosas típicas con jQuery, vamos a explicar cómo mostrar u ocultar elementos de la página, cambiando propiedades de las hojas de estilo en cascada. Para que este artículo tenga un poco más de practicidad, vamos a realizar un ejemplo de formulario donde algunos de los elementos están escondidos. En ese formulario, al marcar una opción de un campo checkbox, se mostrarán esos elementos escondidos y al desmarcar esa opción, se ocultarán.

El método que se dispone en jQuery para alterar las hojas de estilo se llama `css()` y lo podemos invocar sobre elementos de

la página, a los que queremos cambiar sus propiedades CSS. En principio, para mostrar y ocultar elementos, nos viene bien alterar el atributo "display", poniendo el valor "none" para que no aparezca y "block" para que sí lo haga.

Lo que deberíamos saber para poder entender este ejemplo se puede aprender en el [Manual de jQuery](#) que venimos publicando en DesarrolloWeb.com. Aunque como este es un ejemplo básico, será suficiente con estudiar el artículo [Pasos fundamentales para usar jQuery](#).

1.5.1.- Ocultar y mostrar una capa con jQuery

Vamos a ver brevemente cómo usar el mencionado método `css()`. Primero, tendríamos que tener un elemento en la página, que es el que vamos a mostrar u ocultar.

```
<div id="mielemento">
Contenido del elemento...
</div>
```

Para ocultar este elemento, habría que invocar el método `css()` de la siguiente manera:

```
$("#mielemento").css("display", "none");
```

Como se puede ver, se accede al elemento con la función dólar `$()` y el selector `"#mielemento"`. Luego al método `css()` le pasamos el valor "display" y "none", porque queremos alterar a propiedad display y asignarle el valor "none", para ocultar el elemento.

Para mostrarlo haríamos algo parecido, pero colocando el valor "block" en el atributo CSS "display".

```
$("#mielemento").css("display", "block");
```

Nota: el método `css()` admite otros parámetros. Si sólo recibe un parámetro, de tipo string, devuelve el valor CSS asignado a ese parámetro. También podría recibir un sólo parámetro, en este caso de con una notación de objeto con pares llave/valor, y entonces asignaría todos esos estilos CSS, especificados por los pares llave/valor en el objeto, al elemento de la página donde se haya invocado el método.

1.5.2.- Mostrar u ocultar elementos como respuesta a un evento

Ahora que ya sabemos cómo realizar un cambio en el atributo display, vamos a ver cómo poner esta instrucción en marcha cuando el usuario realice acciones en la página. Recordar que al principio del artículo comentaba que íbamos a crear un formulario con una casilla de selección (campo checkbox) y que al activar ese campo se mostrarían otros contenidos en el formulario. Al desactivarlo, se ocultarían esos contenidos del formulario.

Para entender bien lo que deseamos hacer, podemos [ver el ejercicio en marcha en una página aparte](#).

Lo primero que podemos presentar es el formulario con el que vamos a trabajar.

```
<form>
Nombre: <input type="text" name="nombre">
<br>
<input type="checkbox" name="mayor_edad" value="1" 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>
```

Como se podrá ver, es un formulario como otro cualquiera. Sólo que tiene una capa con `id="formulariomayores"`, que contiene los elementos del formulario que queremos mostrar u ocultar al marcar o desmarcar el checkbox. Esa capa estará inicialmente oculta, y para ello hemos colocado el atributo `style="display: none;"`. Podemos fijarnos también en el checkbox con `id="mayoria_edad"`, que es el que va servir para marcar si se desea o no ver la parte final del formulario.

Ahora hay que hacer cosas cuando se haga clic en el checkbox con `id="mayoria_edad"`. Para ello en deberíamos crear un código Javascript y jQuery como este:

```
$(document).ready(function() {
    $("#mayoria_edad").click(function() {
        //lo que se desee hacer al recibir un clic el checkbox
    });
});
```

```
});  
});
```

Como ya hemos comentado, estas líneas sirven para especificar eventos. `$(document).ready()` se hace para lanzar instrucciones cuando el navegador está preparado para recibir las y `$("#mayoria_edad").click()` sirve para realizar acciones cuando se ha hecho clic sobre el elemento con id "mayoria_edad", que es el checkbox del formulario. (Lee el artículo [Pasos para trabajar con jQuery](#) para más información sobre este punto).

Ahora tenemos que ver las instrucciones que debemos ejecutar como respuesta a ese evento.

```
if ($("#mayoria_edad").attr("checked")){  
    $("#formulariomayores").css("display", "block");  
}else{  
    $("#formulariomayores").css("display", "none");  
}
```

Básicamente lo que hacemos es comprobar el estado del atributo "checked" del elemento "#mayoria_edad". Esto se hace con el método `attr()` indicando como parámetro el valor del atributo HTML que queremos comprobar. Entonces, si estaba "checked", se tiene que mostrar el elemento y si no estaba marcado el checkbox, habría que ocultarlo.

Espero que se haya entendido bien. Ahora dejo aquí el código completo de este ejemplo:

```
<html>  
<head>  
    <title>Mostrar Ocultar</title>  
<script src="jquery-1.3.2.min.js" type="text/javascript"></script>  
<script>  
$(document).ready(function(){  
    $("#mayoria_edad").click(function(evento){  
        if ($("#mayoria_edad").attr("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="1" 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>
```

Finalmente, podemos [verlo en marcha en una página aparte](#).

Artículo por Miguel Ángel Álvarez

1.6.- Efectos rápidos con jQuery

Algunos efectos dinámicos se pueden hacer con jQuery con rapidez impactante y extremadamente poco código Javascript.

Una de las ventajas más destacadas de jQuery es la realización de efectos especiales para páginas web, que se desarrollan

rápido y con poco código fuente. Estos efectos sirven para aplicar dinamismo a una página web y una respuesta atractiva frente a la interacción con el usuario, lo que hace que las páginas programadas con jQuery ofrezcan una imagen puntera.

Los efectos con jQuery, al menos un buen puñado de ellos, se pueden realizar sin muchas complicaciones, ya que existen unas funciones que simplifican la tarea de los desarrolladores ([Ver la librería Effects](#)). En muchos casos conseguir un efecto nos llevará una línea de código en nuestro programa, como esta:

```
$("#capaefectos").hide("slow");
```

Con esto conseguimos que el elemento con id="capaefectos" desaparezca de la página. Pero además, el efecto no es un simple fundido del elemento en la página (hacerse transparente), sino que también va acompañado de una reducción de tamaño progresiva hasta desaparecer.

Combinando los efectos con la interacción de usuario, por medio de eventos, podemos conseguir que los efectos respondan a las acciones del visitante, lo que multiplica las posibilidades, manteniendo la sencillez, elegancia y facilidad de manutención del código Javascript. Lo vamos a ver en un ejemplo a continuación.

1.6.1.- Ejemplo de efectos e interacción en jQuery

En el siguiente ejemplo vamos a mostrar un uso sencillo de las funciones de efectos de jQuery. Vamos a implementar un simple efecto de ocultar y mostrar un elemento de la página web.

Podemos ver el [ejemplo en marcha en una página aparte](#).

Como hemos podido ver, vamos a tener una capa y un par de enlaces. Con jQuery haremos que al pulsar los enlaces se oculte y se muestre la capa, con las funciones de la librería Effects.

Para comenzar, este es el código HTML del ejemplo, que comprende tanto la capa como los enlaces.

```
<div id="capaefectos" style="background-color: #cc7700; color:fff; padding:10px;">
Esto es una capa que nos servirá para hacer efectos!
</div>

<p>
<a href="#" id="ocultar">Ocultar la capa</a> |
<a href="#" id="mostrar">Mostrar la capa</a>
</p>
```

Ahora viene la parte interesante, que es en la que asociamos eventos a estos dos enlaces y codificamos las llamadas a las funciones de Effects, que harán que se muestre y oculte la capa.

El código Javascript, que hace uso de jQuery sería el siguiente:

```
$(document).ready(function() {
    $("#ocultar").click(function(event) {
        event.preventDefault();
        $("#capaefectos").hide("slow");
    });

    $("#mostrar").click(function(event) {
        event.preventDefault();
        $("#capaefectos").show(3000);
    });
});
```

Como se puede ver, primero tenemos que definir el evento ready del objeto \$(document), para hacer cosas cuando el documento está preparado para recibir acciones.

Luego se define el evento click sobre cada uno de los dos enlaces. Para ello invoco el método click sobre el enlace, que hemos seleccionado con jQuery a través del identificador de la etiqueta A.

```
$("#ocultar").click(function(event) {
```

Con esto estoy definiendo el evento clic sobre el elemento con id="ocultar".

Nota: leer el artículo anterior [Pasos para utilizar jQuery en tu página web](#), en el que hablábamos sobre eventos y otras generalidades de este

framework Javascript. Podremos encontrar explicaciones más detalladas sobre cómo definir eventos Javascript con jQuery.

Dentro de la función a ejecutar cuando se hace clic, se coloca la llamada a la función de los efectos.

```
$("#capaefectos").hide("slow");
```

Esto hace que nuestra capa, a la que habíamos puesto el identificador (atributo id) "capaefectos", se oculte. Pasamos el parámetro "slow" porque queremos que el efecto sea lento.

Ahora veamos la función de los efectos con otra llamada:

```
$("#capaefectos").show(3000);
```

Esto hace que se muestre el elemento con id "capaefectos", y que el proceso de mostrarse dure 3000 milisegundos.

No hay más complicaciones, así que si habéis entendido esto ya sabéis hacer efectos simples pero atractivos con jQuery en vuestra página web. Ahora podréis ver el código completo de este ejemplo creado por DesarrolloWeb.com para demostrar el uso de efectos.

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html>
<head>
  <title>Efectos con jQuery</title>
  <script src="jquery-1.3.2.min.js" type="text/javascript"></script>
<script>
$(document).ready(function() {
  $("#ocultar").click(function(event) {
    event.preventDefault();
    $("#capaefectos").hide("slow");
  });

  $("#mostrar").click(function(event) {
    event.preventDefault();
    $("#capaefectos").show(3000);
  });
});
</script>
</head>

<body>

<div id="capaefectos" style="background-color: #cc7700; color:fff; padding:10px;">
Esto es una capa que nos servirá para hacer efectos!
<br>
<br>
Pongo este texto simplemente de prueba
</div>

<p>
<a href="#" id="ocultar">Ocultar la capa</a> |
<a href="#" id="mostrar">Mostrar la capa</a>
</p>

</body>
</html>
```

Por último, pongo el [enlace de nuevo al ejemplo en marcha](#).

Como se ha podido comprobar, hacer efectos con jQuery es bastante sencillo. Claro que hay otros detalles importantes y otros tipos de efectos y funcionalidades de personalización de los mismos, pero esto nos ha servido para demostrar lo sencillo que es trabajar con este framework Javascript. En siguientes artículos seguiremos explorando casos de uso típicos de jQuery.

Artículo por *Miguel Angel Alvarez*

1.7.- Callback de funciones jQuery

Con callback de jQuery podemos hacer una secuencia de llamadas a funciones o una pila de funciones que se ejecutarán una detrás de otra.

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.

Nota: para poder entender mejor estas llamadas a efectos, por favor, consulta el artículo [Efectos Rápidos con jQuery](#).

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.

Lo mejor para darse cuenta de este caso es [verlo en marcha](#).

1.7.1.- Cómo realizar una pila de ejecución de funciones

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 el 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()`.

1.7.2.- Ejemplo real de callback con jQuery

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.

```
$("#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 que tiene que durar el efecto fade out (fundido hacia transparente), y luego la función callback, que se ejecutará después de que `fadeOut()` haya terminado.

Como el método `css()` (se encuentra como primera instrucción de la función callback) es instantáneo, no necesita hacerse un callback para ejecutar el `fadeIn()`, sino que se puede escribir directamente en la siguiente línea de código. Así pues, se ve que el callback, al menos en este ejemplo, sólo es necesario hacerlo cuando se ejecutan funciones que realizarán un procesamiento prolongado.

Podemos [ver este ejemplo de callback en una página aparte](#).

Hasta aquí, a lo largo de los capítulos de este [manual de jQuery](#), hemos leído la introducción a este popular framework Javascript, tal como se puede ver en el apartado "How to use jQuery" publicada en la [página de documentación](#). Desde DesarrolloWeb.com hemos enriquecido este tutorial de jQuery aportando nuevos ejemplos y explicaciones adicionales, encaminadas a que cualquier persona, con unos conocimientos básicos de Javascript, pueda entender y aprender a usar estas librerías de programación cross-browser. Ahora podemos hacer una pausa en este manual y volveremos pronto con nuevos artículos para explicar otros asuntos sobre la programación con jQuery.

Artículo por Miguel Ángel Álvarez

1.8.- Uso de Ajax muy sencillo con jQuery

Es muy fácil desarrollar Ajax en jQuery. En este artículo veremos el ejemplo más sencillo de Ajax con el framework Javascript jQuery.

Ha llegado el momento de hacer una primera aproximación a Ajax, en la serie de artículos que estamos publicando en DesarrolloWeb.com para mostrar el uso de este framework (léase el [Manual de jQuery](#)).

Una de las ventajas de los [frameworks Javascript](#) es que nos permiten desarrollar scripts que hacen uso de Ajax de una manera muy fácil y encima, sin tener que complicarnos la vida con la compatibilidad entre distintos navegadores. Sin duda, cualquier persona que sepa un poco de Javascript, podría en poco tiempo empezar a utilizar Ajax con alguno de estos frameworks. Nosotros vamos a demostrar cómo es de sencillo en jQuery.

La primera impresión que he tenido sobre el uso de Ajax en jQuery es realmente grata, por la facilidad con la que se puede realizar un primer ejemplo. Se trata de un ejemplo extremadamente sencillo, pero sirve para abrirnos las puertas a muchas aplicaciones interesantes. Habíamos visto en otros frameworks Javascript ejemplos similares, como en el artículo [Ajax con Mootools](#), pero tenemos que quitarnos el sombrero ante la extremada sencillez con la que se puede hacer un ejemplo similar en jQuery. Sea suficiente decir que en este ejemplo de Ajax utilizaremos tan sólo 4 líneas de código, de las cuales sólo 1 es para ejecutar la propia llamada al servidor por Ajax.

1.8.1.- Traer un contenido con Ajax al pulsar un enlace

En este sencillo ejemplo haremos llamada a Ajax, para traer un contenido, cuando se pulse un enlace. Esto lo hemos puesto en marcha en el servidor de DesarrolloWeb.com y [lo puedes ver en una página aparte](#).

Aquí podemos ver el enlace, al que ponemos un identificador (atributo id) para seleccionarlo desde jQuery.

```
<a href="#" id="enlaceajax">Haz clic!</a>
```

Si hemos leído hasta aquí el [Manual de jQuery](#) podremos saber cómo asignar un evento a un enlace. No obstante, recordemos cómo asignar una función para cuando se haga clic en el enlace:

```
$(document).ready(function() {  
    $("#enlaceajax").click(function(evento) {  
        //elimino el comportamiento por defecto del enlace  
        evento.preventDefault();  
        //Aquí pondría el código de la llamada a Ajax  
    });  
});
```

Ya se trata sólo de poner en marcha Ajax dentro de la función del evento "click" sobre el enlace. Pero antes voy a necesitar una capa donde mostrar el contenido que vamos a recibir del servidor con la llamada Ajax. Le pondremos id="destino" para poder referirnos a ella desde jQuery:

Y ahora la parte más interesante, donde podemos ver qué tan fáciles son las cosas con este framework Javascript. Una única línea de código será suficiente:

```
$("#destino").load("contenido-ajax.html");
```

Con esta simple sentencia estamos realizando una llamada a Ajax con jQuery. Es una simple invocación al método load() de un elemento de la página, en concreto el que habíamos puesto con id="destino". Al método load() le pasamos como parámetro la ruta de la página que queremos cargar dentro del elemento.

El archivo que cargamos con load() en este ejemplo es "contenido-ajax.html" y simplemente le hemos colocado un texto cualquiera. Lo hemos guardado en el mismo directorio que la página web donde está el script jQuery.

Nota: para que este ejemplo funcione debe colocarse en un servidor web, puesto que la llamada por Ajax se hace por http y el archivo que se carga entonces tiene que recibirse por un servidor web, que lo mande con ese protocolo al navegador. Si pones los archivos en tu disco duro y los ejecutas tal cual, no te funcionará. Puedes utilizar cualquier espacio de hosting que tengas o bien un servidor web que puedas tener instalado en tu ordenador.

Así de simple! Podemos ver el código completo de este ejemplo:

```
<html>  
<head>  
    <title>Ajax Simple</title>  
<script src="jquery-1.3.2.min.js" type="text/javascript"></script>  
<script>  
$(document).ready(function() {  
    $("#enlaceajax").click(function(evento) {  
        evento.preventDefault();  
        $("#destino").load("contenido-ajax.html");  
    });  
});  
</script>  
</head>  
<body>  
  
<a href="#" id="enlaceajax">Haz clic!</a>  
<br>  
<div id="destino"></div>  
  
</body>  
</html>
```

Podemos [ver el ejemplo en marcha en una página aparte](#).

Cabría comentar que jQuery tiene muchos otros métodos de realizar conexiones por Ajax, que pueden servir para muchos otros casos de trabajo que podemos encontrarlos. Nosotros hemos escogido el más sencillo para dar una primera demostración de las posibilidades.

1.8.2.- Pasar parámetros y ejecutar acciones después de la llamada a Ajax

El método `load()` que hemos visto en el ejemplo anterior tiene otros dos parámetros opcionales que podemos utilizar si fuera necesario:

Parámetros a pasar a la página: la página que carguemos con Ajax puede recibir parámetros por la URL, que se especifican con la típica notación de propiedades y valores de jQuery.

```
{nombre: "Pepe", edad: 45}
```

Por ejemplo, con ese código estaríamos enviando a la página los datos nombre y edad, con los valores "pepe" y 45. Esos datos viajarían en la URL, por el método "POST".

Nota: Desde jQuery 1.3 también se pueden enviar los parámetros a la página a cargar con Ajax por medio de una variable de tipo string, en lugar de una notación de objetos como hemos comentado. Cuando se use un string para especificar los parámetros a enviar en el request http, éstos viajan por el método GET. Cuando se utiliza una notación de objetos como la que hemos visto, los datos viajan por el método POST.

Función callback: como otros métodos de jQuery, podemos especificar opcionalmente una función a ser ejecutada cuando se termine de ejecutar el método. En este caso, cuando se termine la llamada Ajax, se pueden hacer acciones, como borrar un mensaje típico de "cargando...".

Nota: En un artículo anterior ya comentamos el habitual uso de [funciones callback en jQuery](#).

Ahora veamos un código donde hacemos uso de estos dos parámetros:

```
$(document).ready(function() {
    $("#enlaceajax").click(function(evento) {
        evento.preventDefault();
        $("#destino").load("recibe-parametros.php", {nombre: "Pepe", edad: 45}, function() {
            alert("recibidos los datos por ajax");
        });
    });
});
```

En este caso estamos cargando con `load()` una página PHP llamada "recibe-parametros.php". Estamos pasando los parámetros "nombre" y "edad" a una página, que podremos recoger por GET. Además, hemos colocado una función callback en la que simplemente hacemos un `alert()`, que se ejecutará cuando la llamada a Ajax haya terminado.

Este sería el código fuente de "recibe-parametros.php":

```
Recibido el siguiente dato:
<br>
Nombre: <?php echo $_POST["nombre"];?>
<br>
Edad: <?php echo $_POST["edad"];?>
```

Podemos [ver este ejemplo en una página aparte](#).

Con esto hemos podido comprobar lo sencillo que es realizar con jQuery una carga de contenidos que se reciben por Ajax. Como decía, existen muchas otras maneras de hacer conexiones Ajax con jQuery, como el ejemplo del artículo siguiente que nos enseña a [mostrar un mensaje de carga mientras esperamos la respuesta Ajax del servidor](#). Además, para complementar esta información, también podéis ver el [vídeo de Ajax con jQuery](#).

Artículo por Miguel Ángel Álvarez

1.9.- Ajax jQuery con mensaje de carga

Vemos más posibilidades de Ajax en jQuery, modificando un ejemplo anterior, para crear un mensaje de carga mientras que el usuario espera la respuesta Ajax del servidor.

Queremos ver algunas cosas típicas que podríamos desear hacer con Ajax en una página web, facilitando el proceso de desarrollo con el framework Javascript jQuery. En esta línea de artículos ya publicamos hace poco un artículo sobre el uso de [Ajax en jQuery simplificado](#). En el mencionado artículo vimos cómo hacer una llamada Ajax con 1 sola línea de código (el Ajax en sí era una línea de código, aunque se necesitan varias instrucciones más para asociar las acciones Ajax como respuesta a eventos en la página).

Una de las cosas que más habitualmente podríamos desear hacer cuando se realiza una llamada Ajax es la creación de un mensaje de carga, que podemos colocar con un simple texto "cargando..." o con la típica imagen de [Ajax Loader](#). En este artículo veremos cómo crear ese mensaje de carga al realizar una solicitud Ajax con jQuery.

Para los interesados, se puede [ver este ejemplo de Ajax en una página aparte](#).

1.9.1.- Por qué un mensaje de carga al hacer Ajax

Cuando hacemos una solicitud por Ajax, los resultados de dicha llamada a veces tardan en llegar. Durante ese tiempo el usuario puede no ver ninguna reacción por parte del navegador, lo que le puede confundir y pensar que no ha hecho clic correctamente en el enlace o botón. Sería normal en ese caso que el usuario pulse repetidas veces un enlace o un botón de envío de formulario, generando repetidas e innecesarias llamadas al servidor, lo que puede derivar en diversos problemas.

Así pues, es conveniente mostrar un mensaje de carga para advertir que su solicitud fue realizada y el proceso está en marcha y esperando respuesta del servidor.

Vamos a explicar la implementación de este mensaje de carga, pero siempre teniendo en cuenta que nuestra intención en este ejemplo es mantener un código pequeño que se pueda entender fácilmente. Aunque queremos remarcar que las cosas se podrían hacer de otra manera, algo mejorada, cuando sepamos más cosas sobre el framework Javascript jQuery y pongamos en marcha algunas prácticas aconsejables de programación orientada a objetos.

1.9.2.- Preparando el código HTML

Como un primer paso, vamos a mostrar el código HTML que tendremos en la página que hará la solicitud Ajax.

```
<a href="#" id="enlaceajax">Haz clic!</a>
<div id="cargando" style="display:none; color: green;">Cargando...</div>
<br>
<div id="destino"></div>
```

Como se puede ver, tenemos tres elementos: 1) el enlace, que activará la llamada a Ajax cuando se haga clic sobre él. 2) una capa con id="cargando" que es donde está el mensaje de carga (nosotros hemos colocado un texto, pero se podría colocar cualquier cosa, como el típico gif animado que muestra que se está procesando la solicitud (conviene fijarse también que esa capa cargando está oculta de momento, gracias al atributo de estilo CSS display:none). 3) la capa "destino", donde mostraremos la respuesta recibida tras la solicitud Ajax.

1.9.3.- Llamada a Ajax con jQuery y el mensaje de carga

En este punto vamos a describir cómo se tendría que hacer la llamada al servidor con Ajax, pero lo cierto es que este proceso ya lo explicamos con detalle anteriormente, por lo que os refiero al artículo [Uso de Ajax muy sencillo con jQuery](#), donde encontraréis unas explicaciones que voy a dar por sabidas en este caso. Lo que explicaré en este artículo es cómo se tiene que mostrar el mensaje de carga cuando se inicia la llamada y como eliminarlo una vez hemos recibido la respuesta por Ajax.

Otra cosa que el lector deberá conocer para poder entender este ejemplo es [Mostrar y ocultar elementos de la página con jQuery](#).

```
$(document).ready(function(){
    $("#enlaceajax").click(function(evento){
        evento.preventDefault();
        $("#cargando").css("display", "inline");
        $("#destino").load("pagina-lenta.php", function(){
            $("#cargando").css("display", "none");
        });
    });
});
```

```
}}
```

Voy comentando línea a línea el código anterior.

En la primera línea se está especificando un método `ready()` sobre el objeto `document`, que sirve para generar un código a ser ejecutado cuando la página está lista para recibir instrucciones Javascript que trabajen con el DOM.

En la segunda línea se accede al elemento con identificador "enlaceajax" (es decir, el enlace que activará el Ajax) para definir un evento "click".

En la siguiente línea se ejecuta el método `preventDefault()` sobre el evento producido al hacer clic sobre el enlace. Esto se hace para anular el comportamiento típico del enlace.

Ahora viene la parte en la que se mostrará el mensaje de carga:

```
$("#cargando").css("display", "inline");
```

Simplemente se muestra la capa con `id="cargando"`, con un simple cambio en el atributo CSS `display` de la capa. Ese cambio de atributo lo hacemos con el método `css()` sobre el elemento que queremos alterar, tal como se vio en el artículo [Mostrar y ocultar elementos de la página con jQuery](#).

Ahora, con la siguiente línea de código:

```
$("#destino").load("pagina-lenta.php", function(){
```

Se hace la llamada Ajax, con el método `load()` sobre la capa que queremos actualizar con el contenido traído por Ajax, que es la capa con `id="destino"`. Este método recibe la URL de la página a cargar y una [función callback](#), que se ejecutará después que el método `load()` se haya terminado de procesar, esto es, después de que la solicitud Ajax se haya recibido y se haya mostrado su contenido en la capa que recibe el método.

Entonces, en esa función callback, tenemos que ocultar la capa con el mensaje de carga, puesto que cuando se ejecute esta función ya se habrá realizado todo el procesamiento Ajax. Eso lo conseguimos con el método `css()`, alterando la propiedad `display`, de manera similar a como lo habíamos realizado para mostrar la capa cargando.

```
$("#cargando").css("display", "none");
```

Esto es todo. Realmente no tiene ninguna complicación especial. Aunque, como decía, estas cosas se podrán hacer de una manera más elegante cuando aprendamos un poquito más sobre jQuery.

Por si sirve para aclarar las cosas, dejo a continuación el código completo de la página que hace la solicitud con jQuery:

```
<html>
<head>
  <title>Ajax Simple</title>
<script src="jquery-1.3.2.min.js" type="text/javascript"></script>
<script>
$(document).ready(function(){
  $("#enlaceajax").click(function(evento){
    evento.preventDefault();
    $("#cargando").css("display", "inline");
    $("#destino").load("pagina-lenta.php", function(){
      $("#cargando").css("display", "none");
    });
  });
});
</script>
</head>

<body>
Esto es un Ajax con un mensaje de cargando...
<br>
<br>

<a href="#" id="enlaceajax">Haz clic!</a>
<div id="cargando" style="display:none; color: green;">Cargando...</div>
<br>
<div id="destino"></div>

</body>
</html>
```

1.9.4.- Código de la página PHP que se invoca por Ajax

El código de la página PHP que traemos por ajax "pagina-lenta.php" es el siguiente:

```
<?php
sleep(3);
echo ("He tardado 3 segundos en ejecutar esta página...");
?>
```

En realidad no tiene nada en especial. Simplemente hacemos un sleep(3) para ordenarle a PHP que espere 3 segundos antes de terminar de procesar la página y enviarla al cliente. Así consigo que la solicitud http tarde un poco en ser respondida y podamos ver el mensaje de carga durante un poco más de tiempo en la página.

Finalmente, pongo de nuevo el [enlace para ver este ejercicio en marcha](#).

Si te ha interesado este ejemplo y quieres obtener una ayuda adicional para crear tus propios scripts en Ajax, te recomendamos ver el [vídeo donde explicamos a hacer Ajax con jQuery](#).

Artículo por Miguel Angel Alvarez

Parte 2: Core de jQuery

Empezamos a adentrarnos en el framework Javascript para conocer los detalles de este sistema, comenzando con el núcleo de jQuery: el core, que contiene los métodos más esenciales.

2.1.- Core de jQuery

El core de jQuery lo forman las funciones más recurridas y que forman la base sobre la que se asienta el cualquier cosa en este framework Javascript.

Hasta este momento ya hemos publicado varios artículos en el [Manual de jQuery](#) de DesarrolloWeb.com, pero realmente lo que hemos visto es una mera introducción a las posibilidades del API, así como unas explicaciones básicas para empezar a trabajar con jQuery en nuestras páginas web. Sin embargo, nos gustaría ir publicando una documentación en español completa, para que cualquier desarrollador pueda aprender a fondo este framework Javascript y crear cualquier tipo de aplicación enriquecida del cliente.

Las personas con un nivel medio-alto de Javascript y de programación orientada a objetos, con lo que hemos visto hasta el momento en el manual y la referencia del [API jQuery](#), podrán con relativa facilidad adentrarse en características avanzadas del framework, pero para los demás y los que deseen una ayuda adicional, esperamos que este artículo y los siguientes puedan abrir un camino, sencillo de seguir, para aprender a hacer todo tipo de desarrollos con Javascript y jQuery.

Comencemos pues por el inicio, tal como está en la documentación del sistema, tratando en primer caso el "Core". De todos modos, cabe decir que lo que hemos podido ver en los artículos anteriores de este [Manual de jQuery](#), nos va a venir muy bien para poder entender todo lo que viene a continuación y sobre todo, poder poner en marcha ejemplos un poco más elaborados.

2.1.1.- Core de jQuery

El core de jQuery es la base sobre la que se trabaja para hacer cualquier cosa con jQuery. Contiene una serie de clases y métodos útiles para hacer tareas reiterativas, que vamos a necesitar en las aplicaciones. Integra desde funciones que serán útiles en cualquier script, por sencillo que sea, hasta funciones menos recurridas pero que nos facilitarán la vida a hora de hacer código limpio, corto y reutilizable.

Utilizaremos el Core para realizar cosas útiles con objetos, clases, datos, etc, pero realmente lo que más haremos será utilizar la función jQuery, que es el pilar fundamental sobre el que se basarán las aplicaciones.

Core tiene las funciones clasificadas en diversos apartados:

\$() (La función jQuery):

Es la función principal de jQuery, que además tiene diversas utilidades según los parámetros que se le envíen. Su utilidad principal es obtener elementos de la página.

Accesorios de objetos:

Es una gama de funciones de diversa y variada utilidad, que sirven de utilidad para hacer cosas con objetos, tales como iterar con cada uno de sus elementos, saber su tamaño, longitud, el selector o contexto con el que se obtuvo, obtener todos sus elementos DOM que contenga, etc.

Trabajo con datos:

Unas funciones útiles para trabajar con datos y asociarlos a elementos, una forma de guardar información adicional a elementos de la página. También tiene diversas funciones para trabajar con colas y administrar la salida y entrada de sus elementos.

Plugins:

Funciones que permiten extender los elementos jQuery para incorporar nuevos métodos, algo que se utiliza habitualmente a la hora de crear plugins para añadir funcionalidades a jQuery.

Interoperabilidad:

Funciones que permiten que jQuery no tenga problemas de compatibilidad con otras librerías Javascript que también suelen utilizar la función dólar `$()`.

Nota: En el momento de escribir este manual estamos en la release 1.3.2, en la que se han publicado como novedad un par de métodos, de los clasificados en accesorios de objetos. Como debemos saber, de vez en cuando actualizan las librerías para incorporar nuevos métodos.

En el próximo artículo comenzaremos a tratar a fondo el Core de jQuery, hablando de la [función dólar `\$\(\)`](#).

Artículo por Miguel Angel Alvarez

2.2.- Función jQuery o función `$()`

Veremos con detalle la función más importante de jQuery y para ello comenzaremos viendo su uso más común: seleccionar elementos de la página y obtener un objeto jQuery con los elementos seleccionados.

Con el objetivo de seguir tratando el [Core de jQuery](#), del que ya empezamos a hablar en este [Manual de jQuery](#), vamos a explicar la función jQuery, también conocida como `$()`.

El funcionamiento del Core de jQuery se basa en esta función. 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](#).

En realidad esta función se invoca también con el símbolo dólar `$()`, lo que sería una manera resumida de utilizarla y que nos ahorrará bastantes caracteres en el código y a la postre, tiempo de descarga y peso de los scripts de la página.

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

2.2.1.- 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.

Nota: Como hemos dicho, `$()` es un resumen o forma corta de invocar la función jQuery. También podríamos haber escrito la línea de esta manera: `var elem1 = jQuery("#elem1");`

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

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

Este método `css()` no forma parte del core, pero 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.

Podemos [ver en una página aparte este pequeño script en uso](#).

El código de esta página es el siguiente:

```
<html>  
<head>  
  <title>función jquery</title>  
  <script src="../jquery-1.3.2.min.js" type="text/javascript"></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>
```

Si queremos, podemos utilizar el segundo parámetro opcional, que es el contexto. Con él podríamos conseguir seleccionar elementos que pertenecen a una zona concreta de nuestro documento. Por defecto el contexto es la página entera, pero lo podemos restringir de esta manera:

```
var inputs = $("input", document.forms[0]);
inputs.css("color", "red");
```

Con la primera línea conseguimos seleccionar todos los elementos INPUT que pertenecen al primer formulario de la página. Se devolverá un objeto jQuery que contiene todos los input seleccionados. Con la segunda línea, invocando el método `css()` sobre el objeto jQuery recibido, estaríamos cambiando el color del texto de esos elementos. Esto no seleccionaría los INPUT de otros formularios, como se puede ver en esta [página de ejemplo](#).

Ahora por completar un poco más estas notas, veamos otro ejemplo en el que seleccionamos todos los párrafos (etiqueta P), pero restringimos el contexto únicamente los que están en un DIV con `id="div1"`.

```
var parrafos_div1 = $("p", "#div1");
parrafos_div1.hide();
```

En la segunda línea lanzamos el método `hide()` sobre el objeto jQuery que contiene los párrafos seleccionados, con lo que conseguimos que se oculten. Podemos [ver una página que con este ejemplo en marcha](#).

Contamos con un [video que nos habla de la función \\$ en jQuery](#).

En el siguiente artículo veremos [otros usos de la función jQuery\(\) / función dólar \\$\(\)](#).

Artículo por Miguel Ángel Álvarez

2.3.- Otros usos de la función \$()

La función jQuery(), o con su abreviación \$(), tiene otros usos interesantes cuando le enviamos parámetros distintos a los vistos en el artículo anterior.

En el [pasado capítulo del manual de jQuery](#) vimos unas primeras notas sobre el Core de jQuery y comenzamos a explicar la función `jQuery()`, que es la más importante en este framework Javascript. Ahora veremos como esta función, que también se puede acceder por medio del símbolo dólar `$()`, puede tener otras aplicaciones útiles, cuando recibe parámetros distintos a los que vimos anteriormente.

2.3.1.- 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");
```

Nota: el método `appendTo()` no pertenece al Core, pero nos viene bien utilizarlo y así hacer algo con los elementos que acabamos de crear.

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

```
<html>
<head>
  <title>función jquery</title>
  <script src="../jquery-1.3.2.min.js" type="text/javascript"></script>
</script>
<script>
$(document).ready(function() {
  var nuevosElementos = $("<div>Estos elementos ..</b>.<h1>Titulo...</h1></div>");
  nuevosElementos.appendTo("body");
});
</script>
</head>
<body>
<p>Esto es el cuerpo de la página, que tiene poca cosa...</p>
</body>
</html>
```

Ahora, dejamos el link para [ver el ejemplo en una página aparte](#).

2.3.2.- Función jQuery pasando elementos

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 `$(document.body)` enviando el parámetro `document.body`.

Nota: La variable `document.body` corresponde con un elemento del DOM de Javascript, que crea automáticamente el navegador y hace referencia al documento de la página.

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.

Algo como esto no funcionaría porque estaríamos intentando lanzar un método de jQuery directamente sobre el objeto DOM sin haberlo extendido:

```
document.body.css("background-color", "#ff8833");
```

No funcionará porque no podemos llamar a un método jQuery sobre un objeto DOM, si es que no lo hemos extendido con la función `$(document.body)`.

Nota: Esta función acepta documentos XML y objetos Window, aunque no son propiamente elementos del DOM.

Podemos [ver ahora una página donde se encuentra este ejemplo en marcha](#).

2.3.3.- Función jQuery pasando una función

En la función `$(document.body)` 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.

Nota: Ya explicamos lo que era un callback en el artículo [Callback de funciones jQuery](#)

En esa función podemos colocar cualquier código, que se ejecutará sólo cuando el DOM está listo para recibir comandos que lo modifiquen. Con ello, esta función nos vale perfectamente para hacer cualquier cosa dentro de la página que afecte al DOM.

Ejemplo:

```
$(function () {  
    //Aquí puedo hacer cualquier cosa con el DOM  
});
```

Este callback con la función jQuery `$()` sería una abreviatura de otro método que hemos visto repetidas veces a lo largo de este manual para definir acciones cuando el DOM está listo:

```
$(document).ready(function () {  
    //Aquí puedo hacer cualquier cosa con el DOM  
});
```

Incluso podemos hacer varios callback, para agregar distintas acciones a realizar cuando el DOM está listo, las veces que queramos, igual que cuando definíamos el evento `ready()` sobre el objeto `document`.

Podemos ver el código de una página que hace uso de la función dólar, pasando por parámetro una función.

```
<html>  
<head>  
    <title>función jquery</title>  
    <script src="../jquery-1.3.2.min.js" type="text/javascript"></script>  
<script>  
  
$(function () {  
    var documento = $("p");  
    documento.css("background-color", "#ff8833");  
});  
  
$(function () {  
    var documento = $("b");  
    documento.css("color", "#fff");  
});  
  
//si colocase aquí este código, no funcionaría, porque el DOM no estaría listo para realizar  
acciones  
//var documento = $("p");  
//documento.css("background-color", "#ff8833");  
  
</script>  
</head>  
<body>  
<p><b>Párrafo</b>!!</p>  
<p>Otro <b>párrafo</b></p>  
</body>  
</html>
```

Se puede [ver en marcha en una página aparte](#).

Hasta aquí hemos visto todas las posibilidades que existen para trabajar con la función jQuery. Realmente a lo largo de este manual ya la habíamos utilizado muchas veces y en lo sucesivo continuaremos usándola, ya que cualquier cosa que deseemos hacer en una página web va a depender en algún momento de invocar a `$()` en alguna de sus variantes.

Artículo por Miguel Angel Alvarez

2.4.- Core/each: each del core de jQuery

Conozcamos con detalle el método each del core de jQuery, para ejecutar una función en cada uno de los elementos de un grupo.

Hay algo que tenemos que tener en cuenta, en la medida de lo posible, cuando creamos código Javascript: crear un código de calidad y lo más corto posible. Para ello también nos facilitan mucho las cosas los frameworks y métodos como `each()`, que veremos en este artículo.

El método `each()` pertenece al juego de [funciones del CORE de jQuery](#), cuyas particularidades ya comenzamos a analizar en el [manual de jQuery](#). Se trata de un método para realizar acciones con todos los elementos que concuerdan con una selección realizada con la función jQuery -también llamada función `$()`-. Útil porque nos da una manera cómoda de iterar con elementos de la página y hacer cosas con ellos más o menos complejas de una manera rápida y sin utilizar mucho código para definir el bucle.

2.4.1.- Cómo funciona each

Each es un método que se utiliza sobre un conjunto de elementos que hayamos seleccionado con la función jQuery. Con `each` realizamos una iteración por todos los elementos del DOM que se hayan seleccionado.

El método `each` recibe una función que es la que se tiene que ejecutar para cada elemento y dentro de esa función con la variable "this" tenemos una referencia a ese elemento del DOM. Adicionalmente, la función que se envía a `each`, puede recibir un parámetro que es el índice actual sobre el que se está iterando.

Quiero explicar las bondades de `each()` con un ejemplo. Por ejemplo, veamos esta línea de código:

```
$("#p").css("background-color", "#eee");
```

Como ya sabemos, con `$("#p")` seleccionamos todos los párrafos de la página. Luego con el método CSS asignamos un estilo, en este caso para cambiar el color del fondo. Esto en realidad jQuery lo hace con una iteración con todos los párrafos de la página, sin que tengamos que hacer nosotros nada más y es genial que se permita en el uso del framework. ¿Pero qué pasa si queremos cambiar el fondo de los párrafos utilizando colores alternos?

En este caso no podemos hacerlo en una sola línea de código, pero `each` nos vendrá como anillo al dedo.

Imaginemos que tenemos una serie de párrafos en la página y queremos cambiar el color de fondo a los mismos, de manera que tengan colores alternos, para hacer dinámicamente un típico diseño para los listados.

Entonces podríamos hacer lo siguiente:

```
$("#p").each(function(i) {
    if(i%2==0) {
        $(this).css("background-color", "#eee");
    } else {
        $(this).css("background-color", "#ccc");
    }
});
```

Con `$("#p")` tengo todos los párrafos. Ahora con `each` puedo recorrerlos uno a uno. Para cada uno ejecutaremos la función que enviamos como parámetro a `each()`. En esa función recibo como parámetro una variable "i" que contiene el índice actual sobre el que estoy iterando.

Con `if(i%2==0)` estoy viendo si el entero del índice "i" es par o impar. Siendo par coloco un color de fondo al elemento y siendo impar coloco otro color de fondo.

Como se puede ver, con la variable "this" tenemos acceso al elemento actual. Pero OJO, que este elemento no es un objeto jQuery, así que no podríamos enviarle métodos del framework jQuery hasta que no lo expandamos con la función jQuery. Así pues, tenemos que hacer `$(this)` para poder invocar al método `css()`. Por si no queda claro este punto mirar las diferencias entre estas dos líneas de código:

```
this.css("background-color", "#ccc");
$(this).css("background-color", "#ccc");
```

En la primera línea no estaríamos extendiendo la variable `this` con las funcionalidades de jQuery, luego no funcionaría.

En la segunda línea, que es la que habíamos utilizado en el script de ejemplo, sí estamos extendiendo la variable "this" con la función jQuery. De ese modo, se puede invocar a cualquier método de jQuery sobre los elementos.

Podemos [ver un ejemplo en marcha que hace uso de ese script](#).

Este sería su código fuente completo:

```
<html>
<head>
  <title>each del core de jQuery</title>
  <script src="../jquery-1.3.2.min.js" type="text/javascript"></script>
</script>
$(document).ready(function() {
  $("p").each(function(i) {
    if(i%2==0) {
      $(this).css("background-color", "#eee");
    }else{
      $(this).css("background-color", "#ccc");
    }
  });
});
</script>
</head>
<body>
<p>Primer párrafo</p>
<p>Otro párrafo</p>
<p>Tecer párrafo</p>
<p>Uno más</p>
<p>y acabo...</p>
</body>
</html>
```

2.4.2.- Retornando valores en la función que enviamos a each

Ahora vamos a ver un par de posibilidades interesantes al utilizar each. Resulta que la función que enviamos como parámetro a each() puede devolver valores y dependiendo de éstos, conseguir comportamientos parecidos a los conocidos break o continue de los bucles Javascript.

Si la función devuelve "false", se consigue detener por completo el proceso de iteraciones de each(). Esto es como si hiciéramos el típico "break".

Si la función devuelve "true", se consigue pasar directamente a la próxima iteración del bucle. Es como hacer el típico "continue".

Para ver estos dos casos realizaremos otro ejemplo de uso de each.

Tenemos varios DIV, donde cada uno tiene un texto.

```
<div>red</div>
<div>blue</div>
<div>red</div>
<div>white</div>
<div>red</div>
<div>green</div>
<div>orange</div>
<div>red</div>
<div>nada</div>
<div>red</div>
<div>blue</div>
```

Ahora queremos hacer un recorrido a esos DIV y en cada uno, mirar el texto que aparece. Entonces colocaremos como color del texto del DIV el color que aparece escrito en el DIV. Pero con dos casos especiales:

- Si el texto del DIV es "white", entonces no queremos hacer nada con ese elemento.
- Si el texto del DIV es "nada", entonces detendremos el bucle y dejaremos de colorear los siguientes elementos.

Esto lo podríamos hacer con el siguiente código:

```
$("#div").each(function(i) {
  elemento = $(this);
  if(elemento.html() == "white")
    return true;
  if(elemento.html() == "nada")
    return false;
```

```
elemento.css("color", elemento.html());  
});
```

Ahora podremos [ver este ejemplo en funcionamiento en una página aparte](#).

Artículo por *Miguel Ángel Álvarez*

2.5.- Método size() y propiedad length del core de jQuery

Dos formas de acceder al número de elementos que hay en un objeto jQuery, a través del método size() y la propiedad length.

Vamos a ver en este artículo cómo obtener el número de elementos que tiene el objeto jQuery. Como pudimos aprender previamente en nuestro [manual de jQuery](#), este framework Javascript tiene como base la llamada "función jQuery" que devuelve el objeto jQuery, en el que hay uno o varios elementos de la página, según el selector que se le haya pasado.

En un pasado artículo de DesarrolloWeb.com ya explicamos con detalle qué era la [función jQuery y cómo utilizarla](#). Pues bien, ahora veremos rápidamente cómo saber cuántos elementos encontramos y seleccionamos por medio de esta función, lo que puede ser útil por diversos motivos al hacer código Javascript.

2.5.1.- Método size() del Core de jQuery

Este método sirve, como decimos, para obtener el número de elementos seleccionados con la función jQuery. Simplemente devuelve el número de elementos que hay en el objeto, como un entero.

Por ejemplo, veamos este código:

```
var parrafos = $("p");  
alert ("Hay " + parrafos.size() + " párrafos en la página");
```

Con la primera línea selecciono todos los párrafos de la página y los meto en el objeto jQuery de la variable "parrafos". Mediante la segunda línea muestro el número de párrafos encontrados, con una llamada al método size().

No tiene más misterio, salvo que en jQuery existe otro modo de hacer esto, pero bastante más rápido.

Podemos [ver una página en marcha con este ejemplo de uso de size\(\)](#).

2.5.2.- Propiedad length del objeto jQuery

La propiedad length, que existe en cualquier objeto jQuery, nos sirve para obtener el número de elementos de la página que hemos seleccionado. Lo interesante de esta propiedad es que almacena directamente este valor, por lo que es más rápido y más aconsejable que calcular los elementos seleccionados con el método size().

Tanto el método size() con la propiedad length devolverán el mismo valor, siendo las únicas diferencias la mencionada rapidez adicional de la propiedad y el acceso a este valor, que como es una propiedad, se accede a través del operador punto, pero sin colocar los paréntesis después de length. Por ejemplo, veamos este código:

```
var ElementosMitexto = $(".mitexto");  
alert ("Hay " + ElementosMitexto.length + " elementos de la clase mitexto");
```

Con la primera línea estamos utilizando la función jQuery para seleccionar todos los elementos de la página que tienen la clase CSS "mitexto". Con la segunda línea se muestra en una caja de alerta el número de elementos seleccionados con ElementosMitexto.length.

Podemos ver el código completo de una página que hace uso de este método:

```
<html>  
<head>  
  <title>propiedad length del core jQuery</title>  
  <script src="../jquery-1.3.2.min.js" type="text/javascript"></script>
```



```
<script>
$(document).ready(function(){
  //selecciono todos los elementos de la clase "mitexto"
  var ElementosMitexto = $(".mitexto");
  //muestro el número de los párrafos encontrados
  alert ("Hay " + ElementosMitexto.length + " elementos de la clase mitexto");
});
</script>
</head>

<body>

<p>Esto es un párrafo normal</p>
<p class="mitexto">Esto es un párrafo de la clase "mitexto"</p>
<div>Un div normal</div>
<div class="mitexto">Ahora un div de la clase "mitexto"</div>
</body>
</html>
```

Para acabar, dejamos el enlace a una página donde se puede ver el [ejemplo de la propiedad length del objeto jQuery en funcionamiento](#).

Artículo por *Miguel Ángel Álvarez*

2.6.- Método data() Core jQuery

El método `data()` del core de jQuery sirve para almacenar información en los elementos de la página, en pares nombre de variable y valor. Veremos también el método `removeData()` para eliminar datos almacenados.

Seguimos viendo componentes interesantes del "Core" de jQuery, donde están las clases y métodos más básicos de este framework Javascript. En esta entrega del [manual de jQuery](#) de DesarrolloWeb.com, explicaremos el uso del método `data()` y `removeData()`, que sirven para almacenar, consultar y eliminar cualquier tipo de dato en elementos de la página.

En algunas ocasiones resulta útil almacenar variables u objetos en determinados elementos de la página. Aunque quizás no es una acción muy corriente en los primeros pasos con jQuery, en el futuro encontraréis que resulta útil y veréis herramientas y plugins que utilizan este mecanismo para su operativa. De modo que conviene al menos saber que esto es posible y conocer de qué manera podemos utilizar los elementos de la página para guardar cosas en ellos.

Para ello vamos a comentar dos métodos distintos que forman parte del core de jQuery:

2.6.1.- Método data()

Este método del objeto jQuery sirve tanto para guardar un dato en un elemento como para consultarlo. Según el número de parámetros que reciba, realiza una u otra acción.

- Si recibe un parámetro `data(nombre)`: devuelve el valor que haya en el dato cuyo nombre se pasa por parámetro.
- Si recibe dos parámetros `data(nombre, valor)`: almacena un dato, cuyo nombre recibe en el primer parámetro, con el valor que recibe en el segundo parámetro.

Como `data()` es un método que pertenece al objeto jQuery, podemos almacenar estos pares (dato, valor) en cualquiera de los elementos que seleccionemos con la [función jQuery\(\)](#).

Veamos un caso de uso simple. Por ejemplo tenemos un elemento de la página como este:

```
<div id="capa">
En esta división (elemento id="capa") voy a guardar y leer datos sobre este elemento.
</div>
```

Ahora podríamos usar le método `data()` de la siguiente manera:

```
$("#capa").data("midato", "mivalor");
```

Con esta línea hemos guardado un dato llamado "midato" con el valor "mivalor", en el elemento con identificador (atributo id) "capa".

Ahora podríamos leer ese dato en cualquier momento para acceder a su valor, de la siguiente manera:

```
alert($("#capa").data("midato"));
```

En esta línea de código extraemos el dato "midato" del elemento con identificador "capa" y lo mostramos en una caja de alerta.

Podemos ver una [página en marcha que hace uso de esas dos funciones](#).

2.6.2.- Método removeData()

Este método sirve para eliminar un dato de un elemento y su funcionamiento es tan simple como enviar por parámetro el dato que se quiere eliminar del elemento.

```
$("#capa").removeData("midato")
```

Con esta línea habríamos eliminado el dato llamado "midato" del elemento con identificador "capa".

2.6.3.- Ejemplo completo de los métodos data() y removeData() del Core de jQuery

Veamos un ejemplo completo del uso de estos métodos que acabamos de aprender. Se trata de una página que tiene un elemento sobre el que vamos a guardar datos. Además tiene tres botones para guardar un dato, leerlo y borrarlo. El dato que se guardará tendrá como valor lo que se haya escrito en un campo de texto que aparece también en la página.

Podemos ver el [ejemplo en marcha en una página aparte](#).

Tenemos, para comenzar, un elemento de la página, que es donde vamos a guardar los pares dato-valor con data().

```
<div id="division">
En esta división (elemento id="division") voy a guardar datos con la función data y luego los voy a leer.
</div>
```

Luego tendremos este formulario, que contiene el campo de texto así como los tres botones de los que hemos hablado.

```
<form name="formul">
Escribe un valor a guardar, leer o eliminar:
<input type="text" name="valor" id="valor">
<br>
<input type="button" value="guardar dato" id="guardar">
<input type="button" value="leer dato" id="leer">
<input type="button" value="eliminar dato" id="eliminar">
</form>
```

Ahora se trata de asignar los comportamientos a estos botones con Javascript, haciendo uso de jQuery.

Este sería el script para agregar el evento click al botón de guardar datos.

```
$("#guardar").click(function(evento) {
    var valor = document.formul.valor.value;
    //Esta misma línea de código se puede codificar así también con jQuery
    //var valor = $("#valor").attr("value");
    $("#division").data("midato", valor);
    $("#division").html('He guardado en este elemento (id="division") un dato llamado "midato" con el valor "' + valor + '"');
});
```

Como se puede ver, primero se recibe el texto del campo de texto que había en el formulario. Para ello se muestran dos maneras de hacerlo:

- A través de la jerarquía de objetos del navegador, con document.formul.valor.value
- A través de su identificador, con un método de jQuery llamado attr() que sirve para recuperar el valor de un atributo HTML pasado por parámetro sobre el elemento que recibe el método. Este modo de obtener el atributo

value del campo de texto está comentado, pues sólo lo quería señalar, para que se vea el modo de acceder a un elemento de formulario utilizando las funciones del framework Javascript jQuery.

Luego, se guarda el dato "midato" con el valor que se recuperó del atributo value del campo de texto. Para ello utilizamos el método `data()` tal como comentábamos.

Por último se muestra un mensaje en el HTML del elemento con `id="division"`, por medio del método `html()` de jQuery, para informar sobre la acción que acabamos de realizar.

Ahora mostramos el código para asignar un comportamiento al evento click sobre el segundo botón:

```
$("#leer").click(function(evento){
    valor = $("#division").data("midato");
    $("#division").html('En este elemento (id="division") leo un dato llamado "midato" con el valor ' + valor + '');
});
```

Como se puede ver, se recupera el valor del dato "midato" guardado sobre el elemento "#division" (etiqueta HTML con `id="division"`), y se almacena en una variable. Luego se crea un mensaje para mostrar el valor del dato.

Para acabar, tenemos el código del evento click sobre el botón de eliminar el contenido de un dato, que hace uso de `removeData()`.

```
$("#eliminar").click(function(evento){
    $("#division").removeData("midato");
    $("#division").html('Acabo de eliminar del elemento (id="division") el dato llamado "midato"');
});
```

Como se verá, el método `removeData()` se invoca sobre el elemento que tiene el dato que pretendemos eliminar. Más tarde se muestra un mensaje informando sobre la acción que se ha realizado.

Para comprobar el funcionamiento de estos métodos habría que crear un dato, escribiendo el valor en el campo de texto y pulsando el botón "guardar dato". Luego podríamos leer ese dato con el botón "leer dato". Por último podríamos eliminar el dato con el botón "eliminar dato". Si, una vez eliminado pulsamos sobre el botón de "leer dato" veremos que el valor del dato aparece como "undefined", puesto que ese dato ha sido borrado (esto también ocurre si no se ha guardado ningún dato todavía, por ejemplo cuando se acaba de cargar la página).

Sería interesante ver el código fuente completo de esta página, para hacernos una idea más exacta de cómo se integrarían todos estos elementos. <html>

```
<head>
  <title>Ejemplos de uso de la función data del core de jQuery</title>
  <script src="../../jquery-1.3.2.min.js" type="text/javascript"></script>
</script>
$(document).ready(function(){

    $("#guardar").click(function(evento){
        var valor = document.formul.valor.value;
        //Esta misma línea de código se puede codificar así también con jQuery
        //var valor = $("#valor").attr("value");
        $("#division").data("midato",valor);
        $("#division").html('He guardado en este elemento (id="division") un dato llamado "midato" con el valor ' + valor + '');
    });

    $("#leer").click(function(evento){
        valor = $("#division").data("midato");
        $("#division").html('En este elemento (id="division") leo un dato llamado "midato" con el valor ' + valor + '');
    });

    $("#eliminar").click(function(evento){
        $("#division").removeData("midato");
        $("#division").html('Acabo de eliminar del elemento (id="division") el dato llamado "midato"');
    });
});
</script>
</head>
```

```
<body>

<div id="division">
En esta división (elemento id="division") voy a guardar datos con la función data y luego los voy a leer
</div>
<br>
<form name="formul">
Escribe un valor a guardar, leer o eliminar:
<input type="text" name="valor" id="valor">
<br>
<input type="button" value="guardar dato" id="guardar">
<input type="button" value="leer dato" id="leer">
<input type="button" value="eliminar dato" id="eliminar">
</form>

</body>
</html>
```

De nuevo, dejamos el [enlace al ejemplo en marcha](#).

Para seguir os indicamos la lectura del siguiente artículo de este manual, donde puedes obtener [explicaciones adicionales y ejemplos de uso de estos métodos data\(\) y removeData\(\)](#).

Artículo por *Miguel Angel Alvarez*

2.7.- Consideraciones interesantes de data() y removeData()

Ahora veremos algunos puntos interesantes y nuevos ejemplos sobre el funcionamiento de los métodos data() y removeData() de jQuery.

Existen algunos puntos que debemos conocer sobre el funcionamiento de estos métodos que no hemos explicado todavía en el artículo anterior, en el que se [comenzó a tratar acerca de data\(\) y removeData\(\)](#). Veamos a continuación una serie de consideraciones:

Admite cualquier tipo de dato: Podemos guardar lo que deseemos por medio del método data(). Los [ejemplos anteriores](#) hemos guardado simplemente cadenas de texto, pero soportaría cualquier tipo de variable, numérica, un array o incluso un objeto Javascript o jQuery.

Se guarda un dato por cada elemento del objeto jQuery seleccionado: En caso que en el objeto jQuery sobre el que estemos almacenando cosas con data() haya referencias a varios elementos de la página, el dato se almacena en todos los elementos. (recordemos que, según lo explicado anteriormente en [desarrolloweb.com](#), un objeto jQuery puede tener seleccionados varios elementos de la página, como todos los enlaces presentes, los elementos de una determinada clase CSS, etc. dependiendo del selector escogido al hacer uso de la función dólar)

Los objetos se almacenan por referencia: En el caso que estemos almacenando un objeto Javascript con data() sobre uno o varios elementos, no se copia el objeto, sino que se asigna por referencia. Esto quiere decir que no se harían copias independientes del objeto a guardar, sino que permanecería tal cual y lo que se asignaría como dato es una referencia a ese único objeto.

Ahora, para investigar un poco sobre estas posibilidades, hemos creado un par de ejemplos un poco más complejos que hacen uso de los métodos data() y removeData(). Son ejemplos más avanzados, que hacen uso de algunas cosas que no hemos explicado todavía en este [manual de jQuery](#). No obstante, vendrá bien verlos para aprender algunos usos de estas funcionalidades.

Para empezar, quiero mostrar una página de ejemplo donde existen tres enlaces y dos botones. Al pulsar cualquiera de los enlaces mostraremos el contenido de un dato almacenado en ellos con data(). Los botones, por su parte, servirán para almacenar contenidos en datos sobre esos enlaces. Además tendremos una capa con id="mensaje" que nos servirá para mostrar cosas por pantalla.

Podemos [ver el ejemplo en marcha en una página aparte](#).

El código de los elementos HTML será el siguiente:

```
<a href="#" id="enlace1">Enlace 1</a>
<br>
<a href="#" id="enlace2">Enlace 2</a>
<br>
<a href="#" id="enlace3">Enlace 3</a>
<br>
<br>
<div id="mensaje">
Mensaje...
</div>
<br>
<button id="guardar">guardar "midato" con valor "mivalor" en todos los enlaces</button>
<br>
<button id="guardarenlace1">guardar "midato" con valor "otro valor" en el enlace 1</button>
```

Ahora veamos cómo aplicar eventos a los elementos de la página, para almacenar datos y mostrarlos.

Comencemos por el código de los eventos de los botones.

```
$("#guardar").click(function(evento){
    $("a").data("midato", "mivalor");
    $("#mensaje").html('He guardado en todos los enlaces un dato llamado "midato" con el valor "mivalor"');
});
```

Con este código estamos almacenando datos en todos los enlaces. Cabe fijarse que con la función jQuery \$("a") obtenemos un objeto jQuery donde están todos los enlaces de la página. Luego, al invocar data() sobre ese objeto, estamos almacenando ese dato en todos los enlaces existentes.

```
$("#guardarenlace1").click(function(evento){
    $("#enlace1").data("midato", "otro valor");
    $("#mensaje").html('He guardado en el enlace1 un dato llamado "midato" con el valor "otro valor"');
});
```

En este otro código del evento click para el segundo botón, almacenamos "otro valor" sobre el dato de antes, pero sólo lo hacemos sobre el enlace 1, dado que hemos utilizado el selector \$("#enlace1"), con el identificador único del primer enlace.

Y ahora podríamos ver el código para asignar un evento a todos los enlaces, para que al pulsarlos nos muestre lo que haya en el dato almacenado con data(), si es que hay algo.

```
$("a").click(function(evento){
    evento.preventDefault();
    valorAlmacenado = $(this).data("midato");
    $("#mensaje").html("En el enlace <b>" + $(this).attr("id") + "</b> tiene el dato 'midato' como "
+ valorAlmacenado);
});
```

Como se puede ver, estamos creando un evento click, pero lo estamos haciendo sobre los tres enlaces que hay en la página a la vez, dado el selector utilizado en la función jQuery \$("a"). Luego el código del evento será el mismo para los tres enlaces.

Lo primero que se hace es un evento.preventDefault() que permite que el enlace no tenga el comportamiento típico (ir a la URL del href). A continuación hacemos:

```
valorAlmacenado = $(this).data("midato");
```

Como se puede ver, se está extrayendo el valor almacenado en el enlace actual, que recibe el evento. Con \$(this) obtenemos el objeto jQuery del elemento que ha recibido el evento, que es el enlace sobre el que se ha pulsado y no todos los enlaces. Con el método data("midato"), invocado sobre \$(this), obtenemos el valor del dato "midato" almacenado en el enlace que fue pulsado solamente.

Luego se muestra un mensaje para indicar el valor que había en el dato. Pero claro, este código, como es común para todos los enlaces, tiene que acceder también a \$(this) para saber qué enlace en concreto fue el que se pulsó. Para identificar el enlace se hace \$(this).attr("id"), que devuelve el atributo "id" del enlace sobre el que se hizo clic.

A continuación se puede ver el código completo de esta página.

```
<html>
```

```
<head>
  <title>Ejemplos de uso de la función data del core de jQuery</title>
  <script src="../../jquery-1.3.2.min.js" type="text/javascript"></script>
</script>
$(document).ready(function() {

  $("#a").click(function(evento){
    evento.preventDefault();
    valorAlmacenado = $(this).data("midato");
    $("#mensaje").html("En el enlace <b>" + $(this).attr("id") + "</b> tiene el dato 'midato' como " + valorAlmacenado);
  });

  $("#guardar").click(function(evento){
    $("#a").data("midato", "mivalor");
    $("#mensaje").html("He guardado en todos los enlaces un dato llamado "midato" con el valor "mivalor");
  });

  $("#guardarenlace1").click(function(evento){
    $("#enlace1").data("midato", "otro valor");
    $("#mensaje").html("He guardado en el enlace1 un dato llamado "midato" con el valor "otro valor");
  });
});
</script>
</head>

<body>

<a href="#" id="enlace1">Enlace 1</a>
<br>
<a href="#" id="enlace2">Enlace 2</a>
<br>
<a href="#" id="enlace3">Enlace 3</a>
<br>
<br>
<div id="mensaje">
Mensaje...
</div>
<br>
<button id="guardar">guardar "midato" con valor "mivalor" en todos los enlaces</button>
<br>
<button id="guardarenlace1">guardar "midato" con valor "otro valor" en el enlace 1</button>

</body>
</html>
```

Si se desea, se puede [ver el ejemplo en marcha](#) en una página aparte. ht

2.7.1.- Datos de tipo objeto asignados por referencia con data()

Sobre el punto que comentábamos antes, sobre los objetos Javascript que se asignan por medio de data(), que siempre se hace por referencia, hemos creado otro ejemplo, del que simplemente vamos a colocar un enlace para verlo en funcionamiento y su código. <http://www.desarrolloweb.com/articulos/ejemplos/jquery/core/data3.html>

El ejemplo es bastante similar al anterior, con la salvedad que se ha creado un par de acciones adicionales para almacenar en los elementos variables de tipo objeto.

Luego, al operar sobre esos datos de tipo objeto, comprobamos que en realidad sólo existe un objeto compartido por todos los elementos a los que fue asignado. Es decir, no se hicieron copias del objeto, sino que se asignaron en los datos simplemente su referencia.

Puede verse este [ejemplo en marcha en una página aparte](#).

El código completo se puede ver a continuación.

```
<html>
```

```
<head>
  <title>Ejemplos de uso de la función data del core de jQuery</title>
  <script src="../jquery-1.3.2.min.js" type="text/javascript"></script>
</script>
$(document).ready(function() {

  $("a.enlacealmacenar").click(function(evento) {
    evento.preventDefault();
    var valorAlmacenado = $(this).data("midato");
    var mensaje = "En el enlace <b>" + $(this).attr("id") + "</b> tiene el dato 'midato' como " +
valorAlmacenado;
    var valorAlmacenado2 = $(this).data("miobjeto");
    mensaje += "<br>Además, he leído un dato llamado 'miobjeto' con valor " + valorAlmacenado2;
    $("#mensaje").html(mensaje);
  });

  $("#guardar").click(function(evento) {
    evento.preventDefault();
    $("a").data("midato", "mivalor");
    $("#mensaje").html('He guardado en todos los enlaces un dato llamado "midato" con el valor
"mivalor"');
  });

  $("#guardarenlace1").click(function(evento) {
    evento.preventDefault();
    $("#enlace1").data("midato", "otro valor");
    $("#mensaje").html('He guardado en el enlace1 un dato llamado "midato" con el valor "otro
valor"');
  });

  $("#guardarobjeto").click(function(evento) {
    evento.preventDefault();
    $("a").data("miobjeto", $("#capapruebas"));
    $("#mensaje").html('He guardado todos los enlaces un dato llamado "miobjeto" con el valor un
objeto que es el objeto jquery de seleccionar la capa con id "capapruebas"');
  });

  $("#operarobjetoenlace1").click(function(evento) {
    evento.preventDefault();
    $("#enlace1").data("miobjeto").html("cambio el html del objeto que hay en el dato 'miobjeto'
del 'enlace1'");
  });

  $("#operarobjetoenlace2").click(function(evento) {
    evento.preventDefault();
    $("#mensaje").html("Este es el HTML que hay en el objeto asociado a enlace2 en el dato
'miobjeto':<br>" + $("#enlace2").data("miobjeto").html());
  });

});
</script>
</head>

<body>

<a href="#" id="enlace1" class="enlacealmacenar">Enlace 1</a>
<br>
<a href="#" id="enlace2" class="enlacealmacenar">Enlace 2</a>
<br>
<a href="#" id="enlace3" class="enlacealmacenar">Enlace 3</a>
<br>
<br>
<div id="mensaje">
Mensaje...
</div>
<br>

<ol style="line-height: 200%;">
<li>
<a id="guardar" href="#">guardar "midato" con valor "mivalor" en todos los enlaces</a>

```

```
</li>
<li>
<a id="guardarenlace1" href="#">guardar "midato" con valor "otro valor" en el enlace 1</a>
</li>
<li>
<a id="guardarobjeto" href="#">guardar "miobjeto" con una referencia a la capa de pruebas</a>
</li>
<li style="line-height: 100%;">
<a id="operarobjetoenlace1" href="#">Recuperar un objeto del enlace1 para hacer cosas con él
<SPAN style="font-size: 8pt; font-weight: bold">
PULSAR ESTE ENLACE SÓLO DESPUÉS DE HABER ALMACENADO EL OBJETO EN LOS ENLACES POR MEDIO DEL ENLACE DE
ESTA LISTA MARCADO COMO 3)
</SPAN>
</a></li>
<li style="line-height: 100%;">
<a id="operarobjetoenlace2" href="#">Recuperar un objeto del enlace2 para hacer cosas con él
<SPAN style="font-size: 8pt; font-weight: bold">
PULSAR ESTE ENLACE SÓLO DESPUÉS DE HABER ALMACENADO EL OBJETO EN LOS ENLACES POR MEDIO DEL ENLACE DE
ESTA LISTA MARCADO COMO 3)
</SPAN>
</a></li>
</ol>
<br>
<br>

<div id="capapruebas">
Este es el texto de una capa de pruebas... con id="capapruebas"
</div>

</body>
</html>
```

Hemos visto diversos ejemplos de uso de `data()` y `removeData()`, métodos básicos de jQuery. Puede que ahora no se les encuentre mucha utilidad, pero nos servirán para resolver problemas futuros y entender cómo funcionan diversos plugins o componentes más avanzados de jQuery.

Por lo que respecta al Core de jQuery, ya hemos visto diversas funcionalidades en desarrolloweb.com en artículos de este [manual](#). Por ahora lo vamos a dejar por aquí, aunque hay diversos métodos del Core que no hemos llegado a ver. En los próximos artículos pasaremos página y comenzaremos a ver otros temas interesantes que nos permitirán explotar un poco más nuestra creatividad, poniendo en marcha utilidades más cercanas a lo que pueden ser nuestras necesidades del día a día.

Artículo por *Miguel Angel Alvarez*

Parte 3:

Selectores de jQuery

Comenzamos a analizar en profundidad las diferentes maneras que tenemos en jQuery de seleccionar conjuntos de elementos de la página, a través de distintos tipos de selectores.

3.1.- Selectores en jQuery

Los selectores sirven para seleccionar elementos de la página a partir de una cadena de texto que le pasamos a la función jQuery.

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. Hasta en los ejemplos más básicos del [Manual de jQuery](#) se tienen que utilizar selectores para acceder a los elementos que deseamos alterar, así que inevitablemente, si has leído este manual hasta este artículo, los habrás utilizado ya.

En mi opinión, una de las cosas que más potentes de jQuery son los selectores, al menos comparando este framework Javascript con otros que conozco. Veremos en este artículo cómo utilizarlos y aprovecharnos de su potencia.

Para empezar, veamos un selector, para aclarar las ideas y refrescar la memoria. Cuando utilizamos la [función jQuery \(o función dólar\)](#), 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.

3.1.1.- 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. Como entiendo que todas las personas que intenten profundizar en el framework jQuery deben haber conocido CSS anteriormente, no habrá ningún problema con ellos.

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
```

3.1.2.- Conclusión sobre los selectores

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 y que dejamos para próximos artículos.

Nota: Si todavía no has quedado muy claro, podéis pasaros por el [Videotutorial sobre los selectores en jQuery](#)

Ahora, os recomendamos seguir el aprendizaje con el siguiente artículo, en el que pondremos en práctica los selectores que hemos conocido hasta el momento: [Ejemplo para practicar con selectores de jQuery](#).

Artículo por *Miguel Angel Alvarez*

3.2.- Ejemplo para practicar con selectores en jQuery

Ejemplo de página que nos permitirá hacer prácticas con los selectores de jQuery.

En el artículo anterior explicamos lo que eran los [selectores de jQuery](#) y los tipos de selectores básicos. Ahora, para que podamos ver por la práctica cómo funcionan cada uno de los selectores, hemos creado un ejemplo donde podremos escribir varios selectores y ver cómo funcionan, es decir, qué elementos de la página se consigue seleccionar con cada uno.

En este ejemplo tenemos una página que tiene varias etiquetas y un formulario. En el formulario hay un campo de texto y un botón. En el campo de texto podemos escribir cualquier selector y pulsando luego el botón, mediante jQuery, hacemos que parpadeen los elementos que concuerdan con ese selector.

El ejemplo puede [verse en una página aparte](#).

Aclaración: Por cierto, comento una cosa que resulta para la mayoría debe resultar obvia, pero quizás alguien pueda cometer el error. En los ejemplos del [artículo anterior](#), escribíamos los selectores entre comillas, porque un selector es una cadena de caracteres. Pero en este caso, en la [página del ejemplo](#), en el campo de texto hay que escribir los selectores sin las comillas. Si ponemos las comillas en realidad sería como intentar hacer un selector que incluyese el carácter comillas ". Esto es porque en el propio campo de texto cualquier cosa que escribamos ya es una cadena de caracteres de por sí.

Para hacer este ejemplo tenemos que utilizar varios métodos y funciones jQuery de los cuales, casi todos, ya hemos hablado a lo largo del manual.

Veamos el formulario que hemos creado en la página:

```
<form>
Selector: <input type="Text" name="camposelector" id="camposelector">
<input type="button" id="boton" value="Ver qué elementos seleccionas">
</form>
```

Como se puede ver, tiene un campo INPUT de texto al que le hemos puesto un identificador para referirnos a él mediante jQuery. Fijarse también el INPUT para hacer un botón, al que también le pusimos un identificador.

Ahora veamos el código Javascript empleado:

```
$(document).ready(function() {
    $("#boton").click(function(evento) {
        var selectorEscrito = $("#camposelector").attr("value");
        if (selectorEscrito=="") {
            alert("Escribe algo en el campo de texto")
        } else {
            elementosSeleccionados = $(selectorEscrito);
            elementosSeleccionados.fadeOut("slow", function() {
                elementosSeleccionados.fadeIn("slow");
            });
        }
    });
});
```

Con `document.ready()` indicamos una función a invocar cuando la página está lista para recibir acciones de programación que modifiquen su estructura.

Con `$("#boton").click()` indicamos una función a ejecutar cuando se hace clic sobre el botón.

```
var selectorEscrito = $("#camposelector").attr("value");
```

Nos sirve para acceder al atributo `value` del campo de texto, es decir, a lo que haya escrito dentro.

Si no hay nada escrito en el campo, muestro un mensaje de alerta, porque en este caso el selector cadena vacía no sería válido y recibiríamos un mensaje de error.

Si había algo en el campo, pues selecciono con jQuery los elementos de la página que corresponden con el selector escrito en el campo de texto. Eso se hace con la línea:

```
elementosSeleccionados = $(selectorEscrito);
```

Luego, sobre el elemento o elementos seleccionados, invoco el método `fadeOut()`, que sirve para ocultar elementos de la página. A `fadeOut()` le paso dos parámetros, uno es la velocidad con la que tiene que hacer el efecto y otro es una función callback, a ejecutar sólo en el momento que el efecto haya concluido. Eso es con la línea:

```
elementosSeleccionados.fadeOut("slow", function() {
```

Por último, en la función callback realizamos una llamada al método `fadeIn()` sobre el mismo objeto jQuery resultado de aplicar el selector anterior, que sirve para que los elementos ocultos se muestren de nuevo en la página. Esto último con la línea:

```
elementosSeleccionados.fadeIn("slow");
```

En resumen, ocultando y mostrando luego los elementos de vuelta conseguimos ese parpadeo. Si nos resulta extraño este código, recordamos que en el [Manual de jQuery](#) de desarrolloweb.com ya hemos publicado varios artículos que aclaran los

puntos tratados en este ejemplo, como los [efectos rápidos](#) o las [funciones callback](#).

3.2.1.- Código completo del ejemplo de selectores

Escribimos aquí para acabar el código completo de este ejemplo de trabajo con selectores.

```
<html>
<head>
  <title>Título de la página</title>
<style type="text/css">
.rojo{
  color: #cc0000;
}
.verde{
  color: #00cc00;
}
.azul{
  color: #0000cc;
}
.fondogris{
  background-color: #cccccc;
}
body{
  font-family: verdana, arial, helvetica;
}
div{
  margin-bottom: 4px;
}
</style>

<script src="../../jquery-1.3.2.min.js" type="text/javascript"></script>
<script>
$(document).ready(function() {
  $("#boton").click(function(evento){
    var selectorEscrito = $("#camposelector").attr("value");
    if (selectorEscrito==""){
      alert("Escribe algo en el campo de texto")
    }else{
      elementosSeleccionados = $(selectorEscrito);
      elementosSeleccionados.fadeOut("slow", function(){
        elementosSeleccionados.fadeIn("slow");
      });
    }
  });
});
</script>
</head>

<body>
<h1>Selectores en jQuery</h1>
<p>En esta página hay varias etiquetas. Ahora con este formulario puedes escribir un selector, para seleccionar algunas con jQuery, y luego pulsar el botón para ver qué elementos de la página has seleccionado.</p>
<form>
Selector: <input type="Text" name="camposelector" id="camposelector">
<input type="button" id="boton" value="Ver qué elementos seleccionas">
</form>

<p id="p1" class="rojo">Este es un párrafo con id="p1" y class="rojo"</p>

<p id="p2" class="verde">Este es un párrafo con id="p2" y class="verde" y aquí meto una itálica</p>

<p id="p3" class="rojo fondogris">Este es un párrafo con id="p3" y class="rojo fondogris" (es decir, este elemento tiene aplicadas las clases "rojo" y "fondogris"</p>

<p id="p4">Este es un párrafo con id="p4", sin class</p>

<p>Este es un párrafo con sin id ni class</p>
```

```
<div id="div1">Esto es una división con id="div1"</div>

<div id="div2" class="rojo">Esto es una división con id="div2" y class="rojo" y aqui <b>meto una
negrita</b></div>

<div id="div3" class="verde fondogris">Esto es una división con id="div3" y class="verde
fondogris"</div>

<div>Esto es una división sin id ni class</div>

<div class="azul">Esto es una división sin id, con class="azul"</div>

<b>Esto es una etiqueta b</b>

<i>Esto es una etiqueta i</i>
</body>
</html>
```

Dejamos de nuevo el [enlace para ver este ejemplo en marcha](#) y practicar con los selectores de jQuery.

Artículo por *Miguel Ángel Álvarez*

3.3.- Selectores de Jerarquía en jQuery

Selectores que sirven para seleccionar elementos atendiendo a la estructura o jerarquía de las etiquetas de la página.

En los últimos artículos del [Manual de jQuery](#) hemos hablado sobre los selectores. Como ya dijimos, sirven para seleccionar elementos de la página con los que queremos trabajar desde Javascript por medio del framework. En concreto vimos los [selectores básicos](#), con los que podremos resolver la mayoría de nuestras necesidades en cuanto a selección de elementos.

No obstante, 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. Vamos a ver en este artículo qué son los selectores de jerarquía y algunos ejemplos de uso.

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 ancestro 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.parrafo rojo i") //selecciona todas las etiquetas I que hay dentro de los párrafos con clase
"parrafo rojo".
$("#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"
```

Nota: la diferencia entre "ancestor descendant" y "parent > child" es que este último sólo selecciona los hijos directos. Por ejemplo, en el HTML siguiente:

```
<p><b>Párrafo</b> que tiene alguna <b>negrita</b> e <span
class="algo"><i>itálica</i></span> para seleccionar</p>
```

\$("p > b") seleccionaría los mismos elementos que \$("p b"), porque en este caso todas las etiquetas B son hijas directas de P.

Pero en el caso de la itálica (etiqueta I), que está metida dentro del párrafo, pero dentro también de un span, \$("p i") seleccionaría la etiqueta I por ser descendiente de P, pero \$("p > i") no seleccionaría la etiqueta I, por no ser hija directa de P.

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.parraforrojo + p" ) //Esto selecciona los párrafos que están después de cualquier párrafo que
tenga la clase "parraforrojo"
$( "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".
```

3.3.1.- Probando los selectores jQuery de Jerarquía

Hemos hecho un rápido script que prueba los selectores de jerarquía que están disponibles en jQuery. Es una simple página que tiene una serie de elementos y un script para seleccionar y alterar su estilo. Los elementos los vamos seleccionando con diversos tipos de selectores de Jerarquía que hemos visto en este artículo de DesarrolloWeb.com. El ejemplo tendría el siguiente código: <html>

```
<head>
  <title>Probando </title>
  <script src="../jquery-1.3.2.min.js" type="text/javascript"></script>
<script>
$(document).ready(function(){
  //selectores ancestor descendant
  $( "p i" ).css("color", "#66F");
  $( "table.mitabla td" ).css("background-color", "#55ff00");

  //selectores parent > child
  $( "p.parraforrojo > b" ).css("color", "red");
  $( ".mitd > *" ).css("border", "1px solid #ff9900");

  //selectores prev + next
  $( "i + b" ).css("font-size", "40px");
  $( ".elemlista + li" ).css("opacity", 0.2);

  //selectores prev ~ siblings
  $( "#a2 ~ div.clase" ).css("font-size", "180%");
  $( "#miparrafo ~ table" ).css("border", "3px dotted #dd6600");
});
</script>
</head>

<body>

<p class="parraforrojo">
<i>Hola</i> <b>esto</b> es un <b>párrafo</b> rojo <i>donde</i> he <b>puesto</b> unas <b>negritas</b>
```

```
</p>
<p class="parrafo rojo">Otro <b>con</b> clase class="parrafo rojo" <span class="unspan"><b>(esto está
dentro de unspan B, no depende directamente -no child- del párrafo)</b></span></p>
<p>Hola <b>esto</b> es otro <b>párrafo </b>para <i>poner</i> otras <b>negritas</b></p>
<p>hola!!!</p>
<table border=1>
<tr>
  <td><i>Tabla cualquiera</i></td>
  <td>Esta tabla <b>no tiene</b> class de <b>CSS</b></td>
</tr>
</table>
<p id="miparrafo">Este es el párrafo con id="miparrafo"</p>
<table class="mitabla" border=1>
<tr>
  <td colspan=2>Esta tabla tiene una <b>clase CSS</b></td>
</tr>
<tr>
  <td><i>class="mitabla"</i></td>
  <td class="mitd">Y este <b>td</b> le he puesto <i>class="mitd"</i> <span>Una cosa<span>otra
cosa</span></span></td>
</tr>
</table>
<p><b>Párrafo</b> que tiene alguna <b>negrita</b> e <span class="algo"><i>itálica</i></span> para
seleccionar</p>

<div>
<div id="a1">hola</div> <div id="a2">dos</div> <div id="a3">3</div> <span>Cuatro (no es un div)??</span>
</div> <div id="a4" class="clase">Cuatro de verdad</div>
</div>

<ul>
<li>Elem 1</li>
<li class="elem lista">Elem 2</li>
<li>Elem 3</li>
<li>Elem 4</li>
<li class="elem lista">Elem 5</li>
<li class="elem lista">Elem 6</li>
<li>Elem 7</li>
</ul>
</body>
</html>
```

Podemos [ver el ejemplo en marcha en una página aparte](#).

Artículo por *Miguel Angel Alvarez*

Parte 4:

Métodos de Atributos en jQuery

Exploramos diferentes métodos que existen en este framework Javascript para acceder y modificar los atributos del DOM, es decir, de los objetos o elementos que forman parte de una página web.

4.1.- Acceder y modificar atributos HTML desde jQuery

En jQuery existe una función llamada attr() que sirve para recuperar y alterar atributos de los elementos de la página.

En este [Manual de jQuery](#) estamos recorriendo poco a poco la documentación del popular framework Javascript, para ofrecer a los lectores de DesarrolloWeb.com explicaciones detalladas de las clases y métodos disponibles. Le ha llegado el turno al método attr() que sirve para 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.

Pero antes de empezar, vale la pena comentar que la información que encontraréis en este artículo se complementa con el siguiente texto, en el que veremos otros usos de la función attr().

4.1.1.- Lectura de un atributo

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 el caso que invoquemos el método attr sobre un objeto jQuery que contenga varios elementos a la vez, attr() en este caso devolvería el valor del atributo del primero de los elementos que haya en el objeto jQuery. Además, en caso que el elemento no tenga definido ese atributo al que se pretenda acceder, devolvería undefined.

Veamos un ejemplo, también simple, pero un poco más elaborado. Tenemos varios enlaces en la página, con este código HTML:

```
<a href="http://www.elpais.com" title="Diario El País">El País</a>  
<br>  
<a href="http://www.mozilla.org" title="Fundación Mozilla">Mozilla Foundation</a>  
<br>  
<a href="http://es.openoffice.org/" title="Siute de programas de oficina">Open Office</a>
```

Si hacemos algo como esto:


```
$("#a").attr("title")
```

Obtendremos el valor del atributo title del primero de los enlaces. Como tenemos tres enlaces en la página, \$("#a") nos devolvería un objeto jQuery que contiene esos tres enlaces, pues recordar, que attr("title") devuelve el valor del atributo "title" del primero de los elementos del objeto jQuery. Ahora bien, si quisiéramos obtener el valor del atributo "title" de todos los elementos, tendríamos que hacer un recorrido a cada uno de los enlaces con el método each del core de jQuery <http://www.desarrolloweb.com/articulos/core-each-jquery.html>.

Veamos un ejemplo de una página completa que hace ese recorrido con each para recuperar todos los valores de los atributos title de los enlaces que haya en la página:

```
<html>
<head>
  <title>método attr</title>
  <script src="../../jquery-1.3.2.min.js" type="text/javascript"></script>
<script>
$(document).ready(function() {

  $("#a").each(function(i) {
    var titulo = $(this).attr("title");
    alert("Atributo title del enlace " + i + ": " + titulo);
  });

});
</script>
</head>

<body>
<a id="enlace1" href="http://www.elpais.com" title="Diario El País">El País</a>
<br>
<a href="http://www.mozilla.org" title="Fundación Mozilla">Mozilla Foundation</a>
<br>
<a href="http://es.openoffice.org/" title="Siute de programas de oficina">Open Office</a>
</body>
</html>
```

Podemos [ver el ejemplo en marcha en una página aparte](#).

4.1.2.- Modificar un atributo

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.

Si lo deseas, puedes [ver el ejemplo en marcha en una página aparte](#).

4.1.3.- Modificar varios valores de atributos a la vez

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.

A estas alturas ya debemos saber crear variables con notación objeto, pero voy a dejar un ejemplo para que se pueda ver perfectamente este uso del método.

Imaginar 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.desarrolloweb.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 Desarrollo Web y además se les creará un estilo CSS para que sean de color naranja.

Podemos [ver una página aparte con este ejemplo en marcha](#).

En el siguiente artículo veremos un [ejemplo más elaborado sobre la modificación de atributos de elementos a través de la función attr\(\)](#), en el que para obtener el valor del atributo a modificar utilizamos una función que pasamos también como parámetro a attr(). Además, para los interesados en aprender en vídeo puede accederse a las [funciones de Atributos en el Videotutorial de jQuery](#).

Artículo por Miguel Ángel Álvarez

4.2.- Método attr() de jQuery, otros usos y removeAttr()

Un uso adicional del método attr() de jQuery, para modificar atributos con el valor devuelto de una función y borrar atributos de elementos de la página con removeAttr().

En el artículo anterior del [Manual de jQuery](#) ya comenzamos a [explicar el método attr\(\)](#), que pertenece al paquete de funciones para modificación de atributos de cualquiera de los elementos de una página web. En esta ocasión nos detendremos un uso adicional del método attr(), que seguro nos resultarán útiles para mantener el control dinámico de los atributos de las etiquetas HTML, con sus correspondientes ejemplos. Este uso que nos faltaba por ver nos servirá cuando tenemos que asignar el valor de un atributo con la respuesta de una función Javascript.

Además veremos también en este artículo otro método relacionado que sirve para eliminar por completo un atributo de cualquier elemento de la página, el [método removeAttr\(\)](#).

4.2.1.- Asignar un valor de atributo procesado por una función

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();
});
```

Para que se asimile mejor el uso de jQuery en una página, mostramos el código completo de este ejemplo.

```
<html>
<head>
  <title>método attr</title>
  <script src="../../jquery-1.3.2.min.js" type="text/javascript"></script>
<script>
$(document).ready(function() {

  $('.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();
  });

});
```

```
</script>
</head>

<body>
<form>
<input type="text" class="fecha">
<input type="text" class="nofecha">
<input type="text" class="fecha">
</form>

</body>
</html>
```

Si se desea, se puede [ver en marcha el ejemplo en una página aparte](#).

4.2.2.- Eliminar un atributo de uno o varios elementos con removeAttr()

Para acabar 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.

Podemos ver el [ejemplo de removeAttr\(\) en marcha en una página aparte](#).

Artículo por Miguel Angel Alvarez

4.3.- Funciones CSS de jQuery para conocer el tamaño y posición de elementos

Seguimos viendo funciones CSS del framework Javascript jQuery, en este caso las que sirven para conocer el tamaño y posición de los elementos en la página.

Entre las clasificaciones de funciones jQuery que existen diversas que sirven para controlar los atributos de CSS de los elementos de la página, ya sea para acceder a los valores actuales de los atributos CSS o para alterarlos. En artículos anteriores del [Manual de jQuery](#) pudimos conocer varias de estas funciones, por ejemplo en el artículo de [Añadir y quitar clases CSS sobre elementos](#).

En este artículo vamos a ver otras de las funciones que pone a nuestra disposición jQuery para acceder a la posición de los elementos en la página y a sus dimensiones. Estas funciones, aunque estaría mejor llamarles métodos (ya que pertenecen al [objeto jQuery](#)), son meramente informativas, para saber dónde están posicionados los elementos dentro del documento y sus medidas internas y externas. Lo veremos con detalle en breve, pero antes quiero señalar para los despistados que si queremos alterar las propiedades CSS de un elemento de la página con jQuery recordemos que está disponible el método `css()`, que hemos visto anteriormente en repetidas ocasiones a lo largo de este manual, enviándole como primer parámetro el nombre del atributo CSS a alterar y como segundo parámetro el valor del mismo.

Ahora voy a dar un listado de los métodos nuevos que vamos a ver en este artículo, comenzando por los que sirven para conocer las dimensiones de un elemento.

Métodos `innerWidth()` e `innerHeight()`:

Reciben un objeto jQuery y devuelven las dimensiones internas del primer elemento que haya en dicho objeto jQuery, esto es, la anchura y altura respectivamente del elemento contando el padding del elemento pero no el borde.

Métodos `outerWidth()` e `outerHeight()`:

Reciben un objeto jQuery y devuelven las dimensiones externas del primer elemento de dicho objeto jQuery recibido por parámetro, esto es, la anchura y altura respectivamente del elemento contando el padding del elemento y su borde.

Nota: Como podremos imaginarnos, si un elemento no tiene borde los valores de `innerWidth` e `outerWidth` serán exactamente los mismos, así como los valores de `innerHeight` y `outerHeight`.

Métodos `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.

Nota: Para acceder a los valores top y left del objeto de retorno podemos hacer algo así:

```
posicionReal = $("#idelemento").offset();
alert(posicionReal.top);
alert(posicionReal.left);
```

4.3.1.- Función que muestra las dimensiones de un elemento

Por hacer unas pruebas con estos métodos, vamos a comenzar creando una función que muestra en una caja de alerta las dimensiones de un elemento cuyo [selector](#) se envíe por parámetro. A la función enviaremos el selector y luego con jQuery mostraremos sus valores de anchura y altura, tanto de la parte interior del elemento (`innerWidth` e `innerHeight`), como del elemento completo con su borde (`outerWidth` y `outerHeight`).

```
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);
}
```

Como decíamos, las dimensiones externas toman en cuenta el borde del elemento, si es que tiene, y las dimensiones internas no toman en cuenta el posible borde.

4.3.2.- Función para mostrar la posición de un elemento

Ahora vamos a hacer una función similar a la anterior para mostrar un ejemplo de uso de las funciones `position()` y `offset()`. Esta función recibe un [selector](#) y muestra la localización de este elemento, tal como me la devuelven los métodos `position()` y `offset()`.

```
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);
}
```

Si invocamos esta función sobre un elemento cualquiera que no tenga margen, las dos posiciones devueltas por `position()` y `offset()` serán las mismas, pero si aplicamos un margen a ese elemento, el elemento cambiará de lugar en la página y entonces el valor de `offset()` también cambiará, pero no el de `position()`.

4.3.3.- Ejemplo completo sobre los métodos de dimensiones y posición de elementos

Las dos funciones anteriores las podemos ver en marcha en un ejemplo que hemos creado para poder explicar mejor todos los métodos comentados en este artículo de DesarrolloWeb.com.

En el ejemplo simplemente se realizan las acciones para averiguar las posiciones y dimensiones de un par de elementos de la página. Además, tenemos un par de botones para alterar el CSS de los elementos dinámicamente y así volver a ver sus posiciones y dimensiones y comprobar cómo han cambiado.

Realmente no sirve de mucho el ejemplo, pero al menos esperamos que resultará bastante didáctico. Podemos [verlo en marcha en una página aparte](#).

Ahora el código de este ejemplo, que no debería resultar muy complicado si hemos seguido el manual de jQuery hasta este punto.

```
<html>
<head>
<title>Funciones CSS en jQuery</title>
<script src="../jquery-1.4.1.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>
```

Para acabar, podemos [ver este script funcionando en una página aparte](#).

Artículo por *Miguel Angel Alvarez*

Parte 5:

Eventos en jQuery

Los eventos son una parte fundamental en el desarrollo de aplicaciones enriquecidas del lado del cliente. Aprendemos todo sobre los eventos en jQuery.

5.1.- Eventos en jQuery

Una introducción al trabajo con eventos en el framework Javascript jQuery.

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.

Así pues, merece la pena estudiar los eventos a fondo, algo que haremos a lo largo de éste y varios siguientes artículos del [Manual de jQuery](#). No obstante, cabe decir que, a lo largo del presente manual, hemos ya trabajado en repetidas ocasiones 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 clic, pero hay mucho más.

Comenzaremos por refrescar lo que ya deberíamos saber sobre los eventos, que aprendimos en uno de los primeros artículos del presente manual: [Pasos para utilizar jQuery en tu página web](#).

Nota: Puedes ver el [videotutorial Introducción a los eventos en jQuery](#) para que te sea más sencillo todo.

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 clicComo 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 clic, 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.

Convenía explicar todos estos puntos, aunque probablemente ya los conocíamos, si es que hemos seguido con atención este

[Manual de jQuery](#). Es importante que el lector tenga en mente esta estructura de trabajo con eventos para poder asimilar fácilmente los nuevos conocimientos.

Si lo deseamos, podemos [ver el anterior script en marcha en una página aparte](#).

En el ejemplo anterior vimos cómo realizar un evento clic, pero claro que no es el único evento que podemos aplicar a una web. En el próximo artículo veremos un listado de los tipos de evento disponibles en jQuery, pero antes de eso vamos a ver un ejemplo adicional sobre eventos, en el que vamos a incorporar el evento `dblclick` que aun no habíamos visto.

5.1.1.- Ejemplo de 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  
dblclick</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.

```
<html>  
<head>  
<title>Trabajando con eventos</title>  
<script src="../jquery-1.4.1.min.js"></script>  
<script>  
var numClics = 0;  
var numDobleClics = 0;
```



```
$(document).ready(function() {  
  
    $("#micapa").dblclick(function(e) {  
        numDobleClics++;  
        $("#mensaje").html("Doble Clic " + numDobleClics);  
    });  
    $("#micapa").click(function(e) {  
        numClics++;  
        $("#mensaje").html("Clic " + numClics);  
    });  
})  
</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>
```

Quizás quieras [ver funcionando de este ejemplo de evento clic y doble-clic](#).

En este artículo sólo hemos conocido los manejadores de eventos clic y doble-clic, pero hay muchos más. En el próximo artículo presentaremos un listado de los tipos de manejadores de eventos disponibles en jQuery.

Artículo por *Miguel Angel Alvarez*

5.2.- Manejadores de eventos en jQuery

Un listado con los distintos manejadores eventos que podemos definir en jQuery, ordenados por los tipos eventos de ratón, eventos de teclado o cualquiera de los dos.

En el capítulo anterior realizamos una primera introducción a los eventos en jQuery, que no resultaba mucho más que un simple repaso a todo lo que hemos visto hasta el momento en el [Manual de jQuery](#) que venimos publicando en DesarrolloWeb.com. Ahora veremos un listado completo de todos los eventos que podremos realizar con este framework Javascript.

Con jQuery podemos implementar todos los eventos que existen en Javascript, y alguno un poco más evolucionado, para los que hay una función propia para cada uno. Lo cierto es que la documentación de jQuery, en la sección de eventos, mantiene una lista de los posibles métodos para crear eventos, aunque están mezclados con otra serie de propiedades y métodos relacionados con eventos. He aquí un resumen de los tipos de eventos con sus funciones:

5.2.1.- 1) Eventos relacionados con el ratón

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

click()

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

dblclick()

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

hover()

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

mousedown()

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

mouseup()

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

mouseenter()

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

mouseleave()

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

mousemove()

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

mouseout()

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

mouseover()

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

toggle()

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

5.2.2.- 2) Eventos relacionados con el teclado

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

keydown()

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

keypress()

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

keyup()

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

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.

5.2.3.- 3) Eventos combinados teclado o ratón

Ahora mostramos varios eventos que pueden producirse tanto por el ratón como por el teclado, es decir, como resultado de una acción con el ratón 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 ratón.

focus()

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

Esta enumeración de los tipos de manejadores de eventos se completa con ejemplos y explicaciones adicionales en los siguientes artículos del [Manual de jQuery](#).

Artículo por *Miguel Ángel Álvarez*

5.3.- Introducción Objeto evento en jQuery

Explicaciones iniciales sobre el objeto evento en jQuery y mostramos cómo averiguar la posición del ratón al producirse un evento en la página.

Vamos a hacer un inciso en el [Manual de jQuery](#) para dar una breve introducción al objeto evento en jQuery y ofrecer un ejemplo bastante práctico, para saber cuál es la posición del ratón al producirse un evento. Digo inciso porque no vamos a explicar todo lo que podremos encontrarnos en el objeto evento, sino que vamos a dar algunas nociones que deberemos conocer para poder acompañar los siguientes artículos sobre eventos. En breve crearemos un artículo que explique todas las propiedades y métodos de este importante objeto de jQuery.

Lo cierto es que lo que vamos a explicar ahora ya lo habíamos adelantado brevemente en otros artículos anteriores en los que comenzamos a ver los [eventos de jQuery](#). Como ya hemos empezado a utilizar el objeto de evento, no debería resultarnos del todo extraño, pero tenemos muchas otras cosas que comentar.

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.

Como cualquier otro objeto, el mencionado objeto de evento contiene diversas propiedades y métodos, los cuales detallaremos uno por uno más adelante. Sin embargo, cabe decir que nosotros ya hemos utilizado uno de los métodos en bastantes ejemplos a lo largo de este manual. Se trata del método `preventDefault()` del objeto evento, que sirve para prevenir (no realizar) el comportamiento por defecto de ese evento que estamos codificando.

El ejemplo que hemos realizado varias veces sobre `preventDefault()` es 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.

5.3.1.- Averiguar la posición del ratón al hacer clic

En el objeto evento, entre otras muchas cosas, existen dos propiedades que nos informarán sobre la posición del ratón 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.

Se puede [ver una página con este código en funcionamiento](#).

Este código se puede modificar fácilmente para que se muestre las coordenadas del ratón 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.

```
<html>
<head>
<title>Trabajando con el objeto evento</title>
<script src="../../jquery-1.4.1.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>
```

Podemos [ver el ejemplo en funcionamiento en una página aparte](#).

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 ratón 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)
});
```

Si lo deseas, puedes [ver el script en marcha aquí](#).

Con las nociones que tenemos en este momento sobre el objeto evento podremos continuar con las explicaciones sobre eventos, en las que utilizaremos varios aspectos de este objeto. Así pues, puedes continuar esta lectura aprendiendo acerca de los [Eventos de ratón](#).

Artículo por *Miguel Angel Alvarez*

5.4.- Eventos de ratón en jQuery mouseenter y mouseleave

Práctica con eventos de ratón en jQuery, en la que mostraremos el uso de mouseenter y mouseleave, junto con el objeto evento, para averiguar la posición del ratón.

En este artículo de DesarrolloWeb.com veremos un ejemplo de página sencilla que utiliza eventos de ratón, para la construcción de un sistema de tip muy simple, es decir, construiremos una serie de áreas "calientes" en la página, sobre las que situando el ratón por encima, aparecerá un mensaje explicativo que tenemos en otra capa.

Como ya debemos de conocer, si hemos leído hasta este punto el [Manual de jQuery](#), existen [diversos eventos](#) que se invocan

al realizar acciones con el ratón, como clics, movimiento del puntero o posicionar el puntero sobre ciertos elementos. En este artículo utilizaremos `mouseenter` y `mouseleave`, que son los eventos más interesantes y útiles si queremos detectar el momento en el que entramos con el puntero del ratón 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 ratón en el momento de producirse el evento, que podemos extraer con las propiedades `pageX` y `pageY` del objeto evento.

Nota: Para el que llegue aquí sin haber leído otras informaciones sobre eventos en jQuery, le recomendamos comenzar la lectura por el artículo de [Eventos en jQuery](#).

5.4.1.- Efecto de tip simple en jQuery con los eventos `mouseenter` y `mouseleave`

En el artículo anterior mostramos cómo [averiguar la posición del ratón al hacer clic 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 ratón 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 ratón.

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 ratón 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.

```
<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 src="../jquery-1.4.1.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>
```

Ahora podemos [ver el ejercicio en marcha](#).

Con esto estamos aprendiendo un poco más sobre eventos en jQuery. Hemos visto un par de aplicaciones interesantes de eventos de ratón, concretamente mouseenter y mouseleave Pero aun nos quedan bastantes cosas por ver que dejaremos para próximos artículos.

Nota: Tenemos un video titulado "[Videotutorial: manejo de eventos al detalle en jQuery](#)" que os puede ayudar con esto de los eventos.

Artículo por *Miguel Ángel Álvarez*

5.5.- Eventos de teclado en jQuery

Cómo trabajar con eventos de teclado en jQuery y saber qué teclas han pulsado los usuarios, a través de la propiedad which del objeto evento.

Estamos aprendiendo sobre los eventos en jQuery y ahora 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 conocido en el [manual de jQuery](#), pero con los eventos de teclado hay algo que todavía no hemos visto y que resulta fundamental. Se trata que, cuando se produce el evento de teclado, 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, como vimos en el artículo [Manejadores de eventos en jQuery](#), keydown, keypress y keyup. Realmente no actúan por separado, sino que se produce una combinación de éstos al ir presionando y soltando las teclas, como se puede deducir de las explicaciones del mencionado artículo.

Nota: 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.

5.5.1.- Secuencia de eventos de teclado

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 + ": " + evento.which + ", ")
}
```

Esta función recibe el evento y escribe en una capa el tipo de evento, que se consigue con la propiedad type del objeto evento, y luego un código de la tecla pulsada, que se consigue con la propiedad which del objeto evento.

Nota: el tipo de evento no lo habíamos visto todavía, pero es otra de las propiedades que encontramos en el objeto evento que recibe la función que tiene el código a ejecutar por el evento. Esta propiedad type simplemente es un string con la cadena que identifica el tipo de evento que se está procesando ("keydown", "keyup", "click" o cualquier otro). La tecla pulsada, que se obtiene con la propiedad which, la trataremos con detalle dentro de poco.

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).

Esto se puede [ver en marcha en una página aparte](#).

Creo que merece la pena presentar el código completo del anterior ejemplo:

```
<html>
<head>
<title>Trabajando con eventos de teclado en jQuery</title>
<script src="../jquery-1.4.1.min.js"></script>
<script>
function operaEvento(evento){
    $("#loescrito").html($("#loescrito").html() + evento.type + ": " + evento.which + ", ")
}
```

```
$(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>
```

5.5.2.- Averiguar qué tecla fue pulsada

A través de la propiedad `which` del objeto evento de jQuery 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.

```
<html>
<head>
<title>Trabajando con eventos de teclado en jQuery</title>
<script src="../jquery-1.4.1.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>
```

Podemos ver el [ejemplo en marcha en una página aparte](#).

Con esto habremos aprendido ya a manejar eventos de teclado, aunque os recomendamos experimentar vosotros mismos con este tipo de eventos modificando el script y ver nuestro [Videotutorial: manejo de eventos al detalle en jQuery](#).

Artículo por *Miguel Angel Alvarez*

5.6.- Definir eventos con bind() y eliminarlos con unbind()

El método bind() sirve para definir eventos de manera genérica, de cualquier tipo. Con el método unbind() podemos eliminar un manejador cualquiera indicado con bind().

A lo largo del [Manual de jQuery](#) ya hemos aprendido bastantes cosas sobre los eventos en este framework Javascript. Hemos conocido cómo a partir de diversos métodos como click(), mouseenter() y similares, podemos asignar un manejador de evento (una función) a un tipo de evento concreto.

Ahora vamos a aprender a utilizar una única función de jQuery para definir cualquier tipo de evento, lo que sería como una manera genérica de definir eventos, de cualquier tipo, sobre elementos de la página. Además mostraremos cómo se puede eliminar un evento, quitando un posible manejador de eventos definido con anterioridad.

Aprendimos en el pasado a definir eventos por medio de unas funciones específicas para cada tipo de evento. Por ejemplo:

```
$("#elem1").click(function(){
    //evento clic sobre el elemento con id "elem1"
});

$("#elem2").mouseenter(function(){
    //evento de entrar con el ratón sobre el elemento con id "elem2"
});
```

Estas maneras de trabajar son perfectamente viables y muy cómodas de utilizar para crear eventos de un tipo en concreto, pero en jQuery existe otra manera de definirlos con la que ganaremos alguna ventaja.

5.6.1.- Método bind() para definir cualquier tipo de evento

Con el método bind() podemos definir de una manera genérica cualquier tipo de evento, o incluso un mismo manejador de eventos para distintos tipos de eventos distintos. El uso más habitual de este método es el siguiente:

```
bind(tipo_de_evento, manejador)
```

Como primer parámetro enviamos el tipo de evento que queremos definir. Si se desea, podríamos especificar varios tipos de eventos separados por un espacio y así asignar un mismo manejador de evento para varios tipos de situaciones.

Como segundo parámetro se indica el manejador o función a ejecutar cuando se produzca el evento, igual que se definía con los métodos click(), mouseleave() o similares, para un tipo de evento en concreto.

Un ejemplo sencillo de este modo de definir eventos es el siguiente:

```
$(".miclase").bind("click", function(){
    alert("Has hecho clic");
});
```

Al hacer clic en cualquier elemento de la clase CSS "miclase", se mostrará un mensaje en una caja de alerta.

Ahora podemos ver cómo se crearía una función que se asignaría para varios tipos de eventos a la vez.

```
$("#p").bind("click mouseenter mouseleave", function(e){
    if ($(this).css("color")!="rgb(250, 100, 0)")
```

```
    $(this).css("color", "rgb(250, 100, 0)");  
else  
    $(this).css("color", "rgb(150, 0, 255)");  
})
```

Como se puede ver, se ha definido un evento para todos los párrafos de la página, que se activará con los tipos de eventos: "click mouseenter mouseleave". La función que hace de manejador de eventos averigua el color del elemento y lo va intercambiando entre dos colores distintos. Este evento se ejecutará al hacer clic, al entrar en el elemento con el puntero del ratón o al salir del elemento con el ratón.

5.6.2.- Eliminar un evento con la función unbind()

Ahora vamos a aprender a hacer el paso contrario, es decir, eliminar un evento previamente asignado a un elemento o varios elementos de la página. El procedimiento es bastante sencillo.

Si invocamos a unbind() sin ningún parámetro, eliminamos todos los manejadores de eventos, de cualquier tipo de evento, de los objetos jQuery.

```
$("#p").unbind();
```

Así hemos eliminado todos los eventos asociados con los párrafos de la página. Pero quizás una situación más habitual es que deseemos eliminar sólo los eventos de un tipo y para ello simplemente tenemos que indicar como parámetro ese tipo concreto de evento deseado.

```
$("#p").unbind("click");
```

Esta sentencia provocará se descarten que todos los manejadores de eventos asociados al clic sobre los párrafos. Como se puede entender, sobre esos elementos no ocurrirá nada en el momento en que hagamos clic. Además, en el supuesto que otros scripts Javascript hayan definido algún manejador de evento clic sobre alguno de los elementos, unbind() también eliminará esos posibles eventos.

Para no eliminar todos los manejadores de eventos de un tipo determinado podemos especificar la función que deseamos descartar en la lista de parámetros de la llamada a unbind(). Esto funcionaría en un esquema de código como el siguiente.

```
var funcionManejador = function(e) {  
    // cualquier código  
}  
$("#p").bind('click', funcionManejador);  
$("#p").unbind('click', funcionManejador);
```

Siempre tendremos que colocar la función dentro de una variable, para poder referirnos a esa misma variable tanto al crear el evento con bind(), como al descartarlo con unbind().

5.6.3.- Ejemplos con los métodos bind() y unbind() de jQuery

A continuación puede verse el código de una página completa que pone en práctica las explicaciones ofrecidas en el artículo.

El ejemplo se puede [ver en marcha en una página aparte](#).

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"  
"http://www.w3.org/TR/html4/strict.dtd"  
>  
<html lang="es">  
<head>  
<title>Eventos bind</title>  
    <script src="../jquery-1.4.2.min.js"></script>  
<script language="javascript">  
$(document).ready(function() {  
  
    $("#p").bind("click mouseenter mouseleave", function(e) {  
        if ($("#this").css("color")!="rgb(250, 100, 0)")  
            $(this).css("color", "rgb(250, 100, 0)");  
        else  
            $(this).css("color", "rgb(150, 0, 255)");  
    })  
})
```

```
function clicAlerta(){
    alert("Has hecho clic");
}

$(".miclase").bind("click", clicAlerta);

$("#quitarevento").bind("click", function(){
    $(".miclase").unbind("click", clicAlerta);
})
})
</script>
</head>
<body>

<p class="miclase">Primer párrafo</p>
<p>Otro párrafo</p>

<input type=button value="Quitar el alert del clic del primer párrafo" id="quitarevento">
</body>
</html>
```

En el próximo artículo veremos una utilidad interesante para definir eventos sobre elementos actuales y futuros sobre un selector jQuery, que nos facilitará las cosas en scripts más complejos.

Artículo por *Miguel Angel Alvarez*

5.7.- Eventos definidos con `live()` en jQuery

Método `live()` para definir eventos en jQuery: cómo crear eventos sobre elementos que coincidan con un selector, en el momento actual o en el futuro.

Hemos visto muchas técnicas para definir y tratar eventos en jQuery a lo largo de los artículos anteriores del [Manual de jQuery](#). Para seguir explorando las posibilidades de este framework Javascript vamos a aprender ahora a definir eventos "live". Es algo así como una definición de evento "en directo" o durante toda la "vida" de la página, es decir, una definición de un evento sobre los elementos actuales y futuros que casen con un selector.

El método `live()` funciona de manera similar al [método `bind\(\)` de jQuery](#), con la particularidad que la asignación del evento es "dinámica" y afecta no sólo a los elementos que casen con el selector en el momento de la invocación, sino también todos los elementos que se puedan definir en un futuro y que casen con ese selector.

No sé si se habrá podido entender exactamente lo que se consigue con `live()`, pero lo podemos ver con un ejemplo sencillo que lo aclarará todo. Veamos esta sentencia:

```
$(".miclase").bind("click", mifuncion);
```

Está definiendo un evento "click" sobre todos los elementos de la clase (class de CSS) "miclase". Hasta aquí ya debemos de conocer todos esta el método `bind()`, por lo que no debería haber ningún problema, pero ahora veamos esta misma sentencia pero utilizando el método `live()`.

```
$(".miclase").live("click", mifuncion);
```

Esto sirve para lo mismo que hacíamos con `bind()`, pero además afectará a todos los elementos que puedan tener la clase "miclase" en el futuro y no sólo en el momento que se ejecuta esa sentencia.

¿Cómo puede haber otros elementos de esa clase en el futuro? Pues simplemente porque los crees dinámicamente con jQuery o porque asignes dinámicamente una clase CSS, u otro atributo, a un elemento que no la tenía, o que traigas por Ajax un contenido y que tenga elementos que casen con el selector, etc.

5.7.1.- Ejemplo de asignación de manejador de evento por `live()`

Veamos la siguiente demostración del funcionamiento de `live()`. Tenemos varios elementos:

```
<div class="verde">Esta capa tiene la clase verde (haz clic)</div>
<div class="verde">Segunda capa donde coloco la clase verde</div>
<div id="noverde">Tercera capa que no es verde</div>
<div class="verde">Otra con clase verde</div>
```

Sin varias divisiones donde todas menos una tienen la clase "verde". Veamos como puedo asignar un evento de tipo "click" por medio del método live():

```
$(".verde").live("click", function(e){
    var elem = $(this);
    if (elem.html()!="Hiciste clic!!"){
        elem.html("Hiciste clic!!");
    }else{
        elem.html("Hiciste de nuevo clic!!");
    }
});
```

Es un evento que permite cambiar el texto del elemento cuando se pulsa sobre él y lo aplicamos sobre todos los elementos de la clase "verde".

Ahora tenemos un par de botones para hacer cosas con la página y cambiarla un poco.

```
<input type="button" value="insertar nuevo elemento verde" id="insertarelem">
<input type="button" value="Poner la clase verde en el div que no la tiene" id="ponerclaseverde">
```

Cuando se pulse el primer botón, voy a insertar un nuevo elemento en la página al que le pondremos la clase "verde". Eso lo consigo con este código:

```
$("#insertarelem").click(function(e){
    var nuevoElemento = $('<div class="verde">Elemento creado e insertado dinamicamente</div>');
    nuevoElemento.appendTo($(document.body));
});
```

Los elementos que se creen al apretar ese botón tendrán la clase verde y por tanto la funcionalidad especificada con el método live() para definir el evento clic.

El segundo botón asigna la clase "verde" al elemento DIV del principio, que no la tenía, lo que conseguimos así:

```
$("#ponerclaseverde").click(function(e){
    $("#noverde").addClass("verde");
});
```

Al asignar esa clase al elemento también se aplicará la funcionalidad definida para el evento click con live().

Esto lo podemos [ver en funcionamiento en una página aparte](#).

Para acabar, dejamos el código completo de esta página de ejemplo de live() en jQuery.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd"
>
<html>
<head>
<title>Eventos live</title>
<style type="text/css">

    .verde{color: green;}

</style>
<script src="../jquery-1.4.2.min.js"></script>
<script language="javascript">
$(document).ready(function(){
    $(".verde").live("click", function(e){
        var elem = $(this);
        if (elem.html()!="Hiciste clic!!"){
            elem.html("Hiciste clic!!");
        }else{
            elem.html("Hiciste de nuevo clic!!");
        }
    })
});

$("#insertarelem").click(function(e){
```

```
    var nuevoElemento = $('<div class="verde">Este elemento se ha creado e insertado dinamicamente
(haz clic)</div>');
    nuevoElemento.appendTo($(document.body));
});

$("#ponerclaseverde").click(function(e){
    $("#noverde").addClass("verde");
});
})
</script>
</head>
<body>

<div class="verde">Esta capa tiene la clase verde (haz clic)</div>
<div class="verde">Segunda capa donde coloco la clase verde</div>
<div id="noverde">Tercera capa que no es verde</div>
<div class="verde">Otra con clase verde</div>

<input type="button" value="insertar nuevo elemento verde" id="insertarelem">
<input type="button" value="Poner la clase verde en el div que no la tiene" id="ponerclaseverde">
</body>
</html>
```

Artículo por *Miguel Angel Alvarez*

Parte 6:

Desarrollo de plugins en jQuery

Los plugins en jQuery nos permiten hacer desarrollos que podremos reutilizar con facilidad en diversos sitios y que también podrán usar otros desarrolladores. Los plugins te ayudarán a hacer código de calidad en jQuery.

6.1.- Plugins en jQuery

Veamos qué son los Plugins en jQuery y cómo podemos crearlos para expandir las posibilidades del framework.

Si has llegado a este punto en el [Manual de jQuery](#) y has puesto en práctica los ejemplos realizados hasta ahora, no me cabe duda que tendrás ya una pequeña idea de las cosas que se pueden hacer con el framework. Habrás comprobado que, con pocas líneas de código, se pueden hacer diversos efectos y dotar a la página de interacción con el usuario, pero quizás todavía te sientas un poco perdido a la hora de encarar el desarrollo de problemas más complejos con los que podrás enfrentarte.

Todavía nos queda mucho camino por delante, pero lo bueno es que, con lo que sabes hasta ahora, tienes una base suficiente

para empezar a hacer cosas más divertidas y ejemplos que merezcan más la pena como práctica para tu día a día. Además, tendrás que aprender a programar de una manera adecuada en jQuery y sobre todo crear código reutilizable y con un ciclo de vida mayor. Para cumplir todos esos objetivos vamos a pasar directamente a explicar los plugins en jQuery, una de las "herramientas" que utilizarás habitualmente, si quieres hacer cosas más avanzadas con el framework y sacarle todo su provecho.

6.1.1.- Qué son los Plugins

Los plugins son la utilidad que pone jQuery a disposición de los desarrolladores para ampliar las funcionalidades del framework. Por lo general servirán para hacen cosas más complejas necesarias para resolver necesidades específicas, pero las hacen de manera que puedan utilizarse en el futuro en cualquier parte y por cualquier web.

En la práctica un plugin no es más que una función que se añade al objeto jQuery (objeto básico de este framework que devuelve la función jQuery para un selector dado), para que a partir de ese momento responda a nuevos métodos. Como ya sabemos, en este framework todo está basado en el objeto jQuery, así que con los plugins podemos añadirle nuevas utilidades.

Voy a poner un ejemplo un poco abstracto para ver si podemos llegar a la idea de cómo es un plugin. Imagina que necesitas que los elementos de la página "bailen" (parpadeen, se muevan, interactuen con el usuario de una manera concreta, o lo que sea que necesites), pues creas una función para hacer eso. Haces que esa función sea un plugin llamado "bailar" y a partir de entonces cualquier elemento de la página que lo desees podrá bailar. Para ello simplemente invocas ese método del objeto jQuery sobre el elemento o elementos que selecciones.

```
//con esto bailan todos los párrafos
$("p").bailar();

//con esto bailan los elementos de la clase "artista"
$(".artista").bailar();

//con esto baila el elemento con id="lola"
$("#lola").bailar();
```

Espero que el ejemplo no haya parecido muy tonto, pero es que los plugins no son nada del otro mundo, son simplemente eso, extensiones del framework para crear cualquier funcionalidad que podamos necesitar en los elementos de la página, por muy especial, o tonta, que sea.

Lo genial de los plugins es que tú podrás utilizar esa funcionalidad en donde desees a partir de ahora, ya que estará perfectamente a tu disposición, siempre que tengas cargado el plugin. Incluso si tu generosidad es tal, la podrás proporcionar a otras personas para que la utilicen en sus desarrollos. Claro que, para conseguir todo esto, será necesario que programes los plugins atendiendo a una serie de normas, bastante sencillas pero importantes para asegurar que se puedan utilizar en cualquier parte y para cualquier [selector de jQuery](#).

6.1.2.- Cómo se crea un plugin de jQuery

Los plugins en jQuery se crean asignando una función a la propiedad "fn" del objeto jQuery. A partir de entonces, esas funciones asignadas se podrán utilizar en cualquier objeto jQuery, como uno de los muchos métodos que dispone dicho objeto principal del framework.

Nota: La creación de plugins, para ampliar las funcionalidades de jQuery, es una cosa tan básica que la mayoría de las funciones con las que está dotado el propio framework están incluidas en el objeto jQuery por medio de plugins. Es decir, en la construcción del framework en muchas de las ocasiones simplemente se crean plugins para extenderlo. Así pues, esta técnica es usada, no sólo por terceros desarrolladores, para crear nuevos componentes, sino también por el propio equipo de jQuery para el diseño base de este framework.

Si lo deseamos, aparte de seguir los próximos artículos de este manual, podemos ver el código fuente del framework o cómo están hechos los plugins de otros desarrolladores, para tener una idea sobre cómo se utilizan.

A modo de ejemplo, podemos ver a continuación un código fuente de un plugin muy sencillo:

```
jQuery.fn.desaparece = function() {
    this.each(function() {
        elem = $(this);
```

```
    elem.css("display", "none");
  });
  return this;
};
```

Este plugin permitiría hacer desaparecer a los elementos de la página y podríamos invocarlo por ejemplo de la siguiente manera:

```
$("#h1").desaparece();
```

En el siguiente artículo [veremos con mayor detalle la creación de un plugin de jQuery](#) y explicaremos varios temas que resultarán de vital importancia para entender el código anterior y para construirlos nosotros mismos.

Artículo por *Miguel Ángel Álvarez*

6.2.- Reglas para el desarrollo de plugins en jQuery

Para construir plugins en jQuery tenemos que seguir una serie de normas. Además veremos un nuevo ejemplo práctico sobre el desarrollo de plugins en jQuery.

Con los plugins en jQuery podemos ampliar Framework, creando nuevos métodos para dotar de nuevas funcionalidades al objeto jQuery. En el artículo anterior ofrecimos una explicación general sobre los [plugins en jQuery](#) y en el presente vamos a adentrarnos en su desarrollo.

Como habrás comprobado en el [Manual de jQuery](#), el framework ya contiene muchas funciones para hacer cosas interesantes, que se basan en la utilización del objeto jQuery, así que ahora aprendamos a extender este objeto para proporcionar funcionalidades nuevas a nuestras páginas. Pero atención, porque tenemos que realizar el trabajo siguiendo una serie de normas, para asegurar que los plugins funcionen como deben y los pueda utilizar cualquier desarrollador en cualquier página web.

Aquí puedes ver un listado normas, que son sólo unas pocas, pero que resultan tremendamente importantes.

- El archivo que crees con el código de tu plugin lo debes nombrar como jquery.[nombre de tu plugin].js. Por ejemplo jquery.desaparece.js.
- Añade las funciones como nuevos métodos por medio de la propiedad fn del objeto jQuery, para que se conviertan en métodos del propio objeto jQuery.
- Dentro de los métodos que añades como plugins, la palabra "this" será una referencia al objeto jQuery que recibe el método. Por tanto, podemos utilizar "this" para acceder a cualquier propiedad del elemento de la página con el estamos trabajando.
- Debes colocar un punto y coma ";" al final de cada método que crees como plugin, para que el código fuente se pueda comprimir y siga funcionando correctamente. Ese punto y coma debes colocarlo después de cerrar la llave del código de la función.
- El método debe retornar el propio objeto jQuery sobre el que se solicitó la ejecución del plugin. Esto lo podemos conseguir con un return this; al final del código de la función.
- Se debe usar this.each para iterar sobre todo el conjunto de elementos que puede haber seleccionados. Recordemos que los plugins se invocan sobre objetos que se obtienen con selectores y la función jQuery, por lo que pueden haberse seleccionado varios elementos y no sólo uno. Así pues, con this.each podemos iterar sobre cada uno de esos elementos seleccionados. Esto es interesante para producir código limpio, que además será compatible con selectores que correspondan con varios elementos de la página.
- Asigna el plugin siempre al objeto jQuery, en vez de hacerlo sobre el símbolo \$, así los usuarios podrán usar alias personalizados para ese plugin a través del método noConflict(), descartando los problemas que puedan haber si dos plugin tienen el mismo nombre.

Estas reglas serán suficientes para plugins sencillos, aunque quizás en escenarios más complejos en adelante necesitaremos aplicar otras reglas para asegurarnos que todo funcione bien.

6.2.1.- Ejemplo de un plugin en jQuery

Ahora que ya sabemos las reglas básicas para hacer plugins podemos crear uno por nuestra cuenta que nos sirva para practicar lo que hemos aprendido. Te sugiero que identifiques los lugares donde hemos aplicado cada una de las anteriores normas de la lista, o al menos las que se puedan aplicar en este plugin tan simple que vamos a ver.

El plugin que vamos a construir sirve para hacer que los elementos de la página parpadeen, esto es, que desaparezcan y vuelvan a aparecer en un breve instante. Es un ejemplo bien simple, que quizás tenga ya alguna utilidad práctica en tu sitio, para llamar la atención sobre uno o varios elementos de la página.

Para hacerlo, utilizaremos otras funciones del framework como `fadeOut()` (para hacer desaparecer al elemento) y `fadeIn()` (para que aparezca de nuevo).

```
jQuery.fn.parpadea = function() {
  this.each(function(){
    elem = $(this);
    elem.fadeOut(250, function(){
      $(this).fadeIn(250);
    });
  });
  return this;
};
```

Con `this.each` creamos un bucle para cada elemento que pueda haberse seleccionado para invocar el plugin. Con `elem=$(this)` conseguimos extender a `this` con todas las funcionalidades del framework y el objeto jQuery resultante guardarlo en una variable. Luego invocamos `fadeOut()`, enviando como parámetro un número que son los milisegundos que durará el efecto de desaparecer el elemento. Luego enviamos como parámetro una nueva función que es un callback, que se ejecutará cuando haya terminado `fadeOut()` y en esa función callback se encargará simplemente de ejecutar un `fadeIn()` para mostrar de nuevo el elemento.

Nota: A lo largo del [Manual de jQuery](#) hemos visto varias de las cosas que utilizamos en este ejemplo, como los [efectos en jQuery](#) y las [funciones Callback](#).

Ahora veamos cómo podríamos invocar este plugin:

```
$(document).ready(function(){
  //parpadean los elementos de class CSS "parpadear"
  $(".parpadear").parpadea();

  //añado evento clic para un botón. Al pulsar parpadearán los elementos de clase parpadear
  $("#botonparpadear").click(function(){
    $(".parpadear").parpadea();
  })
});
```

Dado el código anterior, al abrir la página parpadearán los elementos de la clase "parpadear" y luego habrá un botón que repetirá la acción de parpadear cuando se pulse.

En este caso no hemos colocado el script en un archivo aparte con el nombre `jquery.parpadea.js`, tal como se recomendaba, pero de momento será suficiente para probar esto de los plugins y quizás más fácil porque así no necesitamos más que un archivo HTML con todo el código junto. Podemos ver el código completo de este ejemplo a continuación:

```
<html>
<head>
<title>Creando plugins en jQuery</title>
  <script src="../jquery-1.4.1.min.js"></script>
<script>
jQuery.fn.parpadea = function() {
  this.each(function(){
    elem = $(this);
    elem.fadeOut(250, function(){
      $(this).fadeIn(250);
    });
  });
  return this;
};
```



```
};
$(document).ready(function(){
  //parpadean los elementos de class CSS "parpadear"
  $(".parpadear").parpadea();

  //añado un evento clic para un botón, para que al pulsarlo parpadeen los elementos de clase
  parpadear
  $("#botonparpadear").click(function(){
    $(".parpadear").parpadea();
  })
})
</script>

</head>
<body>
  <p class="parpadear">Hola que tal, esto parpadeó gracias a jQuery!</p>
  <p>Parafo normal que no va a parpadear.</p>
  <p class="parpadear">Si parpadea</p>
  <p>Parafo normal que no va a parpadear tampoco...</p>
  <div class="parpadear" style="background-color: #ff9966; padding: 10px;">Esta capa también tiene
  la clase parpadear, con lo que ya se sabe...</div>
  <p><input type="button" value="Parpadea de nuevo" id="botonparpadear"></p>
</body>
</html>
```

Para acabar, puedes [acceder al ejercicio en una página aparte](#).

Nota: Contamos con un [taller de jQuery](#) donde recopilamos un conjunto de plugins y mostramos cómo se construyen.

Artículo por *Miguel Angel Alvarez*

6.3.- Plugin jQuery: textarea con cuenta de caracteres

Segundo ejemplo de plugin práctico en jQuery para hacer textarea que lleva la cuenta de los caracteres escritos por el usuario.

Este es un taller práctico sobre jQuery que esperamos sirva para que las personas puedan continuar aprendiendo la manera de crear sus propios plugins. Como ya sabemos, los plugins son una manera óptima de programar tus scripts jQuery, ya que permitirán solucionar sus necesidades y además crear código limpio y reutilizable.

En los dos artículos anteriores ya estuvimos hablando de los [Plugins en jQuery](#) y de las [reglas fundamentales para desarrollarlos](#). También vimos un primer ejemplo de un plugin sencillo, que espero nos haya abierto las miras y dado una idea sobre las posibilidades de construcción de componentes para páginas web. En este artículo continuaremos ofreciendo ejemplos para reforzar lo aprendido y para que las personas puedan familiarizarse aun más con el modo de creación de plugins en jQuery.

El objetivo del ejemplo que ocupará este artículo es la creación de un plugin para conseguir que un campo textarea de formulario informe en todo momento de caracteres que ha escrito el usuario. Es decir, vamos a hacer un método del objeto jQuery que servirá para decirle a los campos de texto textarea que se expandan para convertirse en un textarea que cuente los caracteres en una capa de texto de al lado.

Para tener una idea exacta de nuestros objetivos podemos [ver el ejemplo en marcha que vamos a desarrollar](#).

6.3.1.- Entendamos el plugin textarea con contador de caracteres

Para hacer los textareas que cuenten caracteres nosotros queremos hacer algo como esto en jQuery.

```
$(".textarea").cuentaCaracteres();
```

Con eso queremos conseguir que a todos los textareas del documento HTML les aparezca una información al lado con el

número de caracteres que tenga el textarea escrito dentro. Esa cuenta de caracteres debe mostrarse nada más cargarse la página y actualizarse cuando se escriba algo dentro. Todo eso se automatizará, para que no tengamos que hacer nada, salvo la anterior llamada al plugin.

Entonces, dentro del plugin tenemos que hacer varias cosas.

1. Un bucle con each para recorrer todos los objetos que pueda haber en el objeto jQuery que reciba el método para activar este plugin. Este paso es igual en todos los plugins.
2. Dentro de ese bucle podemos iterar con todos los elementos que haya en el objeto jQuery, que vamos a suponer son textareas. Vamos a crear un nuevo elemento DIV sobre la marcha y vamos a iniciarlo con el texto de la cuenta de caracteres actual del textarea. Ese elemento creado "on the fly" lo añadiremos al cuerpo de la página, justo después de la etiqueta del textarea.
3. Además, haremos un evento, para que cuando el usuario escriba algo en el textarea, el texto con la cuenta de caracteres se actualice automáticamente.

Estos tres pasos serían un resumen del funcionamiento del plugin, cuyo código completo podemos ver a continuación.

```
//creo el plugin cuentaCaracteres
jQuery.fn.cuentaCaracteres = function() {
  //para cada uno de los elementos del objeto jQuery
  this.each(function(){
    //creo una variable elem con el elemento actual, suponemos un textarea
    elem = $(this);
    //creo un elemento DIV sobre la marcha
    var contador = $('<div>Contador caracteres: ' + elem.attr("value").length + '</div>');
    //inserto el DIV después del elemento textarea
    elem.after(contador);
    //guardo una referencia al elemento DIV en los datos del objeto jQuery
    elem.data("campocontador", contador);

    //creo un evento keyup para este elemento actual
    elem.keyup(function(){
      //creo una variable elem con el elemento actual, suponemos un textarea
      var elem = $(this);
      //recupero el objeto que tiene el elemento DIV contador asociado al textarea
      var campocontador = elem.data("campocontador");
      //modifico el texto del contador, para actualizarlo con el número de caracteres escritos
      campocontador.text('Contador caracteres: ' + elem.attr("value").length);
    });
  });
  //siempre tengo que devolver this
  return this;
};
```

El código está comentado para que se pueda entender mejor. Quizás nos pueda llamar más la atención la línea donde se utiliza la [función jQuery para generar sobre la marcha un objeto jQuery](#) con el campo DIV con el que vamos a seguir la cuenta. Vemos que a través del método attr() accedemos al value del textarea y con la propiedad length a su longitud en caracteres.

```
var contador = $('<div>Contador caracteres: ' + elem.attr("value").length + '</div>');
```

Luego también puede que nos llame la atención el funcionamiento del [método data\(\)](#), que nos permite almacenar y recuperar datos que se guardarán en el propio objeto jQuery de cada textarea.

Así guardo una referencia al objeto con la capa contador en el textarea, en un dato llamado "campocontador".

```
elem.data("campocontador", contador);
```

Y con este otro código en el evento recupero esa capa, pues luego en el evento tengo que cambiar el contenido con la cuenta de caracteres actualizada.

```
var campocontador = elem.data("campocontador");
```

Una vez creado el plugin, convierto todos los textareas en textareas-contador de caracteres, con este código:

```
$(document).ready(function(){
  $("textarea").cuentaCaracteres();
});
```

Eso es todo, pero quizás se vea más claro si vemos el código completo del ejemplo.

```
<html>
<head>
<title>Creando plugins en jQuery</title>
  <script src="../jquery-1.4.1.min.js"></script>
</script>

//creo el plugin cuentaCaracteres
jQuery.fn.cuentaCaracteres = function() {
  //para cada uno de los elementos del objeto jQuery
  this.each(function() {
    //creo una variable elem con el elemento actual, suponemos un textarea
    elem = $(this);
    //creo un elemento DIV sobre la marcha
    var contador = $('<div>Contador caracteres: ' + elem.attr("value").length + '</div>');
    //inserto el DIV después del elemento textarea
    elem.after(contador);
    //guardo una referencia al elemento DIV en los datos del objeto jQuery
    elem.data("campocontador", contador);

    //creo un evento keyup para este elemento actual
    elem.keyup(function(){
      //creo una variable elem con el elemento actual, suponemos un textarea
      var elem = $(this);
      //recupero el objeto que tiene el elemento DIV contador asociado al textarea
      var campocontador = elem.data("campocontador");
      //modifico el texto del contador, para actualizarlo con el número de caracteres escritos
      campocontador.text('Contador caracteres: ' + elem.attr("value").length);
    });
  });
  //siempre tengo que devolver this
  return this;
};
$(document).ready(function() {
  $("textarea").cuentaCaracteres();
})
</script>

</head>
<body>
<form>
  <textarea rows=5 cols=30 id="mitextarea">hola</textarea>
  <br>
  <br>
  <textarea rows=5 cols=30 id="otrotextarea">Otra cuenta...</textarea>
</form>
</body>
</html>
```

Este ejemplo se puede [ver en una página aparte](#).

Nota: Si quieres ver más ejemplos prácticos de creación de plugins te recomiendo que leas el [Taller de JQuery](#).

Artículo por *Miguel Angel Alvarez*

6.4.- Gestión de opciones en plugins jQuery

Manera de gestionar opciones en los plugins de jQuery, a través de un objeto de options enviado al invocar el plugin, para hacerlos un poco más versátiles y con configuración más fácil.

Cuando desarrollamos plugins en jQuery debemos [atender a una serie de normas básicas](#) para que estén bien creados y

puedan funcionar en cualquier ámbito. Pero además tenemos una serie de patrones de desarrollo que debemos seguir de manera opcional para facilitarnos la vida a nosotros mismos y a otros desarrolladores que puedan utilizar nuestros plugins.

Una de las tareas típicas que realizaremos es la creación de un sistema para cargar opciones con las que configurar el comportamiento de los plugins. Estas opciones las recibirá el plugin como parámetro cuando lo invocamos inicialmente. Nosotros, como desarrolladores del plugin, tendremos que definir cuáles van a ser esas opciones de configuración y qué valores tendrán por defecto.

La ayuda del sitio de jQuery para la creación de plugins sugiere la manera con la que realizar el proceso de configuración del plugin, por medio de un objeto de "options", que nos facilitará bastante la vida.

6.4.1.- Por qué son interesantes los options en plugins

La idea que hay detrás de la carga de opciones en los plugins ya la conocemos, que éstos sean más configurables y por lo tanto más versátiles. Pero vamos a intentar dar un ejemplo más claro sobre cómo esas opciones pueden hacer a los plugins más versátiles.

Imaginemos un plugin para mostrar una [caja de diálogo como las que hacemos con jQuery UI](#).

Esas cajas de diálogo permiten mostrar mensajes en una capa emergente. Esa caja podría tener diversos parámetros para configurarla, como su altura, anchura, título de la caja, etc. Todos esos parámetros podríamos enviarlos al dar de alta la caja, con un código como este:

```
$("#capa").crearCaja(400, 200, "titulo", ...);
```

Pero eso no es práctico, porque el usuario debería indicar todos los parámetros para crear la caja, o al menos si no indica unos no podría indicar otros que están detrás en la lista. Luego, en el código del plugin, el desarrollador debería comprobar qué parámetros se indican, uno a uno, y darles valores por defecto si no se han indicado, etc. Todo eso ampliaría demasiado el código fuente.

Entonces, lo que se suele hacer al dar de alta el plugin, es indicar una serie de datos con notación de objeto:

```
$("#capa").crearCaja({
  titulo: "titulo",
  anchura: 400,
  altura: 200,
  ...
});
```

El desarrollador del plugin colocará en el código fuente un objeto con las variables de configuración y sus valores por defecto. Luego, cuando se cree el plugin, lo mezclará con el objeto de options enviado por parámetro, con una única sentencia, con lo que obtendrá rápidamente el objeto completo de configuración del plugin que debe ser aplicado.

6.4.2.- Definir opciones por defecto en el código del plugin

Con el siguiente código podemos definir las variables de configuración por defecto de un plugin y combinarlas con las variables de options enviadas por parámetro al invocar el plugin.

```
jQuery.fn.miPlugin = function(cualquierCosa, opciones) {
  //Defino unas opciones por defecto
  var configuracion = {
    dato1: "lo que sea",
    dato2: 78
  }
  //extiende las opciones por defecto con las recibidas
  jQuery.extend(configuracion, opciones);

  //resto del plugin
  //donde tenemos la variable configuracion para personalizar el plugin
}
```

La función principal del plugin recibe dos parámetros, uno "cualquierCosa" y otro "opciones". El primero supongamos que es algo que necesita el plugin, pero la configuración, que es lo que nos importa ahora, se ha recibido en el parámetro "opciones".

Ya dentro de la función del plugin, se define el objeto con las opciones de configuración, con sus valores por defecto, en una variable llamada "configuracion".

En la siguiente línea se mezclan los datos de las opciones de configuración por defecto y las recibidas por el plugin al inicializarse. Luego podremos acceder por medio de la variable "configuracion" todas las opciones del plugin que se va a iniciar.

Nota: El modo en cómo se mezclan los datos por medio de `extend()`, podéis revisar en el artículo sobre el [método jQuery.extend\(\)](#).

6.4.3.- Invocar al plugin enviando el objeto de opciones

Ahora podemos ver el código que utilizaríamos para invocar al plugin pasando las opciones que deseamos:

```
$("#elemento").miPlugin({
  dato1: "Hola amigos!",
  dato2: true
});
```

O podríamos enviar sólo alguno de los datos de configuración, para que el resto se tomen por defecto:

```
$("<div></div>").miPlugin({
  dato2: 2.05
});
```

O no enviar ningún dato al crear el plugin para utilizar los valores por defecto en todas las opciones de configuración.

```
$("p").miPlugin();
```

6.4.4.- Conclusión sobre la configuración de plugins con el objeto de opciones

Hasta el momento no hemos visto más que un código parcial de lo que sería un plugin con options para su configuración. Pero esperamos haber despejado ya algunas dudas. No obstante, veremos mejor cómo funciona todo por medio de un ejemplo en un artículo siguiente.

Continuar la lectura con el ejercicio [Plugin Tip con opciones en jQuery](#).

Artículo por Miguel Angel Alvarez

6.5.- Plugin Tip con opciones en jQuery

Un ejemplo de plugin en jQuery para hacer un sistema de tip más avanzado, que permite configurarse por medio de unas opciones en el plugin.

Hace poco tiempo publicamos un artículo en DesarrolloWeb.com sobre un plugin para mostrar un tip con un mensaje que aparecería en una capa al pasar el ratón sobre un elemento caliente. Eso es lo que llamamos un tip y lo explicamos en el artículo [Plugin jQuery para hacer un Tip simple](#).

Ahora vamos a hacer una modificación en ese plugin para implementar una serie de opciones, que nos permitirán configurar de una manera más versátil el comportamiento del plugin. Las opciones que vamos a implementar serán las siguientes:

- Velocidad de la animación de mostrar y ocultar el tip
- Animación a utilizar para mostrar el tip
- Animación a utilizar para ocultar el tip
- Clase CSS para la capa del tip

Todas esas opciones se definen, junto con los valores por defecto que van a tomar, al crear el código del plugin. En el

anterior artículo ya explicamos de manera general [cómo funciona el sistema de options en plugins](#), que vamos a utilizar a continuación.

Comenzamos por especificar, con notación de objeto, las opciones de configuración por defecto para el plugin:

```
var configuracion = {
  velocidad: 500,
  animacionMuestra: {width: "show"},
  animacionOculta: {opacity: "hide"},
  claseTip: "tip"
}
```

Ahora veamos el inicio del código del plugin, donde debemos observar que en la función que define el plugin se están recibiendo un par de parámetros. El primero es el texto del tip, que necesitamos para crear la capa del tip (Este parámetro ya aparecía en el código del plugin del [artículo anterior](#)). El segundo son las opciones específicas para configurar el plugin.

```
jQuery.fn.creaTip = function(textoTip, opciones) {
  //opciones por defecto
  var configuracion = {
    velocidad: 500,
    animacionMuestra: {width: "show"},
    animacionOculta: {opacity: "hide"},
    claseTip: "tip"
  }
  //extiendo las opciones por defecto con las opciones del parámetro.
  jQuery.extend(configuracion, opciones);

  this.each(function(){
    //código del plugin
  });
});
```

6.5.1.- Método jQuery.extend()

Quizás en este código, lo que más nos llame la atención sea el lugar donde extiendo las opciones por defecto definidas en la variable "configuracion", con las opciones específicas para el plugin concreto, recibidas por medio del parámetro "opciones".

```
jQuery.extend(configuracion, opciones);
```

Esta sentencia es una llamada al método extend() que pertenece a jQuery. Esta función recibe cualquier número de parámetros, que son objetos, y mete las opciones de todos en el primero. Luego, después de la llamada a extend(), el objeto del primer parámetro tendrá sus propiedades más las propiedades del objeto del segundo parámetro. Si alguna de las opciones tenía el mismo nombre, al final el valor que prevalece es el que había en el segundo parámetro. Si tenemos dudas con respecto a este método, leer el artículo [jQuery.extend\(\)](#).

Así, podemos ver cómo con extend() las propiedades por defecto del plugin se combinan con las que se envíen en las opciones. Luego, en el código del plugin, podremos acceder a las propiedades a través de la variable configuración, un punto y el nombre de propiedad que queramos acceder.

```
configuracion.velocidad
```

6.5.2.- Código completo del plugin tip con opciones

Veamos todo el código de nuestro primer plugin en implementar el sistema de opciones:

```
jQuery.fn.creaTip = function(textoTip, opciones) {
  var configuracion = {
    velocidad: 500,
    animacionMuestra: {width: "show"},
    animacionOculta: {opacity: "hide"},
    claseTip: "tip"
  }
  jQuery.extend(configuracion, opciones);

  this.each(function(){
```

```
elem = $(this);
var miTip = $('<div class="' + configuracion.claseTip + '">' + textoTip + '</div>');
$(document.body).append(miTip);
elem.data("capatip", miTip);

elem.mouseenter(function(e){
    var miTip = $(this).data("capatip");
    miTip.css({
        left: e.pageX + 10,
        top: e.pageY + 5
    });
    miTip.animate(configuracion.animacionMuestra, configuracion.velocidad);
});
elem.mouseleave(function(e){
    var miTip = $(this).data("capatip");
    miTip.animate(configuracion.animacionOculta, configuracion.velocidad);
});
});

return this;
};
```

6.5.3.- Invocar al plugin con o sin las opciones de configuración

Para acabar, vamos a invocar al plugin del tip con opciones, pero lo vamos a hacer de dos modos, uno con las opciones por defecto y otro con opciones específicas.

Así se llamaría al plugin con las opciones por defecto:

```
$("#elemento1").creaTip("todo bien...");
```

En realidad le estamos pasando un parámetro, pero no son las opciones, sino es el texto que tiene que aparecer en el tip. Como no se indican opciones, ya que no hay segundo parámetro, se toman todas las definidas por defecto en el plugin.

Las opciones, según se puede ver en el código del plugin, se deberían enviar en un segundo parámetro cuando se llama al plugin, tal como se puede ver a continuación:

```
$("#elemento2").creaTip("Otra prueba...", {
    velocidad: 1000,
    claseTip: "otroestilotip",
    animacionMuestra: {
        opacity: "show",
        padding: '25px',
        'font-size': '2em'
    },
    animacionOculta: {
        height: "hide",
        padding: '5px',
        'font-size': '1em'
    }
});
```

Ahora hemos indicado varias opciones específicas, que se tendrán en cuenta al crear el plugin con este segundo código.

Para acabar, dejamos un enlace para [ver el ejemplo en funcionamiento](#).

Artículo por Miguel Ángel Álvarez

6.6.- Funciones y variables dentro de plugins jQuery

Las funciones en los plugins pueden verse como funciones y variables privadas del plugin, que nos sirven para definir una mejor lógica de programación y estructura de datos y código.

Igual que en cualquier lenguaje de programación, podemos utilizar funciones para estructurar el código de nuestros plugins. Del mismo modo, en los plugins podemos tener variables, donde guardar datos u objetos que tengan validez dentro del ámbito de ese plugin. Todo ello nos ayudará bastante a hacer un código más claro, autónomo y compacto.

En este artículo vamos a mostrar cómo sería el esquema de programación de un plugin jQuery un poco más avanzado, que incluye las mencionadas variables y funciones "locales" (o "privadas", como las queramos llamar). Veremos también cómo es posible acceder a esas variables y funciones desde cualquier parte del flujo de código de un plugin, incluso desde el código de otras funciones, como los eventos.

6.6.1.- Esquema de programación de un plugin

A continuación vamos a mostrar el esquema de código de un plugin que incluye funciones y variables. Este plugin no sirve para nada, simplemente es una prueba que estoy realizando para ver cómo puedo crear esas variables, acceder a ellas y comprobar su ámbito en distintos puntos del código del plugin.

```
jQuery.fn.miPlugin = function() {

    //variables que son comunes a todos los elementos
    //que había en el objeto jQuery que recibe el método del plugin
    mivariableComun = "comun";
    alert("Nueva invocación de plugin. Mi variable común: " + mivariableComun)

    this.each(function() {
        //CÓDIGO DEL PLUGIN

        //Elemento sobre el que itero y estoy aplicando el plugin
        elem = $(this);
        //elem es una variable que podremos utilizar en todo el plugin

        //variables específicas para cada elemento
        var miVariable = "x";
        //miVariable se podrá acceder dentro de todo el código que pongamos aquí

        //funcion que será accesible desde cualquier parte del plugin
        function miFuncion(){
            //puedo acceder a variables del plugin
            miVariable = elem.text();

            //Muestro el contenido de la variable
            alert("mi variable local y particular de cada plugin: " + miVariable);

            //cambio la variable comun a todos los elementos de este plugin
            mivariableComun = "Otra cosa comun!"
        }

        //puedo invocar las funciones del plugin
        miFuncion();

        //evento, que tiene una función. Puedo acceder a variables y funciones del plugin
        elem.click(function(){
            //puedo acceder a variables del plugin
            alert("Dentro del evento: " + miVariable);

            //puedo acceder a funciones
            miFuncion();
        });
    });
};
```

Para definir esas variables y funciones locales al plugin, de manera que estén accesibles dentro del plugin y a su vez tengan acceso a todos los datos del mismo, debemos colocarlas dentro de la iteración que se hace con `this.each()`.

Como se puede entender del código del plugin anterior, todas esas variables y funciones se pueden invocar o acceder en cualquier parte, siempre y cuando estemos dentro del `this.each()`, donde fueron creadas.

Este plugin, aunque no valga para mucho, lo hemos publicado y se puede [ver en marcha en una página aparte](#).

Nota: Antes del `this.each()` se pueden colocar también variables, pero tenemos que tener en cuenta que existirá una misma copia de esa variable para todos los elementos donde se está aplicando el plugin.

Veamos el siguiente HTML.

```
<div id="esteDiv">
  Este div
</div>
<span class="misspan">span1</span>
<span class="misspan">span2</span>
<span class="misspan">span3</span>
```

Ahora veamos estas dos llamadas al plugin anterior.

```
$("#esteDiv").miPlugin();
$(".misspan").miPlugin();
```

Como se puede ver, con la primera llamada se ejecuta el plugin sobre un elemento de la página con `id="esteDiv"`. Es un único elemento de la página, luego el plugin sólo se aplica una vez. Sin embargo, en la segunda llamada, se ejecuta el plugin sobre varios elementos con la class de CSS "misspan". En este segundo caso el plugin se ejecutará sobre tres elementos y entonces podremos comprobar que las variables que se habían definido fuera del `this.each()` sólo existen una vez y su valor es común para los tres elementos sobre los que se aplicó el plugin en la segunda llamada.

Lo cierto es que quizás todo esto quede un poco confuso, o no se entienda muy bien para qué podremos querer todas esas variables y funciones locales al plugin. Pero a medida que vayamos trabajando y planteándonos plugins más complicados, veremos que nos son de mucha utilidad para almacenar datos a los que queremos acceder más tarde, o para organizar el código de nuestro plugin en distintas funciones, que se pueden llamar repetidas veces y desde varios sitios.

Para los que conocen un poco de programación orientada a objetos, quizás les aclare un poco este simil: Si un plugin fuese como un objeto, las variables de dentro del bloque `this.each()` de los plugins serían como las propiedades de ese objeto y las funciones serían como métodos de ese objeto. Aunque hay que salvar las distancias, porque un plugin no sigue exactamente el modelo que conocemos en las clases de programación orientada a objetos.

Para que veamos un caso práctico de plugin que tiene varias variables y funciones locales hemos realizado el ejemplo del siguiente artículo: [Checkbox con diseño personalizado con jQuery](#).

Artículo por *Miguel Angel Alvarez*

6.7.- Plugin checkbox personalizado con jQuery

Un plugin en jQuery para hacer un campo de formulario checkbox pero con un diseño distinto, totalmente personalizable por el desarrollador.

A veces los campos de formulario que tenemos disponibles en HTML son un poco "aburridos", por decirlo de alguna manera. Quiero decir que son siempre iguales para todas las páginas y existen pocas opciones para configurar su aspecto, sobre todo en el caso de los elementos checkbox. Como diseñadores caprichosos, nosotros podríamos desear que nuestros checkboxes tuvieran un aspecto o color determinado, que haga mejor combinación con otros elementos de nuestro layout. Y estas son cosas que podemos conseguir fácilmente con un poco de jQuery.

En este artículo pretendemos hacer un plugin para crear campos checkbox personalizados, con las mismas funcionalidades de los checkbox normales, pero que tengan un aspecto configurable por el desarrollador. Para ello utilizaremos el modelo de creación de plugins en jQuery, de modo que haremos todo el trabajo en un plugin que cualquier persona podría utilizar en su sitio web y configurar los checkbox según sus preferencias.

Para seguir las explicaciones de este artículo necesitaremos saber acerca de la creación de [plugins en jQuery](#) y en concreto

vamos a practicar con dos cosas que hemos aprendido recientemente:

- [Gestión de opciones en plugins jQuery](#)
- [Crear Funciones y variables dentro de plugins jQuery](#)

Para apreciar con exactitud cómo serán algunos ejemplos de checkbox que vamos a realizar, podemos [echar un vistazo al ejemplo en marcha](#).

6.7.1.- Personalización del plugin por medio de objeto de opciones

Podemos comenzar por ver el principio de código del plugin, donde estamos definiendo las variables de configuración por defecto y las estamos extendiendo con las variables de configuración definidas al invocarlo.

```
jQuery.fn.checkboxPersonalizado = function(opciones) {
    //opciones de configuración por defecto
    var conf = {
        activo: true,
        colorTextos: {
            activo: "#00f",
            pasivo: "#669"
        },
        textos: {
            activo: "",
            pasivo: ""
        },
        imagen: {
            activo: 'images/thumb_up.png',
            pasivo: 'images/thumb_down.png'
        },
        cssElemento: {
            padding: "2px 2px 2px 24px",
            display: "block",
            margin: "2px",
            border: "1px solid #eee",
            cursor: "pointer"
        },
        cssAdicional: {

        },
        nameCheck: ""
    };
    //Las extiendo con las opciones recibidas al invocar el plugin
    jQuery.extend(conf, opciones);

    this.each(function() {

        //CÓDIGO DEL PLUGIN

    });
    return this;
};
```

Tal como se puede ver, se han definido varias variables para configurar el objeto, que se dispondrán en un objeto que tenemos en la variable "configuracion". Entre las variables de configuración tenemos una llamada "activo" con un valor booleano para decidir si el elemento checkbox estaría o no seleccionado desde el principio. Tenemos una variable "colorTextos", para definir el color del texto cuando el elemento está activo y pasivo. También tenemos otra serie de configuraciones para los estados de activo y pasivo (seleccionado o no seleccionado), como la imagen que se tiene que mostrar al lado del texto.

Ahora veamos el código del plugin, lo que iría dentro de this.each(). Recordemos que cada variable creada aquí es accesible dentro de todo el bloque de código definido por las llaves del this.each(). Así mismo, las funciones declaradas aquí son accesibles desde cualquier parte de este bloque.

```
//variables locales al plugin
var miCheck = $(this);
```

```
var activo = conf.activo
//el elemento checkbox interno pero no visible
var elementoCheck = $('<input type="checkbox" style="display: none;" />');
//el nombre del checkbox puede ser configurado desde options o con el propio texto del campo
if(conf.nameCheck==""){
    elementoCheck.attr("name", miCheck.text());
}else{
    elementoCheck.attr("name", conf.nameCheck);
}
//inserto el checkbox en la página
miCheck.before(elementoCheck);
//aplico estilos que vienen en la configuración
miCheck.css(conf.cssElemento);
miCheck.css(conf.cssAdicional);

//si el elemento estaba marcado para estar activo
if (activo){
    //lo activo
    activar();
}else{
    //lo desactivo
    desactivar();
}

//defino un evento para el elemento
miCheck.click(function(e) {
    //compruebo la variable activo, definida dentro del plugin
    if(activo){
        desactivar();
    }else{
        activar();
    }
    activo = !activo;
});

//función local en el plugin para desactivar el checkbox
function desactivar(){
    //cambio los estilos para el elemento a los marcados como pasivos
    miCheck.css({
        background: "transparent url(" + conf.imagen.pasivo + ") no-repeat 3px",
        color: conf.colorTextos.pasivo
    });
    //si hay un texto específico para cuando estaba pasivo
    if (conf.textos.pasivo!=""){
        miCheck.text(conf.textos.pasivo)
    }
    //desmarcho el checkbox interno que es invisible, pero que se envía como elemento de formulario.
    elementoCheck.removeAttr("checked");
};

function activar(){
    miCheck.css({
        background: "transparent url(" + conf.imagen.activo + ") no-repeat 3px",
        color: conf.colorTextos.activo
    });
    if (conf.textos.activo!=""){
        miCheck.text(conf.textos.activo)
    }
    elementoCheck.attr("checked", "1");
};
```

El código está convenientemente comentado para que se pueda entender mejor. Pero lo que queremos mostrar en este caso es que hemos creado dos funciones dentro del código del plugin: `activar()` y `desactivar()`. Esas dos funciones, al estar dentro del bloque `this.each()`, se pueden acceder desde cualquier parte del plugin y comparten el mismo ámbito de variables que el propio plugin, luego podremos acceder desde ellas a cualquier variable definida en el bloque `this.each()`.

Para que quede un poco más clara la estructura completa del plugin, coloco a continuación su código completo:

```
jQuery.fn.checkboxPersonalizado = function(opciones) {
    //opciones de configuración por defecto
```

```
var conf = {
  activo: true,
  colorTextos: {
    activo: "#00f",
    pasivo: "#669"
  },
  textos: {
    activo: "",
    pasivo: ""
  },
  imagen: {
    activo: 'images/thumb_up.png',
    pasivo: 'images/thumb_down.png'
  },
  cssElemento: {
    padding: "2px 2px 2px 24px",
    display: "block",
    margin: "2px",
    border: "1px solid #eee",
    cursor: "pointer"
  },
  cssAdicional: {

  },
  nameCheck: ""
};
//Las extiendo con las opciones recibidas al invocar el plugin
jQuery.extend(conf, opciones);

this.each(function(){
  var miCheck = $(this);
  var activo = conf.activo
  var elementoCheck = $('<input type="checkbox" style="display: none;" />');
  if(conf.nameCheck==""){
    elementoCheck.attr("name", miCheck.text());
  }else{
    elementoCheck.attr("name", conf.nameCheck);
  }
  miCheck.before(elementoCheck);
  miCheck.css(conf.cssElemento);
  miCheck.css(conf.cssAdicional);

  if (activo){
    activar();
  }else{
    desactivar();
  }
  miCheck.click(function(e){
    if(activo){
      desactivar();
    }else{
      activar();
    }
    activo = !activo;
  });

  function desactivar(){
    miCheck.css({
      background: "transparent url(" + conf.imagen.pasivo + ") no-repeat 3px",
      color: conf.colorTextos.pasivo
    });
    if (conf.textos.pasivo!=""){
      miCheck.text(conf.textos.pasivo)
    }
    elementoCheck.removeAttr("checked");
  };

  function activar(){
    miCheck.css({
      background: "transparent url(" + conf.imagen.activo + ") no-repeat 3px",
```

```

        color: conf.colorTextos.activo
    });
    if (conf.textos.activo!=""){
        miCheck.text(conf.textos.activo)
    }
    elementoCheck.attr("checked","1");
    };
    });
    return this;
};

```

6.7.2.- Invocar al plugin checkbox personalizado con jQuery

Ya que hemos hecho un checkbox personalizado, por un objeto de options, vamos a mostrar cómo se pueden crear varios tipos de checkbox con este código. Veamos el siguiente HTML:

```

<span class="ch">Seleccionar</span>
<span class="ch">Me interesa</span>
<span class="ch">Oooo</span>
<br>
<br>
<span id="otro">otro suelto</span>

```

Se puede apreciar que tenemos simples elementos SPAN. Por un lado tenemos 3 SPAN con la clase "ch" y por otro lado otro SPAN suelto con identificador "otro". Ahora veamos cómo los convertiríamos en campos de formulario checkbox personalizados:

```

$(".ch").checkboxPersonalizado();

```

Así crearíamos 3 checkbox, en los 3 primeros SPAN que tenían la class "ch". Estos checkbox personalizados se crearían con las opciones por defecto.

```

$("#otro").checkboxPersonalizado({
    activo: false,
    colorTextos: {
        activo: "#f80",
        pasivo: "#98a"
    },
    imagen: {
        activo: 'images/weather_cloudy.png',
        pasivo: 'images/weather_rain.png'
    },
    textos: {
        activo: 'Buen tiempo :)',
        pasivo: 'Buena cara ;)'
    },
    cssAdicional: {
        border: "1px solid #dd5",
        width: "100px"
    },
    nameCheck: "buen_tiempo"
});

```

En este segundo caso de invocación al plugin estamos convirtiendo en un checkbox personalizado el último SPAN, que tenía identificador "otro". En este segundo caso estamos utilizando multitud de variables de configuración específicas, que harán que el checkbox tenga un aspecto radicalmente diferente a los anteriores.

Para acabar, se puede [ver el ejemplo en funcionamiento en una página aparte](#).

Artículo por Miguel Angel Alvarez

6.8.- Alias personalizado y ocultar código en plugins jQuery

Cómo crear un alias personalizado a \$, para mejorar la compatibilidad en todos los contextos, y ocultar

el código privado de los plugins jQuery.

En principio un plugin jQuery está pensado para que todas las personas lo puedan usar en sus páginas web y de hecho, hemos aprendido muchas convenciones para potenciarlo a lo largo de los capítulos dedicados a la creación de plugins en el [Manual de jQuery](#). En este artículo vamos a aprender otra buena práctica que ayudará a que nuestros plugins funcionen correctamente en todos los sitios web.

Se trata de ocultar de una manera sencilla todo el código de nuestros plugins y utilizar un alias para la variable \$ que puede dar conflictos con otras librerías. Algo que nos ayudará de dos maneras:

El símbolo \$ se utiliza en muchos otros frameworks y componentes Javascript y si el web donde se coloque el plugin utiliza alguno de ellos, pueden ocurrir conflictos, algo que no ocurrirá en el caso que utilicemos un alias.

En el código de los plugins puedes utilizar tus propias variables o funciones, que tendrán el nombre que hayas querido. Pero alguno de esos nombres puede que ya los utilicen otros programadores en sus páginas, lo que puede generar conflictos también. Por eso no es mala idea ocultar tu código y hacerlo local a un ámbito propio.

Todo esto se consigue colocando todo tu código dentro de una función que se invoca según se declara.

```
(function() {  
    //Código de tu plugin  
  
    //puedes crear tus variables o funciones  
    //sólo serán visibles aquí  
    var loquesea;  
    function algo(){  
  
    }  
})(); //la función se ejecuta instantáneamente
```

Además, a esa función podríamos enviarle la variable "jQuery" que contiene toda la funcionalidad del framework. Esa variable la recibirás en el parámetro con cualquier alias, como se puede ver en el siguiente código:

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

Como la variable jQuery siempre es una referencia al framework correcta, puedes estar seguro que no tendrá conflictos con otras librerías. Luego la recibimos con el nombre \$, pero en ese caso ya estamos en el ámbito de la función, donde las variables locales pueden tener el nombre que nosotros queramos.

Nota: En este caso estamos recibiendo la variable jQuery con el nombre \$, pero podríamos utilizar cualquier otro nombre para el alias a jQuery.

6.8.1.- Conclusión: una envoltura segura y sencilla para tus plugins

Como podemos entender, colocar esa envoltura en tus plugins, no interfiere en nada a cómo se tienen que diseñar y todo lo que hemos aprendido anteriormente sobre creación de plugins se puede aplicar a este pequeño pero interesante truco. No obstante, para completar las informaciones, a continuación se puede ver un plugin donde realizamos un plugin utilizando esta técnica para ocultar el código y utilizar el alias de \$.

Como hemos visto en este artículo, utilizar esa envoltura para nuestros plugins es una tarea simple, no requiere mucho código y las ventajas son importantes.

Artículo por *Miguel Angel Alvarez*

Parte 7:

Efectos en jQuery

Los efectos son una de las partes más atractivas del framework y que permitirán dotar de dinamismo a nuestra página, hacerla más atractiva y en definitiva, mejorar la experiencia del usuario.

7.1.- jQuery animate(): Animación de propiedades CSS

El método animate() de jQuery permite animar varias propiedades, con valores numéricos, de CSS en un solo paso.

Hemos avanzado bastante en el [Manual de jQuery](#) y con los conocimientos adquiridos hasta el momento ya estamos en disposición de aprender cualquier cosa más avanzada en este framework Javascript. 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 pasados artículos de DesarrolloWeb.com ya mostramos algunas maneras de hacer [efectos sencillos en jQuery](#) y en adelante vamos a explicar el funcionamiento de otros métodos, más complejos pero también más versátiles.

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.

7.1.1.- Parámetros del método animate()

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 de debe indicar 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. Generamente, 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. En los ejemplos de este y siguientes artículos que publiquemos en desarrolloweb.com veremos varias maneras de indicar las propiedades para realizar varias animaciones.

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.

7.1.2.- Ejemplo jQuery del método animate()

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': "20",
  '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.

Pero veamos una página que hace uso de ese método, con el código completo. Como veremos, en la página tendremos además dos enlaces, uno para poner en marcha la animación y otro para restaurar el CSS de los elementos a los valores originales. Así que, de paso que vemos como hacer un animate(), aprenderemos además a lanzar la ejecución de las animaciones como respuesta a eventos de usuario.

```
<html>
<head>
<title>Método animate jQuery</title>
<script src="../jquery-1.4.1.min.js"></script>
<script>
$(document).ready(function() {
  $("#animar").click(function(e) {
    e.preventDefault()
    $("#h1").animate({
      'border-bottom-width': "20",
      'font-size': '25pt'
    });
  });

  $("#restaurar").click(function(e) {
    e.preventDefault()
    $("#h1").css({
      'border-bottom-width': "1",
      '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>
```

Este ejemplo puede [verse en marcha en una página aparte](#).

Artículo por *Miguel Ángel Álvarez*

7.2.- Fading en jQuery

Efectos de cambio de opacidad de los elementos en la página, con los métodos de fading en jQuery, fadeIn(), fadeOut() y fadeTo().

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.

A lo largo del [Manual de jQuery](#) que venimos publicando en DesarrolloWeb.com hemos utilizado alguno de estos métodos para hacer efectos rápidos en jQuery, pero ahora los vamos a explicar de manera más detenida. Además, realizaremos nuevas prácticas con estos tipos de efectos de cambio de opacidad y trabajaremos con las funciones callback para realizar una pequeña cadena de efectos, que se ejecutan cuando los anteriores hayan acabado.

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 opacidad 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.

7.2.1.- Ejemplos con efectos de fundido fadeOut() y fadeIn() en jQuery

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.

Para hacernos una idea de lo que vamos a conseguir en este ejercicio, podemos [ver el ejemplo en marcha](#).

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();
```

7.2.2.- Ejemplo con `fadeTo()`

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 opacidad 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 queremos 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.

7.2.3.- Enviando funciones callback

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 un poco más adelantado, en el que se encadenan varias funciones callback, que se ejecutarían una detrás de la otra.

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

En este código hacemos un efecto de `fadeOut()` sobre cada uno de los elementos 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`.

7.2.4.- Código completo del ejemplo de fading en jQuery

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

```
<!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 src="../jquery-1.4.2.min.js"></script>
<script>
$(document).ready(function(){
    $("#ocultartoda").click(function(e){
        $("#milista").fadeOut();
    });
    $("#mostrartoda").click(function(e){
        $("#milista").fadeIn();
    });
    $("#ocultarmostrartoda").click(function(e){
        $("#milista").fadeOut(function(){
            $(this).fadeIn();
        });
    });
    $("#selopacidad").change(function(e){
        var opacidad_deseada = e.target.options[e.target.selectedIndex].value;
        $("#h1").fadeOut(1000, opacidad_deseada);
    });
    $("#pororden").click(function(e){
        var opacidad_deseada = $("#selopacidad").attr("value");
        $("#e1").fadeOut(500, opacidad_deseada, function(){
            $("#e2").fadeOut(500, opacidad_deseada, function(){
                $("#e3").fadeOut(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>
```

Si lo deseamos, podemos [ver el ejemplo en marcha en una página aparte](#).

Artículo por *Miguel Ángel Álvarez*

7.3.- Colas de efectos en jQuery

Vamos a explicar qué es una cola de efectos, para qué nos sirve y cómo se configuran las colas de efectos en el framework Javascript jQuery.

En el [Manual de jQuery](#) hemos tratado ya en diversos artículos de los efectos en jQuery. De hecho, éste ya es el cuarto artículo que destinamos a tratar las distintas maneras de crear efectos en este framework Javascript. Hasta la fecha hemos publicado las claves para la [creación de efectos simples](#), el versátil método [animate\(\) para realizar la animación de atributos CSS numéricos](#), o las [funciones de para crear efectos de fundido](#).

Todos estos métodos tratados anteriormente, y algunos más que no hemos revisado todavía como `slideUp()` o `slideDown()`, 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 hemos superado este primer paso y ya sabemos hacer toda una gama de efectos simples, vamos a aprender a encadenar varios efectos a ejecutar uno detrás de otro.

Veremos en este artículo y varios que sucederán, 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.

7.3.1.- Funciones de efectos

Vamos a repetir a lo largo de los siguientes artículos un concepto que quiero explicar para que se sepa a qué nos referimos. Se trata de las "Funciones de efectos" que son aquellas que dispone jQuery para crear efectos especiales en páginas web. Como hemos dicho, en diversos artículos anteriores ya se han explicado y mostrado efectos de diversas de las funciones de efectos disponibles.

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()`, `slideUp()` y `slideDown()`, 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.

En este [Manual de jQuery](#) ya hemos visto varios ejemplos sobre estas funciones de efectos y a lo largo de los próximos artículos que publicaremos en desarrolloweb.com veremos diversas otras aplicaciones de muestra donde podremos seguir aprendiendo.

7.3.2.- Cola de efectos por defecto

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.

7.3.3.- Ejemplo de ejecución de efectos en la cola predeterminada de jQuery

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:

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

El resultado del ejercicio completo se puede [ver en una página aparte](#).

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. En el próximo artículo [empezaremos a explicar el modos existentes 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.

Artículo por *Miguel Ángel Álvarez*

7.4.- Método queue() para acceder a una cola de efectos

Veremos cómo hacer cosas con las colas de efectos en jQuery, haciendo nuestra primera prueba con el método queue(), que permite acceder y modificar la cola de efectos.

En el artículo anterior del [Manual de jQuery](#) empezamos a hablar sobre [las colas de efectos](#). Vimos que crear una cola de efectos es una tarea muy sencilla, básicamente porque jQuery gestiona de manera automática la cola de efectos predeterminada. 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 en futuros artículos.

7.4.1.- Método queue([nombreCola])

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.

Si lo deseamos, antes de ponernos a comentar este ejemplo, podemos [ver el ejercicio en marcha que vamos a construir](#).

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.

```
<!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 language="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>
```

Ahora, para acabar, podemos [ver el ejercicio en marcha en una página aparte](#).

En el siguiente artículo continuaremos con el trabajo con colas de efectos y aprenderemos a [encolar funciones que no son las de efectos de jQuery](#), de modo que podremos meter nuestras propias funciones en la cola, con cualquier tipo de instrucción.

Artículo por *Miguel Ángel Álvarez*

7.5.- Meter cualquier tipo de función en una cola de efectos jQuery

En la cola de efectos podemos introducir cualquier tipo de función, aunque no sean efectos jQuery, y para ello vamos a aprender a encolar cualquier conjunto de sentencias con el método `queue()`.

En estos momentos se supone que sabemos [cómo introducir funciones de efectos en las colas de efectos de jQuery](#), ya que en los artículos anteriores del [Manual de jQuery](#) ya habíamos comenzado a analizar cómo funcionaban.

Pero ¿Qué pasa si queremos encolar otro tipo de función Javascript o jQuery? 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.

Nota: El método `queue()` funciona de maneras distintas dependiendo de los parámetros que le enviemos. En el anterior artículo ya empezamos a explicar [cómo utilizar queue\(\) para acceder a una cola de efectos](#).

7.5.1.- `.queue([nombreCola], callback(continua))`

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

- 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. Más adelante explicaremos cómo trabajar con colas distintas de 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();
});
```

7.5.2.- Ejemplo jQuery para encolar funciones que no son efectos

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:

```
<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 (jQuery
1.4)
        continua();
    });
    capa.animate({
        "height": "50px",
        "width": "400px"
    }, 1000);
});
```

El resultado de ejecutar este código Javascript se puede [ver en una página aparte](#).

En el siguiente artículo veremos el último uso que nos queda por explicar del método `queue()` y de paso, otro método interesante, `stop()`, que sirve para detener la ejecución de una cola.

Artículo por *Miguel Angel Alvarez*

7.6.- Método jQuery.extend()

Uno de los métodos del paquete *utilities* de jQuery, que sirve para extender el contenido de dos o más

objetos en uno de ellos.

Vamos a ofrecer una referencia rápida para el método `jQuery.extend()`, que nos ofrece una utilidad para mezclar mezclar varios objetos en uno, es decir, colocar los contenidos de todos esos objetos en uno de ellos.

El método `extend` pertenece a la clase `jQuery`, y se invoca directamente sobre ella, como si fuera un método estático de programación orientada a objetos. Como en `jQuery` la variable `$` es un atajo de la variable `jQuery`, podríamos invocar a este método con estas dos posibilidades de código:

```
jQuery.extend(x, y, z);
```

O bien:

```
$.extend(x, y, z);
```

Estos dos ejemplos de código harían exactamente lo mismo, colocar en el objeto "x" todos los contenidos de los objetos "x", "y" y "z". El método `extend()` puede recibir cualquier número de parámetros y siempre pondrá todos los contenidos de los objetos en el objeto recibido en el primer parámetro.

7.6.1.- Ejemplo de `extend()` de `jQuery`

Veamos cómo funciona `jQuery.extend()` a través de un sencillo ejemplo.

```
var a = {
  uno: "hola",
  otro: "adios"
};
var b = {
  uno: "otra cosa",
  dos: "loquesea"
};
jQuery.extend(a, b);
```

En este caso `extend()` recibe dos parámetros, que son dos objetos. Por tanto, mete las opciones del segundo objeto en el primero. Después de la llamada a `extend()`, el objeto del primer parámetro tendrá sus propiedades más las propiedades del objeto del segundo parámetro. Si alguna de las opciones tenía el mismo nombre, al final el valor que prevalece es el que había en el segundo parámetro.

Así pues, después de su ejecución, el objeto definido en la variable "a" tendrá estos datos:

```
{
  uno: "otra cosa",
  otro: "adios",
  dos: "loquesea"
}
```

Esto quizás parezca que no sirve para mucho, pero en `jQuery` se utiliza bastante por ser una manera cómoda de mezclar dos cosas en una. El caso más claro es mezclar los objetos de "options" para configurar un plugin, pero realmente es una acción que encontraremos por ahí varias veces. De alguna manera, hacer un `extend()` es como hacer que un objeto herede las cosas de otro, lo que lo convierte en un mecanismo que podrá venir bien en diversos contextos.

Para ver otros ejemplos de `extend()` consultar el [Manual de jQuery](#), en la sección donde se habla de la configuración de opciones en plugins en `jQuery`.

Artículo por *Miguel Angel Alvarez*