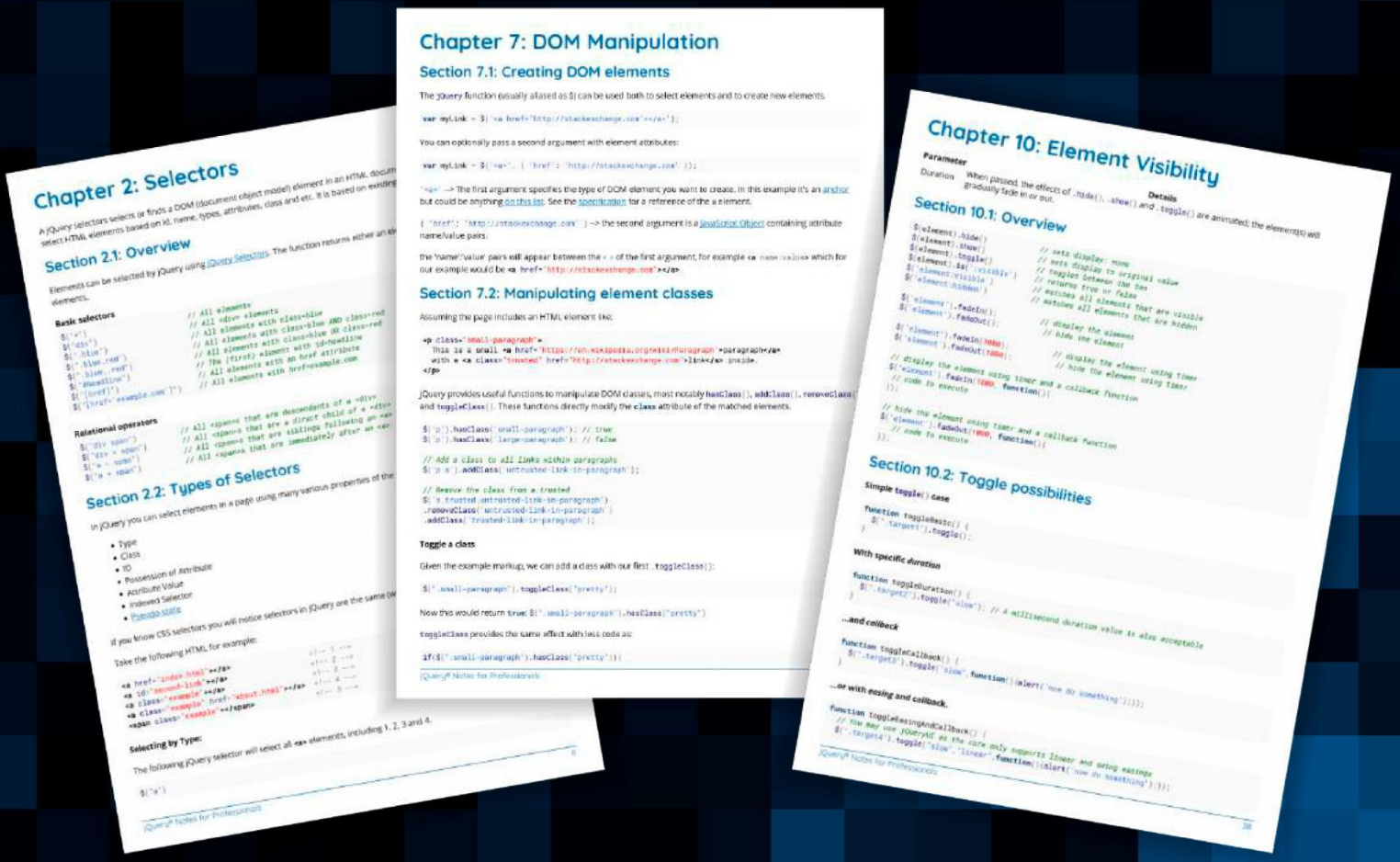
jQuery®

Apuntes para Profesionales



Traducido por:

rortegag

50+ páginas

de consejos y trucos profesionales

Contenidos

Acerca de

Este libro ha sido traducido por rortegag.com

Si desea descargar el libro original, puede descargarlo desde:

https://goalkicker.com/jQueryBook/

Si desea contribuir con una donación, hazlo desde:

https://www.buymeacoffee.com/GoalKickerBooks

Por favor, siéntase libre de compartir este PDF con cualquier persona de forma gratuita, la última versión de este libro se puede descargar desde:

https://goalkicker.com/jQueryBook/

Envíe sus comentarios y correcciones a web@petercv.com

Capítulo 1: Introducción a jQuery

Versión	Notas	Fecha de publicación
1.0	Primera versión estable.	26-08-2006
1.1		14-01-2007
1.2		10-09-2007
1.3	Sizzle introducido en el núcleo.	14-01-2009
1.4		14-01-2010
1.5	Gestión de retrollamadas diferidas, reescritura de módulos ajax.	31-01-2011
1.6	Aumento significativo del rendimiento de los métodos `attr()` y `val()`.	03-05-2011
1.7	Nuevas API de eventos: `on()` y `off()`.	03-11-2011
1.8	Sizzle reescrito, animaciones mejoradas y flexibilidad `$(html, props)`.	09-08-2012
1.9	Eliminación de interfaces obsoletas y limpieza de código.	15-01-2013
1.10	Se han incorporado correcciones de errores y diferencias notificadas en los ciclos beta 1.9 y 2.0.	24-05-2013
1.11		24-01-2014
1.12		08-01-2016
2.0	Se ha eliminado la compatibilidad con IE 6-8 para mejorar el rendimiento y reducir el tamaño.	18-04-2013
2.1		24-01-2014
2.2		08-01-2016
3.0	Aumento masivo de la velocidad de algunos selectores personalizados de jQuery.	09-06-2016
3.1	Se acabaron los errores silenciosos.	07-07-2016
3.2	Se acabaron los errores silenciosos.	16-03-2017
3.3	Se acabaron los errores silenciosos.	19-01-2018
3.4	Mejora del rendimiento en `.width` y `.height`. Compatibilidad con `nonce` y `nomodule`.	10-04-2019
3.5	Corrección de seguridad	10-04-2020
3.6	Devolución de JSON incluso para errores JSONP	02-03-2021
3.7	Mantenerse en orden	11-05-2023

Sección 1.1: Como empezar

Crea un archivo `hola.html` con el siguiente contenido:

```
<!DOCTYPE html>
<html>
      <head>
            <title>¡Hola, Mundo!</title>
      </head>
      <body>
            <div>
                  <p id="hola">Un texto al azar</p>
            </div>
            <script src="https://code.jquery.com/jquery-2.2.4.min.js"></script>
            <script>
                  $(document).ready(function() {
                        $('#hola').text('¡Hola, Mundo!');
                  });
            </script>
      </body>
</html>
```

Demostración en directo en JSBin.

Abre este archivo en un navegador web. Como resultado verá una página con el texto: `¡Hola, Mundo!`

Explicación del código

1. Carga la biblioteca jQuery desde la CDN de jQuery:

```
<script src="https://code.jquery.com/jquery-2.2.4.min.js"></script>
```

Esto introduce la variable global $, un alias para la función jQuery y el espacio de nombres.

> *Ten en cuenta que uno de los errores más comunes que se cometen al incluir jQuery es no cargar la biblioteca ANTES que cualquier otro script o biblioteca que pueda depender de ella o hacer uso de ella.*

2. Define una función que se ejecutará cuando jQuery detecte que el DOM (Document Object Model) está "ready":

```
// Cuando el `document` esté `ready`, ejecuta esta función `...`.
$(document).ready(function() { ... });
// Una versión abreviada de uso común (se comporta igual que la anterior)
$(function() { ... });
```

3. Una vez que el DOM está listo, jQuery ejecuta la función callback mostrada arriba. Dentro de nuestra función, sólo hay una llamada que hace 2 cosas principales:

    1. Obtiene el elemento con el atributo `id` igual a `hola` (nuestro selector `#hola`). El uso de un selector como argumento pasado es el núcleo de la funcionalidad y la nomenclatura de jQuery; toda la biblioteca evolucionó esencialmente a partir de la ampliación de `document.querySelectorAll` MDN.

    2. Establece el `text()` dentro del elemento seleccionado en `¡Hola, Mundo!`.

```
# ↓ - Pasar un `selector` a `$` jQuery, devuelve nuestro elemento
$('#hola').text('¡Hola, Mundo!');
# ↑ - Establecer el texto del elemento
```

Para más información, consulte la página jQuery – Documentación.

Sección 1.2: Evitar colisiones de espacios de nombres

Las bibliotecas distintas de jQuery también pueden utilizar `$` como alias. Esto puede causar interferencias entre esas bibliotecas y jQuery.

Para liberar `$` para su uso con otras bibliotecas:

```
jQuery.noConflict();
```

Después de llamar a esta función, `$` deja de ser un alias de `jQuery`. Sin embargo, puede seguir utilizando la variable `jQuery` para acceder a las funciones de jQuery:

```
jQuery('#hola').text('¡Hola, Mundo!');
```

Opcionalmente, puede asignar una variable diferente como alias para jQuery:

```
var jqy = jQuery.noConflict();
jqy('#hola').text('¡Hola, Mundo!');
```

Por el contrario, para evitar que otras librerías interfieran con jQuery, puede envolver su código jQuery en una expresión de función inmediatamente invocada (IIFE) y pasar `jQuery` como argumento:

```
(function($) {
     $(document).ready(function() {
          $('#hola').text('¡Hola, Mundo!');
     });
})(jQuery);
```

Dentro de este IIFE, `$` es un alias sólo para jQuery.

Otra forma sencilla de **asegurar el alias `$` de jQuery y asegurarse de que el DOM está listo:**

```
jQuery(function( $ ) { // DOM está listo
     // Ahora puedes utilizar $ alias
     $('#hola').text('¡Hola, Mundo!');
});
```

Resumiendo,

- `jQuery.noConflict()` : `$` ya no hace referencia a jQuery, mientras que la variable `jQuery` sí lo hace.
- `var jQuery2 = jQuery.noConflict()` - `$` ya no hace referencia a jQuery, mientras que la variable `jQuery` sí lo hace y también la variable `jQuery2`.

Ahora, existe un tercer escenario - ¿Qué pasa si queremos que jQuery esté disponible **sólo en `jQuery2`**? Usar,

```
var jQuery2 = jQuery.noConflict(true)
```

El resultado es que ni `$` ni `jQuery` hacen referencia a jQuery.

Esto es útil cuando se van a cargar varias versiones de jQuery en la misma página.

```
<script src='https://code.jquery.com/jquery-1.12.4.min.js'></script>
<script>
     var jQuery1 = jQuery.noConflict(true);
</script>
<script src='https://code.jquery.com/jquery-3.1.0.min.js'></script>
<script>
     // Aquí, jQuery1 se refiere a jQuery 1.12.4 mientras que, $ y jQuery se refiere a jQuery
     3.1.0.
</script>
```

https://learn.jquery.com/using-jquery-core/avoid-conflicts-other-libraries/

Sección 1.3: Espacio de nombres de jQuery ("jQuery" y "$")

`jQuery` es el punto de partida para escribir cualquier código jQuery. Se puede utilizar como una función `jQuery(...)` o una variable `jQuery.foo`.

`$` es un alias de `jQuery` y, por lo general, ambos pueden intercambiarse (excepto si se ha utilizado `jQuery.noConflict();` - véase Evitar colisiones de espacios de nombres).

Suponiendo que tengamos este fragmento de HTML –

```
<div id="demo_div" class="demo"></div>
```

Podríamos querer usar jQuery para añadir algo de contenido de texto a este div. Para ello podemos utilizar la función jQuery `text()`. Esto puede ser escrito usando `jQuery` o `$`. Por ejemplo –

```
jQuery("#demo_div").text("¡Texto de demostración!");
```

O –

```
$("#demo_div").text("¡Texto de demostración!");
```

Ambos darán como resultado el mismo HTML final -

```
<div id="demo_div" class="demo">¡Texto de demostración!</div>
```

Como `$` es más conciso que `jQuery`, generalmente es el método preferido para escribir código jQuery.

jQuery utiliza selectores CSS y en el ejemplo anterior se utilizó un selector ID. Para obtener más información sobre los selectores en jQuery consulte tipos de selectores.

Sección 1.4: Cargar jQuery por consola en una página que no lo tiene

A veces uno tiene que trabajar con páginas que no utilizan `jQuery` mientras que la mayoría de los desarrolladores están acostumbrados a tener `jQuery` a mano.

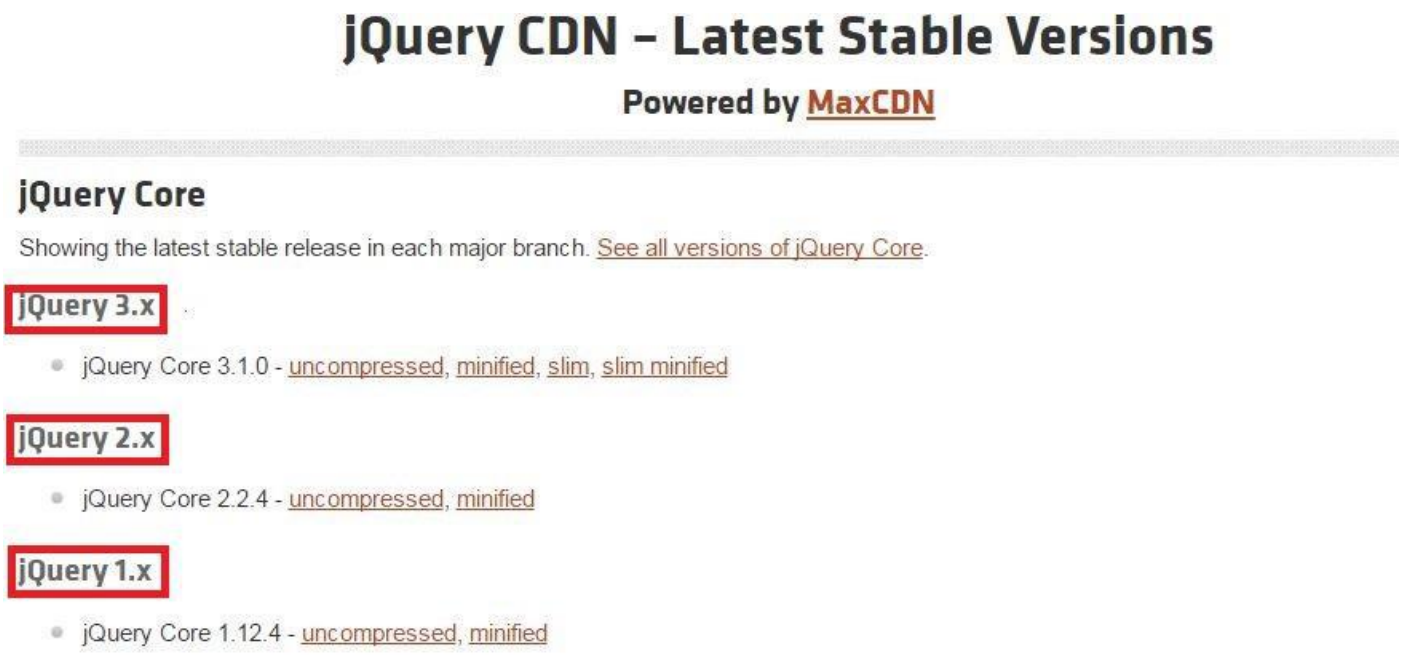
En tales situaciones se puede utilizar la consola de `Chrome Developer Tools` (F12) para añadir manualmente `jQuery` en una página cargada ejecutando lo siguiente:

```
var j = document.createElement('script');
j.onload = function(){ jQuery.noConflict(); };
j.src = "https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js";
document.getElementsByTagName('head')[0].appendChild(j);
```

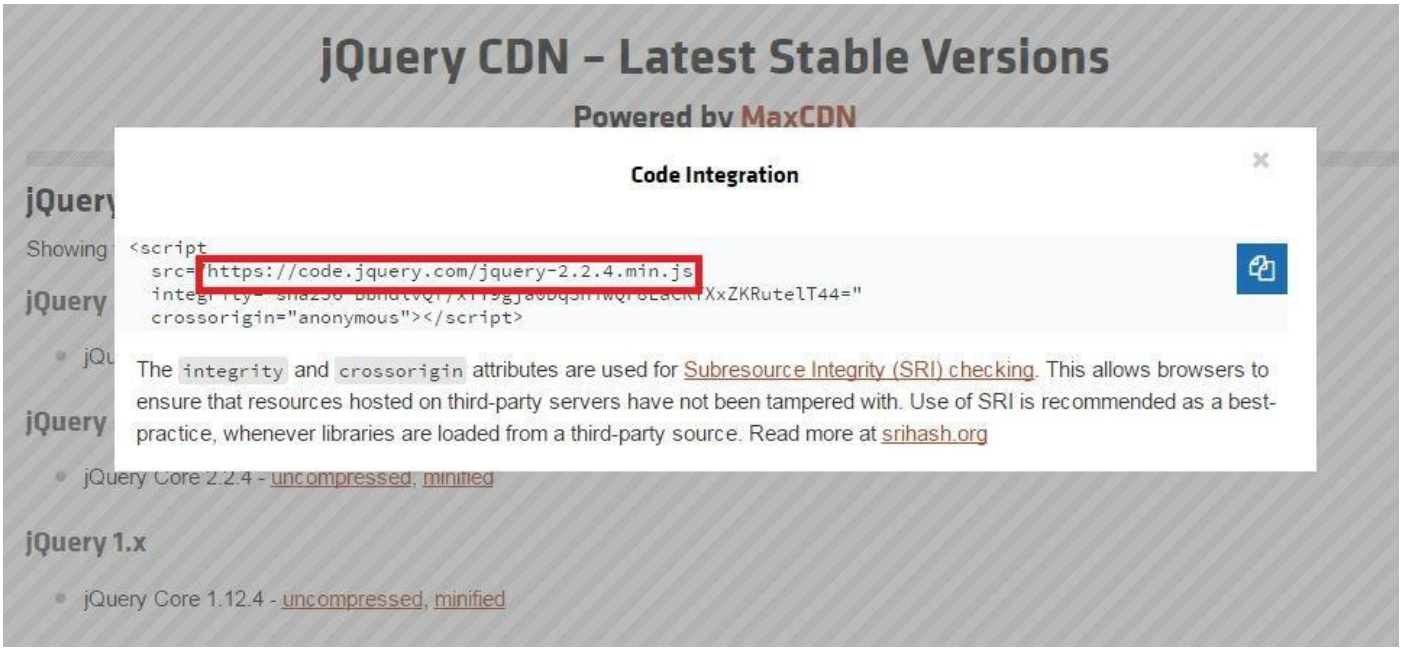
La versión que desea puede diferir de la anterior (`1.12.4`), puede obtener el enlace para la que necesita aquí.

Sección 1.5: Incluir etiqueta script en el encabezado de la página HTML

Para cargar **jQuery** desde la CDN oficial, vaya al sitio web de jQuery. Verá una lista de las diferentes versiones y formatos disponibles.



Ahora, copie la fuente de la versión de jQuery, que desea cargar. Supongamos que desea cargar **jQuery 2.X**, haga clic en la etiqueta **descomprimido** o **minificado** que le mostrará algo como esto:



Copia el código completo (o haz clic en el icono de copiar) y pégalo en el **<head>** o **<body>** de tu html.

La mejor práctica consiste en cargar cualquier biblioteca JavaScript externa en la etiqueta `head` con el atributo `async`. He aquí una demostración:

```
<!DOCTYPE html>
<html>
      <head>
            <title>Cargando jquery-2.2.4</title>
            <script src="https://code.jquery.com/jquery-2.2.4.min.js" async></script>
      </head>
      <body>
            <p> Esta página se ha cargado con jquery.</p>
      </body>
</html>
```

Si utiliza el atributo `async`, tenga en cuenta que las bibliotecas javascript se cargan de forma asíncrona y se ejecutan en cuanto están disponibles. Si se incluyen dos bibliotecas y la segunda depende de la primera, si la segunda se carga y ejecuta antes que la primera, puede producirse un error y la aplicación puede romperse.

Sección 1.6: El objeto jQuery

Cada vez que se llama a jQuery, usando `$()` o `jQuery()`, internamente se está creando una **`new`** instancia de `jQuery`. Este es el código fuente que muestra la nueva instancia:

```
// Definir una copia local de jQuery
jQuery = function( selector, context ) {
      // El objeto jQuery es en realidad sólo el constructor init 'enhanced'
      // Necesita init si se llama a jQuery (sólo permitir que se lance un error si no se incluye)
      return new jQuery.fn.init( selector, context );
}
```

Internamente jQuery se refiere a su prototipo como `.fn`, y el estilo utilizado aquí de instanciar internamente un objeto jQuery permite que ese prototipo sea expuesto sin el uso explícito de **`new`** por el llamador.

Además de crear una instancia (que es como se expone la API de jQuery, como `.each`, `.children`, `.filter`, etc.), internamente jQuery también creará una estructura tipo array para que coincida con el resultado del selector (siempre que se haya pasado como argumento algo que no sea nada, `undefined`, **`null`**, o similar). En el caso de un único elemento, esta estructura en forma de array contendrá sólo ese elemento.

Una demostración simple sería encontrar un elemento con un id, y luego acceder al objeto jQuery para devolver el elemento DOM subyacente (esto también funcionará cuando múltiples elementos coincidan o estén presentes).

```
var $div = $("#myDiv"); // rellenar el objeto jQuery con el resultado del selector id
var div = $div[0]; // acceder a la estructura tipo array del objeto jQuery para obtener el
elemento DOM
```

Capítulo 2: Selectores

Un selector jQuery selecciona o encuentra un elemento DOM (modelo de objetos del documento) en un documento HTML. Se utiliza para seleccionar elementos HTML basándose en id, nombre, tipos, atributos, clase, etc. Se basa en los selectores CSS existentes.

Sección 2.1: Visión general

Los elementos pueden ser seleccionados por jQuery usando Selectores jQuery. La función devuelve un elemento o una lista de elementos.

Selectores básicos

```
$("*") // Todos los elementos
$("div") // Todos los elementos <div>
$(".blue") // Todos los elementos con class=blue
$(".blue.red") // Todos los elementos con class=blue Y class=red
$(".blue,.red") // Todos los elementos con class=blue O class=red
$("#headline") // El (primer) elemento con id=headline
$("[href]") // Todos los elementos con un atributo href
$("[href='example.com']") // Todos los elementos con href=ejemplo.com
```

Operadores relacionales

```
$("div span") // Todos los <span>s descendientes de un <div>
$("div > span") // Todos los <span>s hijos directos de un <div>
$("a ~ span") // Todos los <span>s que son hermanos a continuación de un <a>
$("a + span") // Todos los <span> que se encuentran inmediatamente después de un <a>
```

Sección 2.2: Tipos de selectores

En jQuery puede seleccionar elementos en una página utilizando varias propiedades del elemento, incluyendo:

- Tipo
- Clase
- ID
- Posesión del atributo
- Valor del atributo
- Selector indexado
- Pseudoestado

Si conoces los selectores CSS te darás cuenta de que los selectores en jQuery son los mismos (con pequeñas excepciones).

Tomemos como ejemplo el siguiente HTML:

```
<a href="index.html"></a> <!-- 1 -->
<a id="second-link"></a> <!-- 2 -->
<a class="example"></a> <!-- 3 -->
<a class="example" href="about.html"></a> <!-- 4 -->
<span class="example"></span> <!-- 5 -->
```

Seleccionar por tipo:

El siguiente selector jQuery seleccionará todos los elementos `<a>`, incluidos 1, 2, 3 y 4.

```
$("a")
```

Seleccionar por clase

El siguiente selector jQuery seleccionará todos los elementos de la clase `ejemplo` (incluidos los elementos que no son a), que son 3, 4 y 5.

```
$(".ejemplo")
```

Seleccionar por ID

El siguiente selector jQuery seleccionará el elemento con el ID dado, que es 2.

```
$("#segundo-enlace")
```

Seleccionar por posesión de un atributo

El siguiente selector jQuery seleccionará todos los elementos con un atributo `href` definido, incluidos 1 y 4.

```
$("[href]")
```

Seleccionar por valor de atributo

El siguiente selector jQuery seleccionará todos los elementos donde exista el atributo `href` con valor `index.html`, que es justo 1.

```
$("[href='index.html']")
```

Selección por posición indexada (*Selector indexado*)

El siguiente selector jQuery seleccionará sólo 1, el segundo **<a>** es decir, el `segundo-enlace` porque el índice suministrado es `1` como `eq(1)` (¡Nótese que el índice comienza en `0` por lo tanto el segundo se seleccionó aquí!).

```
$("a:eq(1)")
```

Seleccionar con exclusión indexada

Para excluir un elemento utilizando su índice `:not(:eq())`.

Lo siguiente selecciona elementos **<a>**, excepto que, con el `ejemplo` de clase, que es 1.

```
$("a").not(":eq(0)")
```

Seleccionar con exclusión

Para excluir un elemento de una selección, utiliza `:not()`.

A continuación, se seleccionan los elementos **<a>**, excepto los que tienen la clase `ejemplo`, que son 1 y 2.

```
$("a:not(.ejemplo)")
```

Selección por pseudoestado

También puede seleccionar en jQuery utilizando pseudoestados, incluyendo `:first-child`, `:last-child`, `:first-of-type`, `:last-of-type`, etc.

El siguiente selector jQuery sólo seleccionará el primer elemento **<a>**: número 1.

```
$("a:first-of-type")
```

Combinación de selectores jQuery

También puede aumentar su especificidad combinando varios selectores jQuery; puede combinar cualquier número de ellos o combinarlos todos. También puede seleccionar varias clases, atributos y estados al mismo tiempo.

```
$("a.class1.class2.class3#algunID[attr1][attr2='algo'][attr3='algo']:first-of-type:firstchild")
```

Esto seleccionaría un elemento `<a>` que:

- Tiene las siguientes clases: `clase1`, `clase2` y `clase3`.
- Tiene el siguiente ID: `algunID`.
- Tiene el siguiente atributo: `attr1`.
- Tiene los siguientes atributos y valores: `attr2` con valor `algo`, `attr3` con valor `algo`.
- Tiene los siguientes estados: `first-child` y `first-of-type`.

También puede separar los distintos selectores con una coma:

```
$("a, clase1, #algunID")
```

Esto seleccionaría:

- Todos los elementos `<a>`.
- Todos los elementos que tienen la clase `clase1`.
- Un elemento con el id `#algunID`.

Selección de hijos y hermanos

Los selectores de jQuery suelen ajustarse a las mismas convenciones que CSS, lo que permite seleccionar hijos y hermanos de la misma manera.

- Para seleccionar un hijo no directo, utiliza un espacio.
- Para seleccionar un hijo directo, utiliza un `>`.
- Para seleccionar un hermano adyacente después del primero, utiliza `+`.
- Para seleccionar un hermano no adyacente después del primero, utiliza una `~`.

Selección de comodines

Puede darse el caso de que queramos seleccionar todos los elementos, pero no exista una propiedad común sobre la que seleccionar (clase, atributo, etc.). En ese caso podemos utilizar el selector `*` que simplemente selecciona todos los elementos:

```
$('#wrapper *') // Seleccionar todos los elementos dentro del elemento #wrapper
```

Sección 2.3: Selectores de caché

Cada vez que utilizas un selector en jQuery se busca en el DOM los elementos que coinciden con tu consulta. Hacer esto con demasiada frecuencia o repetidamente disminuirá el rendimiento. Si hace referencia a un selector específico más de una vez, debe añadirlo a la caché asignándolo a una variable:

```
var nav = $('#navegacion');
nav.show();
```

Esto reemplazaría:

```
$('#navegacion').show();
```

El almacenamiento en caché de este selector puede resultar útil si su sitio web necesita mostrar/ocultar este elemento con frecuencia. Si hay varios elementos con el mismo selector, la variable se convertirá en un array de estos elementos:

```
<div class="padre">
     <div class="hijo">Hijo 1</div>
     <div class="hijo">Hijo 2</div>
</div>
<script>
     var hijos = $('.hijo');
     var primerTextoHijo = hijos[0].text();
     console.log(primerTextoHijo);
     // salida: "Hijo 1"
</script>
```

NOTA: El elemento tiene que existir en el DOM en el momento de su asignación a una variable. Si no hay ningún elemento en el DOM con una clase llamada `child` estarás almacenando un array vacío en esa variable.

```
<div class="padre"></div>
<script>
      var padre = $('.padre');
      var hijos = $('.hijo');
      console.log(hijos);
      // salida: []
      padre.append('<div class="hijo">Hijo 1</div>');
      hijos = $('.hijo');
      console.log(hijos[0].text());
      // salida: "Hijo 1"
</script>
```

Recuerda reasignar el selector a la variable después de añadir/eliminar elementos en el DOM con ese selector.

Nota: Cuando se almacenan selectores en caché, muchos desarrolladores comienzan el nombre de la variable con `$` para indicar que se trata de un objeto jQuery, de la siguiente manera:

```
var $nav = $('#navegacion');
$nav.show();
```

Sección 2.4: Combinar selectores

Considera la siguiente estructura DOM.

```
<ul class="padreUl">
      <li> Level 1
            <ul class="hijoUl">
                  <li>Level 1-1 <span> Item - 1 </span></li>
                  <li>Level 1-1 <span> Item - 2 </span></li>
            </ul>
      </li>
      <li> Level 2
            <ul class="hijoUl">
                  <li>Level 2-1 <span> Item - 1 </span></li>
                  <li>Level 2-1 <span> Item - 1 </span></li>
            </ul>
      </li>
</ul>
```

Selectores descendientes e hijos

Dado un padre `<ul>` - `padreUl` buscará sus descendientes (`<li>`),

1. Simple `$('padre hijo')`

>> `$('ul.padreUl li')`

Obtiene todos los descendientes coincidentes del antepasado especificado, *todos los niveles hacia abajo*.

2. `>` - `$('padre > hijo')`

>> `$('ul.padreUl > li')`

Esto encuentra todos los hijos coincidentes (*sólo el 1er nivel hacia abajo*).

3. Selector basado en el contexto - `$('hijo','padre')`

>> `$('li','ul.padreUl')`

Funciona igual que el punto 1.

4. `find()` - `$('padre').find('hijo')`

>> `$('ul.padreUl').find('hijo')`

Funciona igual que el punto 1.

5. `children()` - `$('padre').find('hijo')`

>> `$('ul.padreUl').children('li')`

Funciona igual que el punto 2.

Otras combinaciones

Selector de grupo : ","

Selecciona todos los elementos **<ul>** Y todos los elementos **<li>** Y todos los elementos ****:

```
$('ul, li, span')
```

Selector de múltiplos : "" (sin carácter)

Selecciona todos los elementos **<ul>** con clase `padreUl`:

```
$('ul.padreUl')
```

Selector de hermanos adyacentes : "+"

Seleccionar todos los elementos **<li>** que se colocan inmediatamente después de otro elemento **<li>**:

```
$('li + li')
```

Selector general de hermanos : "~"

Seleccionar todos los elementos **<li>** que sean hermanos de otros elementos **<li>**.

```
$('li ~ li')
```

Sección 2.5: Elementos DOM como selectores

jQuery acepta una amplia variedad de parámetros, y uno de ellos es un elemento DOM real. Pasar un elemento DOM a jQuery hará que la estructura subyacente del objeto jQuery contenga ese elemento.

jQuery detectará que el argumento es un elemento DOM inspeccionando su nodeType.

El uso más común de un elemento DOM es en las callbacks, donde el elemento actual se pasa al constructor de jQuery para acceder a la API de jQuery.

Como en el callback `each` (nota: `each` es una función iteradora).

```
$(".elementos").each(function() {
     // el elemento actual está vinculado a `this` internamente por jQuery cuando se utiliza cada
     var elementoActual = this;
     // en este punto, elementoActual (o this) tiene acceso a la API Nativa
     // construir un objeto jQuery con el elemento elementoActual(this)
     var $elementoActual = $(this);
     // ahora $elementoActual tiene acceso a la API jQuery
});
```

Sección 2.6: Cadenas de caracteres HTML como selectores

jQuery acepta una amplia variedad de parámetros como "selectores", y uno de ellos es una cadena de caracteres HTML. Al pasar una cadena de caracteres HTML a jQuery, la estructura subyacente de tipo array del objeto jQuery contendrá el HTML construido resultante.

jQuery utiliza regex para determinar si la cadena que se pasa al constructor es una cadena HTML, y también que debe empezar por `<`. Ese regex se define como `rquickExpr = /^(?:\s*(<[\w\W]+>)[^>]*|#([\w-]*))$/` (explicación en regex101.com).

El uso más común de una cadena HTML como selector es cuando se necesita crear conjuntos de elementos DOM sólo en código, a menudo esto es utilizado por las bibliotecas para cosas como ventanas emergentes modales.

Por ejemplo, una función que devolviera una etiqueta de anclaje envuelta en un div como plantilla.

```
function template(href,text){
     return $("<div><a href='" + href + "'>" + text + "</a></div>");
}
```

Devolvería un objeto jQuery que contiene.

```
<div>
     <a href="google.com">Google</a>
</div>
```

si se llama como `template("google.com","Google")`.

Capítulo 3: Función each

Sección 3.1: Función each de jQuery

HTML:

```
<ul>
<li>Mango</li>
<li>Libro</li>
</ul>
```

Script:

```
$("li").each(function(index) {
     console.log(index + ": " + $(this).text());
});
```

De este modo, se registra un mensaje por cada elemento de la lista:

0: Mango
1: Libro

Capítulo 4: Atributos

Sección 4.1: Diferencias entre attr() y prop()

`attr()` obtiene/establece el atributo HTML utilizando las funciones DOM `getAttribute()` y `setAttribute()`.`prop()` funciona estableciendo la propiedad DOM sin cambiar el atributo. En muchos casos los dos son intercambiables, pero a veces uno es más necesario que el otro.

Para marcar una casilla de verificación:

```
$('#tosAceptar').prop('checked', true); // el uso de attr() no funcionará correctamente aquí
```

Para eliminar una propiedad se puede utilizar el método `removeProp()`. Del mismo modo `removeAttr()` elimina atributos.

Sección 4.2: Obtener el valor de atributo de un elemento HTML

Cuando se pasa un único parámetro a la función `.attr()`, ésta devuelve el valor del atributo pasado en el elemento seleccionado.

Sintaxis:

```
$([selector]).attr([nombre atributo]);
```

Ejemplo:

HTML:

```
<a href="/inicio">Inicio</a>
```

jQuery:

```
$('a').attr('href');
```

Obtención de atributos `data`:

jQuery ofrece la función `.data()` para tratar los atributos de datos. La función `.data` devuelve el valor del atributo data del elemento seleccionado.

Sintaxis:

```
$([selector]).data([nombre atributo]);
```

Ejemplo:

HTML:

```
<article data-column="3"></article>
```

jQuery:

```
$("article").data("columna")
```

Nota:

> El método `data()` de jQuery le dará acceso a los atributos `data-*`, PERO, no tiene en cuenta el caso del atributo `name`. Referencia

Sección 4.3: Establecer el valor de un atributo HTML

Si quieres añadir un atributo a algún elemento puedes utilizar la función `attr(nombreAtributo, valorAtributo)`. Por ejemplo:

```
$('a').attr('titulo', 'Haz clic en mi');
```

Este ejemplo añadirá el texto `"Haz clic en mi"` a todos los enlaces de la página.

La misma función se utiliza para cambiar los valores de los atributos.

Sección 4.4: Eliminar atributo

Para eliminar un atributo de un elemento puede utilizar la función `.removeAttr(nombreAtributo)`. Por ejemplo:

```
$('#inicio').removeAttr('titulo');
```

Esto eliminará el atributo `titulo` del elemento con ID `inicio`.

Capítulo 5: Evento document-ready

Sección 5.1: ¿Qué es el document-ready y cómo debo utilizarlo?

El código jQuery a menudo se envuelve en `jQuery(function($) { ... });` para que sólo se ejecute después de que el DOM haya terminado de la carga.

```
<script type="text/javascript">
jQuery(function($) {
     // el texto del div será "Hola".
     $("#miDiv").text("Hola");
});
</script>
<div id="miDiv">Texto</div>
```

Esto es importante porque jQuery (y JavaScript en general) no puede seleccionar un elemento DOM que no ha sido renderizado en la página.

```
<script type="text/javascript">
// ningún elemento con id="miDiv" existe en este momento, por lo que $("#miDiv") es una
// selección vacía, y esto no tendrá ningún efecto
$("#miDiv").text("Hola");
</script>
<div id="miDiv">Texto</div>
```

Tenga en cuenta que puede utilizar un alias del espacio de nombres jQuery pasando un controlador personalizado al método `.ready()`. Esto es útil para casos en los que otra librería JS está usando el mismo alias `$` acortado que *jQuery*, lo que crea un conflicto. Para evitar este conflicto, debe llamar a `$.noConflict();` - Esto le obliga a utilizar sólo el espacio de nombres por defecto de *jQuery* (En lugar del alias corto `$`).

Pasando un manejador personalizado al manejador `.ready()`, podrás elegir el nombre del alias que usará *jQuery*.

```
$.noConflict();
jQuery( document ).ready(function( $ ) {
     // Aquí podemos usar '$' como alias de jQuery sin que entre en conflicto con otras
     // bibliotecas que utilicen el mismo espacio de nombres
     $('body').append('<div>Hola</div>')
});
jQuery( document ).ready(function( jq ) {
     // Aquí usamos un alias jQuery personalizado 'jq
     jq('body').append('<div>Hola</div>')
});
```

En lugar de simplemente colocar el código jQuery en la parte inferior de la página, el uso de la función `$(document).ready` garantiza que todos los elementos HTML se han renderizado y que todo el Modelo de Objetos del Documento (DOM) está listo para que se ejecute el código JavaScript. para que se ejecute el código JavaScript.

Sección 5.2: jQuery 2.2.3 y versiones anteriores

Todos estos son equivalentes, el código dentro de los bloques se ejecutará cuando el documento esté listo:

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

Debido a que estos son equivalentes el primero es la forma recomendada, la siguiente es una versión de que con la palabra clave `jQuery` en lugar de la `$` que producen los mismos resultados:

```
jQuery(function() {
      // código
});
```

Sección 5.3: jQuery 3.0

Notación

A partir de jQuery 3.0, sólo se recomienda este formulario:

```
jQuery(function($) {
// Ejecutar cuando el documento esté listo
// $ (primer argumento) será una referencia interna a jQuery
// Nunca confíes en que $ sea una referencia a jQuery en el espacio de nombres global});
```

Todos los demás gestores de document-ready están obsoletos en jQuery 3.0.

Asíncrono

A partir de jQuery 3.0, el manejador ready siempre será llamado de forma asíncrona. Esto significa que, en el código siguiente, el log 'outside handler' siempre se mostrará primero, independientemente de si el documento estaba listo en el punto de ejecución.

```
$(function() {
      console.log("dentro del manejador");
});
console.log("fuera del manejador");
```

> dentro del manejador

> fuera del manejador

Sección 5.4: Adjuntar eventos y manipular el DOM dentro de ready()

Ejemplos de uso de `$(document).ready()`:

1. **Adjuntar controladores de eventos**

Adjuntar manejadores de eventos jQuery

```
$(document).ready(function() {
      $("button").click(function() {
            // Código de la función click
      });
});
```

2. **Ejecutar el código jQuery después de crear la estructura de la página**

```
jQuery(function($) {
      // establecer el valor de un elemento.
      $("#miElemento").val("Hola");
});
```

3. **Manipular la estructura DOM cargada**

Por ejemplo: ocultar un `div` cuando la página se carga por primera vez y mostrarlo al hacer clic en un botón.

```
$(document).ready(function() {
      $("#toggleDiv").hide();
      $("button").click(function() {
            $("#toggleDiv").show();
      });
});
```

Sección 5.5: Diferencia entre $(document).ready() y $(window).load()

`$(window).load()` quedó **obsoleto en la versión 1.8 de jQuery (y se eliminó por completo en la versión 3.0 de jQuery)**, por lo que ya no debería utilizarse. En la página de jQuery sobre este evento se explican los motivos de su eliminación.

> Advertencias del evento load cuando se utiliza con imágenes
>
> Un reto común que los desarrolladores intentan resolver usando el atajo. load() es ejecutar una función
>
> cuando una imagen (o colección de imágenes) se ha cargado completamente. Hay varias advertencias conocidas con
>
> que deben tenerse en cuenta. Estas son:
>
> - No funciona de forma consistente ni fiable entre navegadores.
> - No funciona correctamente en WebKit si el src de la imagen es el mismo que antes.
> - No burbujea correctamente en el árbol DOM.
> - Puede dejar de disparar para las imágenes que ya viven en la caché del navegador

Si aún desea utilizar `load()` se documenta a continuación:

`$(document).ready()` espera hasta que el DOM completo está disponible -- todos los elementos en el HTML han sido analizados y están en el documento. Sin embargo, es posible que recursos como las imágenes no se hayan cargado completamente en este momento. Si es importante esperar hasta que todos los recursos estén cargados, `$(window).load()` **y eres consciente de las limitaciones significativas de este evento**, entonces se puede utilizar lo siguiente en su lugar:

```
$(document).ready(function() {
      console.log($("#mi_imagen_largo").height()); // puede ser 0 porque la imagen no está
      disponible
});
$(window).load(function() {
      console.log($("#my_imagen_largo").height()); // será correcto
});
```

Sección 5.6: Diferencia entre jQuery(fn) y la ejecución del código antes de </body>

El uso del evento document-ready puede tener pequeños inconvenientes de rendimiento, con un retraso en la ejecución de hasta ~300ms. A veces se puede conseguir el mismo comportamiento ejecutando código justo antes de la etiqueta de cierre **</body>**:

```
<body>
      !<span id="saludo"></span> mundo!
      <script>
            $("#saludo").text("Hola");
      </script>
</body>
```

producirá un comportamiento similar, pero actuará antes que como no espera a que se dispare el evento document ready como hace en:

```
<head>
      <script>
      jQuery(function($) {
            $("#saludo").text("Hola");
      });
      </script>
</head>
<body>
      <span id="saludo"></span> mundo!
</body>
```

El énfasis en el hecho de que el primer ejemplo se basa en el conocimiento de su página y la colocación de la secuencia de comandos justo antes de la etiqueta de cierre **</body>** y específicamente después de la etiqueta span.

Capítulo 6: Eventos

Sección 6.1: Eventos delegados

Empecemos con un ejemplo. Aquí está un ejemplo muy simple HTML.

Ejemplo de HTML

```
<html>
      <head>
      </head>
      <body>
            <ul>
                  <li>
                        <a href="algun_url/">Link 1</a>
                  </li>
                  <li>
                        <a href="algún_url/">Link 2</a>
                  </li>
                  <li>
                        <a href="algun_url/">Link 3</a>
                  </li>
            </ul>
            </body>
</html>
```

El problema

Ahora en este ejemplo, queremos añadir un oyente de eventos a todos los elementos `<a>`. El problema es que la lista de este ejemplo es dinámica. Los elementos `<li>` se añaden y eliminan a medida que pasa el tiempo. Sin embargo, la página no se actualiza entre los cambios, lo que nos permitiría utilizar simples escuchadores de eventos de clic a los objetos de enlace (es decir, `$('a').click()`).

El problema que tenemos es cómo añadir eventos a los elementos `<a>` que van y vienen.

Información general - Propagación de eventos

Los eventos delegados sólo son posibles gracias a la propagación de eventos (a menudo denominada burbujeo de eventos). Cada vez que se dispare un evento, se expandirá hacia arriba (hasta la raíz del documento). *Delegan* el manejo de un evento en un elemento antepasado que no cambia, de ahí el nombre de eventos "delegados".

Así que, en el ejemplo anterior, al hacer clic en `<a>` enlace de elemento se activará 'clic' evento en estos elementos en este orden:

- a
- li
- ul
- body
- html
- raíz del documento

Solución

Sabiendo lo que hace el burbujeo de eventos, podemos capturar uno de los eventos deseados que se propagan a través de nuestro código HTML.

Un buen lugar para cogerlo en este ejemplo es el elemento `<ul>`, ya que ese elemento no es dinámico:

```
$('ul').on('click', 'a', function () {
     console.log(this.href); jQuery vincula la función de evento al elemento DOM apuntado
     // de esta forma `this` se refiere al ancla y no a la lista
     // Lo que quieras hacer cuando se haga clic en el enlace
});
```

En la parte superior:

- Tenemos 'ul' que es el destinatario de este oyente de eventos.
- El primer parámetro ('click') define qué eventos intentamos detectar.
- El segundo parámetro ('a') se utiliza para declarar de dónde debe *originarse* el evento (de todos los elementos hijos bajo el receptor de este evento, ul).
- Por último, el tercer parámetro es el código que se ejecuta si se cumplen los requisitos de los parámetros primero y segundo.

En detalle cómo funciona la solución

1. El usuario hace clic en el elemento `<a>`.
2. Eso dispara el evento click en el elemento `<a>`.
3. El evento empieza a burbujear hacia la raíz del documento.
4. El evento burbujea primero en el elemento `<li>` y luego en el elemento `<ul>`.
5. El listener de eventos se ejecuta ya que el elemento `<ul>` tiene el listener de eventos adjunto.
6. El escuchador de eventos detecta primero el evento desencadenante. El evento burbujeante es 'click' y el listener tiene 'click', es un pase.
7. El oyente comprueba si el segundo parámetro ('a') coincide con cada elemento de la cadena de burbujas. Como el último de la cadena es una "a", coincide con el filtro y también es un aprobado.
8. El código del tercer parámetro se ejecuta utilizando el elemento coincidente como si fuera `this`. Si la función no incluye una llamada a `stopPropagation()`, el evento continuará propagándose hacia arriba, hacia la raíz (`document`).

Nota: Si no se dispone de un antepasado adecuado que no cambie, se debe utilizar `document`. Como hábito no utilice `'body'` por las siguientes razones:

`body` tiene un bug, relacionado con el estilo, que puede hacer que los eventos del ratón no le lleguen. Esto depende del navegador y puede ocurrir cuando la altura del cuerpo calculada es 0 (por ejemplo, cuando todos los elementos hijos tienen posiciones absolutas absolutas). Los eventos de ratón siempre burbujean al `document`.

`document` *siempre* existe para tu script, por lo que puedes adjuntar manejadores delegados al `document` fuera de un *manejador DOM-ready* y estar seguro de que seguirán funcionando.

Sección 6.2: Manejadores de eventos Attach y Detach

Adjuntar un manejador de eventos

Desde la versión 1.7 jQuery dispone de la API de eventos `.on()`. De esta manera cualquier evento javascript estándar o personalizado puede ser en el elemento jQuery seleccionado. Hay atajos como `.click()`, pero `.on()` te da más opciones.

HTML

```
<button id="foo">bar</button>
```

jQuery

```
$("#foo").on("click", function() {
     console.log( $(this).text()); //bar
});
```

Separar un manejador de eventos

Naturalmente, también tienes la posibilidad de separar eventos de tus objetos jQuery. Para ello, utiliza `.off(eventos[, selector][, manejador])`.

HTML

```
<button id="hello">hello</button>
```

jQuery

```
$('#hola').on('click', function(){
     console.log('¡hola mundo!');
     $(this).off();
});
```

Al pulsar el botón `$(this)` se referirá al objeto jQuery actual y eliminará todos los manejadores de eventos adjuntos del mismo. También puede especificar qué controlador de eventos debe eliminarse.

jQuery

```
$('#hola').on('click', function(){
     console.log('¡hola mundo!');
     $(this).off('click');
});
$('#hola').on('mouseenter', function(){
     console.log('está a punto de hacer clic');
});
```

En este caso, el evento `mouseenter` seguirá funcionando después de hacer clic.

Sección 6.3: Activación y desactivación de eventos específicos mediante jQuery. (Named Listeners)

A veces se desea desconectar a todos los oyentes registrados previamente.

```
// Añadir un manejador de clic normal
$(document).on("click",function(){
     console.log("Documento clicado 1")
});
// Añadir otro manejador de clics
$(document).on("click",function(){
     console.log("Documento clicado 2")
});
// Eliminar todos los manejadores registrados.
$(document).off("click")
```

Un problema con este método es que TODAS las escuchas vinculadas al documento por otros plugins, etc., también se eliminarían.

La mayoría de las veces, queremos desprendernos de todos los oyentes atados sólo por nosotros.

Para conseguirlo, podemos enlazar escuchas con nombre como,

```
// Añadir receptor de eventos con nombre.
$(document).on("click.mimodulo",function(){
     console.log("Documento clicado 1")
});
$(document).on("click.mymodule",function(){
     console.log("Documento clicado 2")
});
// Elimina el receptor de eventos nombrado.
$(document).off("click.mimodulo");
```

Esto garantiza que no se modifique inadvertidamente ningún otro receptor de clics.

Sección 6.4: originalEvent

A veces habrá propiedades que no estén disponibles en el evento jQuery. Para acceder a las propiedades subyacentes utilice `Event.originalEvent`.

Obtener la rueda del ratón

```
$(document).on("wheel",function(e){
      console.log(e.originalEvent.deltaY)
      // Devuelve un valor entre -100 y 100 dependiendo de la dirección en la que se desplace
})
```

Sección 6.5: Eventos para repetir elementos sin usar ID's

Problema

Hay una serie de elementos que se repiten en la página que usted necesita saber cuál es un evento que se produjo en hacer algo con esa instancia específica. algo con esa instancia específica.

Solución

- Dar a todos los elementos comunes una clase común.
- Aplica un receptor de eventos a una clase. `this` controlador de eventos interno es el elemento selector correspondiente en el que se produjo el evento.
- Recorrer hasta el contenedor más externo que se repite para esa instancia comenzando en `this`.
- Utiliza `find()` dentro de ese contenedor para aislar otros elementos específicos de esa instancia.

HTML

```
<div class="item-wrapper" data-item_id="346">
      <div class="item"><span class="persona">Fred</span></div>
      <div class="item-toolbar">
            <button class="borrar">Borrar</button>
      </div>
</div>
<div class="item-wrapper" data-item_id="393">
      <div clss="item"><span class="persona">Wilma</span></div>
      <div class="item-toolbar">
            <button class="borrar">Borrar</button>
      </div>
</div>
```

jQuery

```
$(function() {
      $('.borrar').on('click', function() {
            // "this" es elemento evento ocurrido en
            var $btn = $(this);
            // pasar al contenedor envolvente
            var $itemWrap = $btn.closest('.item-wrapper');
            // buscar dentro de la envoltura para obtener la persona para esta instancia de botón
            var person = $itemWrap.find('.person').text();
            // enviar delete al servidor y eliminar de la página en caso de éxito de ajax
            $.post('url/string', { id: $itemWrap.data('item_id')}).done(function(response) {
                  $itemWrap.remove()
            }).fail(function() {
                  alert('Ooops, no borrado en el servidor');
            });
      });
});
```

Sección 6.6: Evento de carga del documento .load()

Si quieres que tu script espere a que se cargue un determinado recurso, como una imagen o un PDF puedes usar `.load()`, que es un atajo de teclado para `.on("load", handler)`.

HTML

```
<img src="imagen.jpeg" alt="imagen" id="imagen">
```

jQuery

```
$("#imagen").load(function() {
      // ejecutar script
});
```

Capítulo 7: Manipulación del DOM

Sección 7.1: Creación de elementos del DOM

La función `jQuery` (normalmente denominada `$`) puede utilizarse tanto para seleccionar elementos como para crear elementos nuevos.

```
var miLink = $('<a href="http://stackexchange.com"></a>');
```

Opcionalmente puede pasar un segundo argumento con atributos de elementos:

```
var miLink = $('<a>', {'href': 'http://stackexchange.com'});
```

`'<a>'` --> El primer argumento especifica el tipo de elemento DOM que se desea crear. En este ejemplo es un ancla, pero podría ser cualquier cosa de esta lista. Véase la especificación para una referencia del elemento `a`.

`{ 'href': 'http://stackexchange.com' }` --> el segundo argumento es un objeto JavaScript que contiene pares de atributos pares nombre/valor.

Los pares <nombre>: <valor> aparecerán entre el `< >` del primer argumento, por ejemplo `<a nombre:valor>` que para nuestro ejemplo sería `<a href="http://stackexchange.com"></a>`.

Sección 7.2: Manipular clases de elementos

Suponiendo que la página incluye un elemento HTML como:

```
<p class="párrafo-pequeno">
Este es un pequeño <a href="https://es.wikipedia.org/wiki/P%C3%A1rrafo">párrafo</a>
con un elemento a <a class="trusted" href="http://stackexchange.com">link</a> dentro.
</p>
```

jQuery proporciona funciones útiles para manipular las clases DOM, sobre todo `hasClass()`, `addClass()`, `removeClass()` y `toggleClass()`. Estas funciones modifican directamente el atributo class de los elementos coincidentes.

```
$('p').hasClass('parrafo-pequeno'); // true
$('p').hasClass('parrafo-largo'); // false
// Añadir una clase a todos los enlaces dentro de los párrafos
$('p a').addClass(' enlace-en-parrafo-no-confiable');
// Eliminar la clase de a.trusted
$('a.confiable.enlace-en-parrafo-no-confiable')
.removeClass('enlace-en-parrafo-no-confiable')
.addClass('enlace-en-parrafo-confiable');
```

Alternar una clase

Dado el ejemplo de marcado, podemos añadir una clase con nuestro primer `.toggleClass()`:

```
$(".parrafo-pequeno").toggleClass("bonito");
```

Esto devolvería **`true`**: `$(".parrafo-pequeno").hasClass("bonito")`

`toggleClass` proporciona el mismo efecto con menos código que:

```
if($(".parrafo-pequeno").hasClass("bonito")){
     $(".parrafo-pequeno").removeClass("bonito");
}
else {
     $(".parrafo-pequeno").addClass("bonito");
}
```

alternar Dos clases:

```
$(".parrafo-pequeno").toggleClass("bonito genial");
```

Booleano para añadir/eliminar clases:

```
$(".parrafo-pequeno").toggleClass("bonito",true); // no puede ser verdad/falsedad
$(".parrafo-pequeno").toggleClass("bonito",false);
```

Función para cambiar de clase (ver ejemplo más abajo para evitar un problema)

```
$("div.superficie").toggleClass(function() {
     if ($(this).parent().is(".agua")) {
          return "mojado";
     } else {
          return "seco";
     }
});
```

Usado en ejemplos:

```
// funciones para utilizar en los ejemplos
function contieneCadenaTexto(miCadenaTexto, miSubcadenaTexto) {
     return miCadenaTexto.indexOf(miSubcadenaTexto) !== -1;
}
function isImpar(num) {return num % 2;}
var mostrarClase = true; // queremos añadir la clase
```

Ejemplos:

Utilice el índice del elemento para alternar entre clases pares e impares.

```
$("div.gridrow").toggleClass(function(index, viejasClases, false), mostrarClase ) {
     mostrarClase
     if (isImpar(index)) {
          return "mojado";
     } else {
          return "secado";
     }
});
```

Ejemplo de `toggleClass` más complejo, dado un marcado de cuadrícula simple.

```
<div class="grid">
     <div class="gridrow">fila</div>
     <div class="gridrow">fila</div>
     <div class="gridrow">fila</div>
     <div class="gridrow">fila</div>
     <div class="gridrow">fila</div>
     <div class="gridrow gridfooter">fila ¡pero soy un Footer!</div>
</div>
```

Funciones sencillas para nuestros ejemplos:

```
function isImpar(num) {
     return num % 2;
}
function contieneCadenaTexto(miCadenaTexto, miSubcadenaTexto) {
     return miCadenaTexto.indexOf(miSubcadenaTexto) !== -1;
}
var mostrarClase = true; // queremos añadir la clase
```

Añadir una clase par/impar a los elementos con clase `gridrow`.

```
$("div.gridrow").toggleClass(function(index, viejasClases, mostrarEstaClase) {
     if (isImpar(index)) {
          return "impar";
     } else {
          return "par";
     }
     return viejasClases;
}, mostrarClase);
```

Si la fila tiene una clase `gridfooter`, elimine las clases impar/par, mantenga el resto.

```
$("div.gridrow").toggleClass(function(index, viejasClases, mostrarEstaClase) {
      var esFooter = contieneCadenaTexto(viejasClases, "gridfooter");
      if (esFooter) {
            viejasClases = viejasClases.replace('par', ' ').replace('impar', ' ');
            $(this).toggleClass("par impar", false);
      }
      return viejasClases;
}, mostrarClase);
```

Las clases que se devuelven son las que se ven afectadas. Aquí, si un elemento no tiene un `gridfooter`, añada una clase para par/impar. Este ejemplo ilustra la devolución de la lista de clases VIEJAS. Si se elimina este **`else return viejasClases;`** sólo se añaden las nuevas clases, por lo que la fila con una clase `gridfooter` tendría todas las clases eliminadas si no hubiese si no hubiéramos devuelto las antiguas - de lo contrario se habrían activado (eliminado).

```
$("div.gridrow").toggleClass(function(index, viejasClases, mostrarEstaClase) {
      var esFooter = contieneCadenaTexto(viejasClases, "gridfooter");
      if (!esFooter) {
            if (isImpar(index)) {
                  return "imparClaro";
            } else {
                  return "parClaro";
            }
      } else return viejasClases;
}, mostrarClase);
```

Sección 7.3: Otros métodos de la API

jQuery ofrece una variedad de métodos que se pueden utilizar para la manipulación DOM.

El primero es el método `.empty()`.

Imagine el siguiente marcado:

```
<div id="contenido">
      <div>Algun texto</div>
</div>
```

Llamando a `$('#contenido').empty();`, se eliminaría el div interior. Esto también podría conseguirse utilizando `$('#contenido').html('');`.

Otra función muy útil es la función `.closest()`:

```
<tr id="fila_1">
      <td><button type="button" class="borrar">Borrar</button>
</tr>
```

Si quisiera encontrar la fila más cercana a un botón que fue pulsado dentro de una de las celdas de la fila, entonces podría hacer esto:

```
$('.borrar').click(function() {
      $(this).closest('tr');
});
```

Como probablemente habrá varias filas, cada una con sus propios botones de **`borrado`**, usamos `$(this)` dentro de la función `.click()` para limitar el alcance al botón que hemos pulsado.

Si quisieras obtener el id de la fila que contiene el botón **`Borrar`** que pulsaste, podrías hacer algo como esto:

```
$('.borrar').click(function() {
      var $fila = $(this).closest('tr');
      var id = $fila.attr('id');
});
```

Generalmente se considera una buena práctica anteponer a las variables que contienen objetos jQuery un `$` (signo de dólar) para dejar claro de qué que la variable es.

Una alternativa a `.closest()` es el método `.parents()`:

```
$('.borrar').click(function() {
     var $fila = $(this).parents('tr');
     var id = $fila.attr('id');
});
```

y también hay una función `.parent()`:

```
$('.borrar').click(function() {
     var $fila = $(this).parent().parent();
     var id = $fila.attr('id');
});
```

`.parent()` sólo sube un nivel en el árbol DOM por lo que es bastante inflexible, si se cambiara el botón de borrar para que estuviera contenido dentro de un `span` por ejemplo, entonces el selector jQuery se rompería.

Capítulo 8: Recorrido del DOM

Sección 8.1: Seleccionar los hijos de un elemento

Para seleccionar los hijos de un elemento se puede utilizar el método `.children()`.

```
<div class="padre">
     <h2> Un encabezado</h2>
     <p>Lorem ipsum dolor sit amet...</p>
     <p>Praesent quis dolor turpis...</p>
</div>
```

Cambia el color de *todos* los hijos del elemento `.parent`:

```
$('.padre').children().css("color", "green");
```

El método acepta un argumento selector opcional que puede utilizarse para filtrar los elementos devueltos.

```
// Obtener sólo hijos "p"
$('.padre').children("p").css("color", "green");
```

Sección 8.2: Obtener el siguiente elemento

Para obtener el siguiente elemento puedes utilizar el método `.next()`.

```
<ul>
<li>Mark</li>
<li class="anna">Anna</li>
<li>Paul</li>
</ul>
```

Si estás sobre el elemento "Anna" y quieres obtener el siguiente elemento, "Paul", el método `.next()` te permitirá hacerlo. te permitirá hacerlo.

```
// "Paul" ahora tiene texto verde
$(".anna").next().css("color", "green");
```

El método toma un argumento `selector` opcional, que puede utilizarse si el siguiente elemento debe ser un determinado tipo de elemento.

```
// El siguiente elemento es un "li", "Paul" ahora tiene texto verde
$(".anna").next("li").css("color", "green");
```

Si el siguiente elemento no es del tipo `selector` entonces se devuelve un conjunto vacío, y las modificaciones no harán nada.

```
// El elemento siguiente no es un ".mark", no se hará nada en este caso
$(".anna").next(".mark").css("color", "green");
```

Sección 8.3: Obtener el elemento anterior

Para obtener el elemento anterior puedes utilizar el método `.prev()`.

```
<ul>
<li>Mark</li>
<li class="anna">Anna</li>
<li>Paul</li>
</ul>
```

Si estás sobre el elemento "Anna" y quieres obtener el elemento anterior, "Mark", el método `.prev()` te permitirá hacerlo.

```
// "Marca" ahora tiene texto verde
$(".anna").prev().css("color", "green");
```

El método toma un argumento `selector` opcional, que puede utilizarse si el elemento anterior debe ser un determinado tipo de elemento.

```
// El elemento anterior es un "li", "Marca" ahora tiene texto verde
$(".anna").prev("li").css("color", "green");
```

Si el elemento anterior no es del tipo `selector` entonces se devuelve un conjunto vacío, y las modificaciones no harán nada.

```
// El elemento anterior no es un ".paul", no se hará nada en este caso
$(".anna").prev(".paul").css("color", "green");
```

Sección 8.4: Filtrar una selección

Para filtrar una selección puede utilizar el método `.filter()`.

El método se ejecuta en una selección y devuelve una nueva selección. Si el filtro coincide con un elemento, éste se añade a la selección devuelta; de lo contrario, se ignora. Si no se encuentra ningún elemento, se devuelve una selección vacía.

El HTML

Este es el HTML que utilizaremos.

```
<ul>
     <li class="cero">Cero</li>
     <li class="uno">Uno</li>
     <li class="dos">Dos</li>
     <li class="tres">Tres</li>
</ul>
```

Selector

El filtrado mediante selectores es una de las formas más sencillas de filtrar una selección.

```
$("li").filter(":even").css("color", "green"); // Colorear los elementos pares de verde
$("li").filter(".uno").css("font-weight", "bold"); // Poner ".uno" en negrita
```

Función

Filtrar una selección mediante una función es útil si no es posible utilizar selectores.

La función se llama para cada elemento de la selección. Si devuelve un valor `true`, el elemento se añadirá a la selección devuelta.

```
var selection = $("li").filter(function (index, element) {
     // "index" es la posición del elemento
     // "element" es lo mismo que "this"
     return $(this).hasClass("dos");
});
selection.css("color", "green"); // ".dos" se coloreará de verde
```

Elementos

Puede filtrar por elementos DOM. Si los elementos DOM están en la selección, se incluirán en la selección devuelta.

```
var tres = document.getElementsByClassName("tres");
$("li").filter(tres).css("color", "green");
```

Selección

También puede filtrar una selección por otra selección. Si un elemento se encuentra en ambas selecciones, se incluirá en la selección devuelta.

```
var elems = $(".uno, .tres");
$("li").filter(elems).css("color", "green");
```

Sección 8.5: Método find()

`.find()` nos permite buscar entre los descendientes de estos elementos en el árbol DOM y construir un nuevo objeto jQuery a partir de los elementos coincidentes.

HTML

```
<div class="padre">
      <div class="hijo" name="primero">
            <ul>
                  <li>A1</li>
                  <li>A2</li>
                  <li>A3</li>
            </ul>
      </div>
      <div class="hijo" name="segundo">
            <ul>
                  <li>B1</li>
                  <li>B2</li>
                  <li>B3</li>
            </ul>
      </div>
</div>
```

jQuery

```
$('.padre').find('.hijo[name="segundo"] ul li').css('font-weight','bold');
```

Salida

- A1
- A2
- A3
- **B1**
- **B2**
- **B3**

Sección 8.6: Iterar sobre una lista de elementos jQuery

Cuando necesite iterar sobre la lista de elementos jQuery.

Considera esta estructura DOM:

```
<div class="container">
<div class="red one">RED 1 Info</div>
<div class="red two">RED 2 Info</div>
<div class="red three">RED 3 Info</div>
</div>
```

Para imprimir el texto presente en todos los elementos div con una clase de rojo:

```
$(".red").each(function(key, ele){
var text = $(ele).text();
console.log(text);
});
```

Consejo: `key` es el índice del elemento `div.red` sobre el que estamos iterando, dentro de su padre. `ele` es el elemento HTML, por lo que podemos crear un objeto jQuery a partir de él utilizando `$()` o `jQuery()`, de esta forma: `$(ele)`. Después, podemos llamar a cualquier método jQuery en el objeto, como `css()` o `hide()` etc. En este ejemplo, sólo extraemos el texto del objeto.

Sección 8.7: Selección de hermanos

Para seleccionar los hermanos de un elemento puede utilizar el método `.siblings()`.

Un ejemplo típico en el que se desea modificar los hermanos de un elemento es en un menú:

```
<ul class="menu">
      <li class="seleccionado">Inicio</li>
      <li>Blog</li>
      <li>Acerca de</li>
</ul>
```

Cuando el usuario hace clic en un elemento del menú, la clase `seleccionado` debe añadirse al elemento sobre el que se hace clic y eliminarse de sus *hermanos*:

```
$(".menu").on("click", "li", function () {
      $(this).addClass("seleccionado");
      $(this).siblings().removeClass("seleccionado");
});
```

El método toma un argumento `selector` opcional, que se puede utilizar si necesita restringir los tipos de hermanos que desea seleccionar:

```
$(this).siblings("li").removeClass("seleccionado");
```

Sección 8.8: Método closest()

Devuelve el primer elemento que coincide con el selector comenzando en el elemento y recorriendo el árbol DOM.

HTML

```
<div id="abc" class="fila">
      <div id="xyz" class="fila"></div>
      <p id="origen">Hola</p>
</div>
```

jQuery

```
var target = $('#origen').closest('.fila');
console.log("Fila más cercana:", target.attr('id') );
var target2 = $('#origen').closest('p');
console.log("p más cercano:", target2.attr('id') );
```

Salida

```
"Fila más cercana: abc"
"p más cercano: origen"
```

método `first()`: El método `first` devuelve el primer elemento del conjunto de elementos emparejados.

HTML

```
<div class='.primerEjemplo'>
      <p>Es el primer párrafo de un div.</p>
      <p>Es el segundo párrafo de un div.</p>
      <p>Es el tercer párrafo de un div.</p>
      <p>Es el cuarto párrafo de un div.</p>
      <p>Es el quinto párrafo de un div.</p>
</div>
```

JQuery

```
var primerParrafo = $("div p").first();
console.log("Primer párrafo:", firstParagraph.text());
```

Salida:

```
Primer párrafo: Es el primer párrafo de un div.
```

Capítulo 9: Manipulación de CSS

Sección 9.1: CSS - Getters y Setters

Getter CSS

La función **getter** `.css()` puede aplicarse a cada elemento DOM de la página como se muestra a continuación:

```
// Ancho renderizado en px como cadena de caracteres. ej: `150px`.
// Fíjate en la designación `as a string` - si necesitas un entero verdadero,
// consulta el método `$.width()`.
$("body").css("width");
```

Esta línea devolverá la **anchura calculada** del elemento especificado, cada propiedad CSS que proporciones entre paréntesis devolverá el valor de la propiedad para este elemento DOM `$("selector")`, si pides un atributo CSS que no existe obtendrás `undefined` como respuesta.

También puedes llamar al **getter CSS** con un array de atributos:

```
$("body").css(["animation","width"]);
```

esto devolverá un objeto de todos los atributos con sus valores:

```
Object {animation: "none 0s ease 0s 1 normal none running", width: "529px"}
```

Setter CSS

La función **setter** `.css()` también puede aplicarse a cada elemento DOM de la página.

```
$("selector").css("width", 500);
```

Esta sentencia establece el `ancho` de `$("selector")` a `500px` y devuelve el objeto jQuery para que puedas encadenar más métodos al selector especificado.

El **setter** `.css()` también se puede utilizar pasando un objeto de propiedades CSS y valores como:

```
$("body").css({"height": "100px", width:100, "padding-top":40, paddingBottom:"2em"});
```

Todos los cambios realizados por el definidor se añaden a la propiedad de estilo del elemento DOM, afectando así a los estilos de los elementos (a menos que el valor de la propiedad de estilo ya esté definido como `!important` en algún otro lugar de los estilos).

Sección 9.2: Aumento/Disminución de propiedades numéricas

Las propiedades CSS numéricas pueden incrementarse y decrementarse con la sintaxis `+=` y `-=`, respectivamente, utilizando el método `.css()`:

```
// Incrementar utilizando la sintaxis +=
$("#elemento-objetivo").css("font-size", "+=10");
// También puede especificar la unidad de incremento
$("#elemento-objetivo").css("width", "+=100pt");
$("#elemento-objetivo").css("top", "+=30px");
$("#elemento-objetivo").css("left", "+=3em");
// La disminución se realiza mediante la sintaxis -=
$("#elemento-objetivo").css("height", "-=50pt");
```

Sección 9.3: Establecer la propiedad CSS

Establecer un solo estilo:

```
$('#elemento-objetivo').css('color', '#000000');
```

Establecer varios estilos al mismo tiempo:

```
$('#elemento-objetivo').css({
      'color': '#000000',
      'font-size': '12pt',
      'float': 'left',
});
```

Sección 9.4: Obtener la propiedad CSS

Para obtener la propiedad CSS de un elemento puedes utilizar el método `.css(nombrePropiedad)`:

```
var color = $('#elemento').css('color');
var fontSize = $('#elemento').css('font-size');
```

Capítulo 10: Visibilidad del elemento

Parámetro	Detalles
Duración	Cuando se pasa, los efectos de `.hide()`, `.show()` y `.toggle()` son animados; el elemento(s) se desvanecerá gradualmente.

Sección 10.1: Visión general

```
$(element).hide() // establece display: none
$(element).show() // establece display al su valor original
$(element).toggle() // alterna entre los dos
$(element).is(':visible') // devuelve true o false
$('element:visible') // coincidirá con todos los elementos visibles
$('element:hidden') // coincidirá con todos los elementos ocultos
$('element').fadeIn(); // muestra el elemento
$('element').fadeOut(); // oculta el elemento
$('element').fadeIn(1000); // mostrar el elemento mediante un temporizador
$('element').fadeOut(1000); // ocultar el elemento mediante un temporizador
// mostrar el elemento mediante un temporizador y una función callback
$('element').fadeIn(1000, function(){
     // código a ejecutar
});
// ocultar el elemento mediante un temporizador y una función callback
$('element').fadeOut(1000, function(){
     // código a ejecutar
});
```

Sección 10.2: Alternar posibilidades

Caso simple de `toggle()`

```
function toggleBasico() {
     $(".objetivo1").toggle();
}
```

Con *duración* específica

```
function toggleDuracion() {
     $(".objetivo2").toggle("slow"); // A millisecond duration value is also acceptable
}
```

...y *callback*

```
function toggleCallback() {
     $(".objetivo3").toggle("slow",function(){alert('ahora haz algo');});
}
```

...o con *flexibilización* y *callback*.

```
function toggleFlexibilizacionYCallback() {
     // Puedes utilizar jQueryUI como el núcleo sólo es compatible con lineal y swing easings
     $(".objetivo4").toggle("slow","linear",function(){alert('ahora haz algo');});
}
```

...o con una variedad de *opciones*.

```
function toggleConOpciones() {
     $(".objetivo5").toggle(
          { // Ver todas las opciones posibles en: api.jquery.com/toggle/#toggle-options
               duration:1000, // milisegundos
               easing:"linear",
```

```
                done:function(){
                    alert('ahora haz algo');
                }
            }
        );
}
```

También es posible utilizar una *diapositiva* como animación con `slideToggle()`

```
function toggleDiapositiva() {
    $(".objetivo6").slideToggle(); // Se anima de arriba a abajo, en lugar de la esquina superior
}
```

...o desvanecer cambiando la opacidad con `fadeToggle()`

```
function toggleFading() {
    $( ".objetivo7" ).fadeToggle("slow")
}
```

...o activar una clase con `toggleClass()`

```
function toggleClass() {
    $(".objetivo8").toggleClass('active');
}
```

Un caso común es utilizar `toggle()` para mostrar un elemento mientras se oculta el otro (misma clase)

```
function toggleX() {
    $(".objetivoX").toggle("slow");
}
```

Todos los ejemplos anteriores pueden consultarse aquí

Capítulo 11: Añadir (append)

Parámetros	Detalles
content	Tipos posibles: Elemento, cadena HTML, texto, array, objeto o incluso una función que devuelva una cadena.

Sección 11.1: Uso consecutivo eficaz de .append()

Empezando:

HTML

```
<table id='mi-tabla' width='960' height='500'></table>
```

JS

```
var datos = [
     { tipo: "Nombre", contenido: "John Doe" },
     { tipo: "Fecha Nacimiento", contenido: "01/01/1970" },
     { tipo: "Salario", contenido: "$40000000" },
     // ... más de 300 filas ...
     { tipo: " Sabor favorito ", contenido: "Agrio" }
];
```

Añadir dentro de un bucle

Acabas de recibir un gran array de datos. Ahora es el momento de hacer un bucle a través de ellos y renderizarlos en la página.

Su primer pensamiento puede ser hacer algo como esto:

```
var i; // <- el número del item actual
var cuenta = datos.length; // <- el total
var fila; // <- para mantener una referencia a nuestro objeto fila
// Recorrer el array
for ( i = 0; i < cuenta; ++i ) {
     fila = datos[ i ];
     // Coloca toda la fila en tu tabla
     $('#mi-tabla').append(
          $('<tr></tr>').append(
               $('<td></td>').html(fila.tipo),
               $('<td></td>').html(fila.contenido)
          )
     );
}
```

Esto es *perfectamente válido* y te dará exactamente lo que esperas, pero...

NO lo hagas.

¿Recuerdas esas **más de 300** filas de datos?

Cada uno forzará al navegador a recalcular los valores de anchura, altura y posicionamiento de cada elemento, junto con cualquier otro estilo - a menos que estén separados por un límite de diseño, que desafortunadamente para este ejemplo (ya que son descendientes de un elemento `<table>`), no pueden.

Con cantidades pequeñas y pocas columnas, esta penalización de rendimiento será sin duda insignificante. Pero queremos que cada milisegundo cuente.

Mejores opciones

1. **Añadir a un array separado, añadir después de completar el bucle**

```
/**
* El recorrido repetido del DOM (siguiendo el árbol de elementos hasta
* llegar a lo que estás buscando - como nuestra <table>) también debe evitarse siempre que sea
* posible.
*/
// Guarda la tabla en caché en una variable y utilízala hasta que creas que se ha eliminado
var $miTabla = $('#mi-tabla');
// Para contener nuestros nuevos objetos jQuery <tr>.
var elementoFila = [];
var cuenta = datos.length;
var i;
var fila;
// Recorrer el array
for ( i = 0; i < cuenta; ++i ) {
     elementosFila.push(
          $('<tr></tr>').append(
               $('<td></td>').html(fila.tipo),
               $('<td></td>').html(fila.contenido)
          )
     );
}
// Por último, inserte TODAS las filas a la vez
$miTabla.append(elementosFila);
```

De estas opciones, ésta es la que más depende de jQuery.

2. **Uso de métodos Array.* modernos**

```
var $miTabla = $('#mi-tabla');
// Bucle con el método .map()
// - Esto nos dará un nuevo array basado en el resultado de nuestra función callback
var elementosFila = datos.map(function ( fila ) {
     // Crea una fila
     var $fila = $('<tr></tr>');
     // Crea las columnas
     var $tipo = $('<td></td>').html(fila.tipo);
     var $contenido = $('<td></td>').html(fila.contenido);
     // Añadir las columnas a la fila
     $fila.append($tipo, $contenido);
     // Añadir al array recién generado
     return $fila;
     });
// Por último, introduce TODAS las filas en tu tabla
$miTabla.append(elementosFila);
```

Funcionalmente equivalente al anterior, sólo que más fácil de leer.

3. **Uso de cadenas de caracteres de HTML (en lugar de los métodos integrados de jQuery)**

```
// ...
var elementosFila = datos.map(function ( fila ) {
     var filaHTML = '<tr><td>';
     filaHTML += fila.tipo;
     filaHTML += '</td><td>';
     filaHTML += fila.contenido;
     filaHTML += '</td></tr>';
     return filaHTML;
});
// El uso de .join('') combina todas las cadenas de caracteres separadas en una sola
$miTabla.append(elementosFila.join(''));
```

Perfectamente válido, pero, de nuevo, **no recomendado**. Esto obliga a jQuery para analizar una gran cantidad de texto a la vez y no es necesario. jQuery es muy bueno en lo que hace cuando se utiliza correctamente.

4. **Crear elementos manualmente, añadir al fragmento de documento**

```
var $miTabla = $(document.getElementById('mi-tabla'));
/**
* Crear un fragmento de documento para contener nuestras columnas
* - después de añadir esto a cada fila, se vacía
* para que podamos reutilizarlo en la siguiente iteración.
*/
var fragmentoColumna = document.createDocumentFragment();
/**
* Haz un bucle sobre el array usando .reduce() esta vez.
* Obtenemos una salida ordenada y sin efectos secundarios.
* - En este ejemplo, el resultado será un
* fragmento del documento que contiene todos los elementos <tr>..
*/
var fragmentoFila = datos.reduce(function ( fragmento, fila ) {
      // Crea un fila
      var elFila = document.createElement('tr');
      // Crear las columnas y los nodos de texto interiores
      var elTipo = document.createElement('td');
      var tipoTexto = document.createTextNode(fila.tipo);
      elTipo.appendChild(tipoTexto);
      var elContenido = document.createElement('td');
      var contenidoTexto = document.createTextNode(fila.contenido);
      elContenido.appendChild(contenidoTexto);
      // Añadir las columnas al fragmento de columna
      // - esto sería útil si las columnas se iteraran por separado
      // pero en este ejemplo es sólo para mostrar.
      fragmentoColumna.appendChild(elTipo);
      fragmentoColumna.appendChild(elContenido);
      elFila.appendChild(fragmentoColumna);
      // Añadir elFila al fragmento - esto actúa como un buffer temporal para
      // acumular varios nodos DOM antes de la inserción masiva
      fragmento.appendChild(elFila);
      return fragmento;
}, document.createDocumentFragment());
// Ahora vuelca todo el fragmento en tu tabla
$miTabla.append(fragmentoFila);
```

Mi favorito personal. Esto ilustra una idea general de lo que jQuery hace en un nivel inferior.

Profundizar

- Visor de fuentes jQuery
- `Array.prototype.join()`
- `Array.prototype.map()`
- `Array.prototype.reduce()`
- `document.createDocumentFragment()`
- `document.createTextNode()`
- Fundamentos de Google Web - Rendimiento

Sección 11.2: append de jQuery

HTML

```
<p> Este es un bonito </p>
<p> Me gusta </p>
<ul>
     <li> Punto 1 de la lista</li>
     <li>Punto 2 de la lista</li>
     <li>Punto 3 de la lista</li>
</ul>
<button id="btn-1">Añadir texto</button>
<button id="btn-2">Añadir un elemento a la lista</button>
```

Script

```
$("#btn-1").click(function(){
     $("p").append(" <b>Libro</b>.");
});
$("#btn-2").click(function(){
          $("ul").append("<li>Elemento de lista añadido</li>");
     });
});
```

Sección 11.3: Añadir un elemento a un contenedor

Solución 1:

```
$('#padre').append($('#hijo'));
```

Solución 2:

```
$('#hijo').appendTo($('#padre'));
```

Ambas soluciones anexan el elemento `#hijo` (añadiendo al final) al elemento `#padre`.

Antes:

```
<div id="padre">
     <span>otro contenido</span>
</div>
<div id="hijo">
</div>
```

Después:

```
<div id="padre">
     <span>otro contenido</span>
     <div id="hijo">
     </div>
</div>
```

Nota: Cuando añada contenido que ya existe en el documento, este contenido se eliminará de su contenedor principal original y se añadirá al nuevo contenedor principal. Así que no puedes usar `.append()` o `.appendTo()` para clonar un elemento. Si necesitas un clon usa `.clone()` -> http://api.jquery.com/clone/

Capítulo 12: Anteponer (prepend)

Sección 12.1: Anteponer un elemento a un contenedor

Solución 1:

```
$('#padre').prepend($('#hijo'));
```

Solución 2:

```
$('#hijo').prependTo($('#padre'));
```

Ambas soluciones anteponen el elemento `#hijo` (añadiendo al principio) al elemento `#padre`.

Antes:

```
<div id="padre">
     <span>otro contenido</span>
</div>
<div id="hijo">
</div>
```

Después:

```
<div id="padre">
     <div id="hijo">
     </div>
     <span>otro contenido</span>
</div>
```

Sección 12.2: Método prepend

`prepend()` - Inserta contenido, especificado por el parámetro, al principio de cada elemento del conjunto de elementos coincidentes.

1. **prepend(content [, content])**

```
// con cadena de caracteres html
jQuery('#padre').prepend('<span>hijo</span>');
// o puede utilizar el objeto jQuery
jQuery('#padre').prepend($('#hijo'));
// o puede utilizar elementos múltiples separados por comas para anteponer
jQuery('#padre').prepend($('#hijo1'),$('#hijo2'));
```

2. **prepend(function)**

JQuery `version: 1.4` en adelante se puede utilizar la función de devolución de llamada como argumento. Donde puedes obtener argumentos como la posición del índice del elemento en el conjunto y el valor HTML antiguo del elemento. Dentro de la función, esto se refiere al elemento actual en el conjunto.

```
jQuery('#padre').prepend(function(i,oldHTML){
     // devuelve el valor a añadir
     return '<span>hijo</span>';
});
```

Capítulo 13: Obtener y establecer la anchura y altura de un elemento

Sección 13.1: Obtener y establecer de la anchura y la altura (ignorando el borde)

Obtener anchura y altura:

```
var anchura = $('#elemento-objetivo').width();
var altura = $('#elemento-objetivo').height();
```

Establecer la anchura y la altura:

```
$('#elemento-objetivo').width(50);
$('#elemento-objetivo').height(100);
```

Sección 13.2: Obtener y establecer de innerWidth y innerHeight (ignorando el relleno y el borde)

Obtener anchura y altura:

```
var anchura = $('#elemento-objetivo').innerWidth();
var altura = $('#elemento-objetivo').innerHeight();
```

Establecer la anchura y la altura:

```
$('#elemento-objetivo').innerWidth(50);
$('#elemento-objetivo').innerHeight(100);
```

Sección 13.3: Obtención y configuración de outerWidth y outerHeight (incluidos el relleno y el borde)

Obtener anchura y altura (sin margen):

```
var anchura = $('#elemento-objetivo').outerWidth();
var altura = $('#elemento-objetivo').outerHeight();
```

Obtener anchura y altura (incluido el margen):

```
var anchura = $('#elemento-objetivo').outerWidth(true);
var altura = $('#elemento-objetivo').outerHeight(true);
```

Establecer la anchura y la altura:

```
$('#elemento-objetivo').outerWidth(50);
$('#elemento-objetivo').outerHeight(100);
```

Capítulo 14: Método .animate() de jQuery

Parámetro	Detalles
`properties`	Un objeto de propiedades y valores CSS hacia el que se moverá la animación
`duration`	(por defecto: 400) Cadena de caracteres o número que determina la duración de la animación
`easing`	(por defecto: swing) Cadena de caracteres que indica la función de suavizado que se utilizará para la transición.
`complete`	Una función a la que llamar una vez completada la animación, llamada una vez por cada elemento emparejado.
`start`	Especifica una función que se ejecutará cuando comience la animación.
`step`	Especifica una función que se ejecutará en cada paso de la animación.
`queue`	Un valor booleano que especifica si se coloca o no la animación en la cola de efectos.
`progress`	Especifica una función que se ejecutará después de cada paso de la animación.
`done`	Especifica una función que se ejecutará cuando finalice la animación.
`fail`	Especifica una función que se ejecutará si la animación no se completa.
`specialEasing`	Un mapa de una o más propiedades CSS de los parámetros de estilos, y sus correspondientes funciones de suavizado.
`always`	Especifica una función que se ejecutará si la animación se detiene sin completarse.

Sección 14.1: Animación con callback

A veces tenemos que cambiar la posición de las palabras de un lugar a otro o reducir el tamaño de las palabras y cambiar el color de las palabras de forma automática para mejorar el atractivo de nuestro sitio web o aplicaciones web. JQuery ayuda mucho con este concepto usando `fadeIn()`, `hide()`, `slideDown()` pero su funcionalidad es limitada y sólo hace la tarea específica que se le asigna.

Jquery soluciona este problema proporcionando un método sorprendente y flexible llamado `.animate()`. Este método permite establecer animaciones personalizadas que se utiliza propiedades css que dan permiso para volar sobre las fronteras. Por ejemplo, si le damos a la propiedad de estilo css `width:200;` y la posición actual del elemento DOM es 50, el método `animate` reducirá el valor de la posición actual del valor css dado y animará ese elemento a 150. Pero no necesitamos preocuparnos por esta parte porque el motor de animación se encargará de ello.

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js"></script>
<script>
     $("#btn1").click(function(){
          $("#caja").animate({width: "200px"});
     });
</script>
<button id="btn1">Animate Width</button>
<div id="caja" style="background:#98bf21;height:100px;width:100px;margin:6px;"></div>
```

Lista de propiedades de estilo css que permite el método `.animate()`.

```
backgroundPositionX, backgroundPositionY, borderWidth, borderBottomWidth, borderLeftWidth,
borderRightWidth, borderTopWidth, borderSpacing, margin, marginBottom, marginLeft, marginRight,
marginTop, outlineWidth, padding, paddingBottom, paddingLeft, paddingRight, paddingTop, height,
width, maxHeight, maxWidth, minHeight, minWidth, fontSize, bottom, left, right, top,
letterSpacing,
wordSpacing, lineHeight, textIndent,
```

Velocidad especificada en el método `.animate()`.

```
milliseconds (Ex: 100, 1000, 5000, etc.),
"slow",
"fast"
```

Aligeramiento especificado en el método `.animate()`.

"swing"

"linear"

He aquí algunos ejemplos con opciones de animación complejas.

Ej. 1:

```
$( "#libro" ).animate({
     width: [ "toggle", "swing" ],
     height: [ "toggle", "swing" ],
     opacity: "toggle"
}, 5000, "linear", function() {
     $( this ).after( "<div>Animación completa.</div>" );
});
```

Ej. 2:

```
$("#caja").animate({
     height: "300px",
     width: "300px"
}, {
     duration: 5000,
     easing: "linear",
     complete: function(){
          $(this).after("<p>¡Animación está completada!</p>");
     }
});
```

Capítulo 15: Objetos diferidos y promesas en jQuery

Las promesas de jQuery son una forma inteligente de encadenar operaciones asíncronas de forma modular. Este reemplaza la vieja escuela de anidamiento de callbacks, que no son tan fáciles de reorganizar.

Sección 15.1: jQuery ajax() success, error VS .done(), .fail()

success y error: Un callback de **éxito** que se invoca al completar con éxito una solicitud Ajax.

Un callback **fallido** que se invoca en caso de que se produzca algún error al realizar la petición.

Ejemplo:

```
$.ajax({
     url: 'URL',
     type: 'POST',
     data: tusDatos,
     datatype: 'json',
     success: function (data) { successFunction(data); },
     error: function (jqXHR, textStatus, errorThrown) { errorFunction(); }
});
```

.done() y .fail():

.ajax().done(function(data, textStatus, jqXHR){}); Sustituye al método .success() que quedó obsoleto en jQuery 1.8. Se trata de una construcción alternativa para la función callback de éxito anterior.

.ajax().fail(function(jqXHR, textStatus, errorThrown){}); Sustituye al método .error() que quedó obsoleto en jQuery 1.8. Se trata de una construcción alternativa para la función callback completa anterior.

Ejemplo:

```
$.ajax({
     url: 'URL',
     type: 'POST',
     data: tusDatos,
     datatype: 'json'
})
.done(function (data) { successFunction(data); })
.fail(function (jqXHR, textStatus, errorThrown) { serrorFunction(); });
```

Sección 15.2: Creación básica de promesas

He aquí un ejemplo muy sencillo de una función que "*promete* proceder cuando transcurra un tiempo determinado". Lo hace creando un nuevo objeto `Deferred`, que se resuelve más tarde y devuelve la promesa del `Deferred`:

```
function esperarPromesa(milisegundos){
     // Crear un nuevo objeto Deferred utilizando el método estático de jQuery
     var def = $.Deferred();
     // Hacer algún trabajo asíncrono - en este caso un simple temporizador
     setTimeout(function(){
          // Trabajo terminado... resolver el aplazado, por lo que esa promesa procederá
          def.resolve();
     }, milisegundos);
     // Devuelve inmediatamente una "promesa de proceder cuando termine el tiempo de espera"
     return def.promise();
}
```

Y usarlo así:

```
esperarPromesa(2000).then(function(){
      console.log("He esperado lo suficiente");
});
```

Capítulo 16: AJAX

Parámetro	Detalles
url	Especifica la URL a la que se enviará la solicitud
settings	Un objeto que contiene numerosos valores que afectan al comportamiento de la solicitud
type	El método HTTP que se utilizará para la solicitud
data	Datos que debe enviar la solicitud
success	Una función callback que se llamará si la solicitud tiene éxito
error	Un callback para gestionar el error
statusCode	Un objeto de códigos HTTP numéricos y funciones que se llamarán cuando la respuesta tenga el código correspondiente
dataType	Tipo de datos que espera recibir del servidor
contentType	Tipo de contenido de los datos a enviar al servidor. Por defecto es "`application/x-www-form-urlencoded; charset=UTF-8`"
context	Especifica el contexto que se utilizará dentro de los callbacks, normalmente `this` que se refiere al objetivo actual.

Sección 16.1: Manejo de códigos de respuesta HTTP con $.ajax()

Además de los callbacks de las promesas `.done`, `.fail` y `.always`, que se activan en función de si la solicitud se ha realizado correctamente o no, existe la opción de activar una función cuando el servidor devuelve un código de estado HTTP específico. Esto puede hacerse utilizando el parámetro `statusCode`.

```
$.ajax({
      type: {POST or GET or PUT etc.},
      url: {server.url},
      data: {algunDato: true},
      statusCode: {
            404: function(responseObject, textStatus, jqXHR) {
                  // No se ha encontrado contenido (404)
                  // Este código se ejecutará si el servidor devuelve una respuesta 404
            },
            503: function(responseObject, textStatus, errorThrown) {
                  // Servicio no disponible (503)
                  // Este código se ejecutará si el servidor devuelve una respuesta 503
            }
      }
})
.done(function(data){
      alert(data);
})
.fail(function(jqXHR, textStatus){
      alert('Algo salió mal: ' + textStatus);
})
.always(function(jqXHR, textStatus) {
      alert('La petición Ajax ha finalizado')
});
```

Como dice la documentación oficial de jQuery:

> Si la solicitud tiene éxito, las funciones de código de estado toman los mismos parámetros que el callback de éxito; si resulta en un error (incluida la redirección 3xx), toman los mismos parámetros que el callback de error.

Sección 16.2: Utilizar Ajax para enviar un formulario

A veces puede tener un formulario y desea enviarlo usando ajax.

Supongamos que tiene este sencillo formulario –

```
<form id="ajax_form" action="form_action.php">
      <label for="name">Nombre :</label>
      <input name="name" id="name" type="text" />
      <label for="name">Email :</label>
      <input name="email" id="email" type="text" />
      <input type="submit" value="Enviar" />
</form>
```

Se puede utilizar el siguiente código jQuery (dentro de una llamada a `$(document).ready`) –

```
$('#ajax_form').submit(function(event){
      event.preventDefault();
      var $form = $(this);
      $.ajax({
            type: 'POST',
            url: $form.attr('action'),
            data: $form.serialize(),
            success: function(data) {
                  // Hacer algo con la respuesta
            },
            error: function(error) {
                  // Hacer algo con el error
            }
      });
});
```

Explicación

- `var $form = $(this)` - el formulario, almacenado en caché para su reutilización
- `$('#ajax_form').submit(function(event){` - Cuando el formulario con ID "`ajax_form`" es enviado ejecuta esta función y pasa el evento como parámetro.
- `event.preventDefault();` - Evita que el formulario se envíe normalmente (Alternativamente podemos usar `return false` después de la sentencia `ajax({});` que tendrá el mismo efecto)
- `url: $form.attr('action'),` - Obtener el valor del atributo "`action`" del formulario y usarlo para la propiedad "`url`".
- `data: $form.serialize(),` - Convierte las entradas dentro del formulario en una cadena adecuada para enviar al servidor. En este caso devolverá algo como "`name=Bob&email=bob@bobsemailaddress.com`"

Sección 16.3: Ejemplos todo en uno

Ajax Get:

Solución 1:

```
$.get('url.html', function(data){
      $('#actualizar-caja').html(data);
});
```

Solución 2:

```
$.ajax({
     type: 'GET',
     url: 'url.php',
}).done(function(data){
     $('#actualizar-caja').html(data);
}).fail(function(jqXHR, textStatus){
     alert('Error ocurrido: ' + textStatus);
});
```

Ajax Load: Otro método ajax get creado para simplicidad

```
$('#actualizar-caja').load('url.html');
```

`.load` también se puede llamar con datos adicionales. La parte de datos se puede proporcionar como cadena u objeto.

```
$('#actualizar-caja').load('url.php', {data: "algo"});
$('#actualizar-caja').load('url.php', "data=algo");
```

Si se llama a `.load` con un método callback, la petición al servidor será un post

```
$('#actualizar-caja').load('url.php', {data: "algo"}, function(resolve){
     // haz algo
});
```

Ajax Post:

Solución 1:

```
$.post('url.php',
     {date1Name: data1Value, date2Name: data2Value}, //datos que deben publicarse
     function(data){
          $('#actualizar-caja').html(data);
     }
);
```

Solución 2:

```
$.ajax({
     type: 'Post',
     url: 'url.php',
     data: {date1Name: data1Value, date2Name: data2Value} //datos que deben publicarse
}).done(function(data){
     $('#actualizar-caja').html(data);
}).fail(function(jqXHR, textStatus){
     alert('Error ocurrido: ' + textStatus);
});
```

Ajax Post JSON:

```
var subirDatos = {
     Nombre: nombre,
     Direccion: direccion,
     Telefono: telefono
};
$.ajax({
     type: "POST",
     url: "url.php",
     dataType: "json",
     data: JSON.stringfy(subirDatos),
     success: function (data) {
          //aquí los datos variables están en formato JSON
     }
});
```

Ajax Get JSON:

Solución 1:

```
$.getJSON('url.php', function(data){
     // aquí los datos variables están en formato JSON
});
```

Solución 2:

```
$.ajax({
     type: "Get",
     url: "url.php",
     dataType: "json",
     data: JSON.stringfy(postData),
     success: function (data) {
          // aquí los datos variables están en formato JSON
     },
     error: function(jqXHR, textStatus){
          alert('Error ocurrido: ' + textStatus);
     }
});
```

Sección 16.4: Subida de archivos Ajax

1. Un ejemplo sencillo y completo

Podríamos utilizar este código de ejemplo para cargar los archivos seleccionados por el usuario cada vez que se realice una nueva selección de archivos.

```
<input type="file" id="archivo-input" multiple>
```

```
var archivos;
var fdata = new FormData();
$("#archivo-input").on("change", function (e) {
     archivos = this.archivos;
     $.each(archivos, function (i, archivo) {
          fdata.append("file" + i, archivo);
     });
     fdata.append("NombreCompleto", "John Doe");
     fdata.append("Genero", "Masculino");
     fdata.append("Edad", "24");
     $.ajax({
          url: "/Test/Url",
          type: "post",
          data: fdata, // añadir el objeto FormData al parámetro data
          processData: false, // indicar a jquery que no procese datos
          contentType: false, // indicar a jquery que no establezca content-type
          success: function (response, status, jqxhr) {
               // manejar éxito
          },
          error: function (jqxhr, status, errorMessage) {
               // manejar error
          }
     });
});
```

Ahora vamos a desglosar esto e inspeccionarlo parte por parte.

2. Trabajar con entradas de archivos

Este documento MDN (Utilización de archivos de aplicaciones web) es una buena lectura acerca de varios métodos sobre cómo manejar entradas de archivos. Algunos de estos métodos también se utilizarán en este ejemplo.

Antes de pasar a la carga de archivos, primero tenemos que dar al usuario una forma de seleccionar los archivos que desea cargar. Para ello utilizaremos un `file input`. La propiedad `multiple` permite seleccionar más de un archivo, puedes quitarla si quieres que el usuario seleccione un archivo a la vez.

```
<input type="file" id="file-input" multiple>
```

Utilizaremos el evento `change event` para capturar los archivos.

```
var files;
$("#file-input").on("change", function(e){
     files = this.files;
});
```

Dentro de la función manejador, accedemos a los ficheros a través de la propiedad `files` de nuestro `input`. Esto nos da un FileList, que es un array como objeto.

3. Crear y rellenar el formulario FormData

Para subir archivos con Ajax vamos a utilizar FormData.

```
var fdata = new FormData();
```

FileList que hemos obtenido en el paso anterior es un objeto tipo array y puede ser iterado usando varios métodos incluyendo el bucle for, el bucle for...of y jQuery.each. En este ejemplo nos ceñiremos a jQuery.

```
$.each(files, function(i, file) {
     //...
});
```

Utilizaremos el método append de FormData para añadir los archivos a nuestro objeto formdata.

```
$.each(files, function(i, file) {
     fdata.append("file" + i, file);
});
```

También podemos añadir otros datos que queramos enviar de la misma manera. Digamos que queremos enviar alguna información personal que hemos recibido del usuario junto con los archivos. Podríamos añadir esta información en nuestro objeto formdata.

```
fdata.append("FullName", "John Doe");
fdata.append("Gender", "Male");
fdata.append("Age", "24");
//...
```

4. Envío de archivos con Ajax

```
$.ajax({
     url: "/Test/Url",
     type: "post",
     data: fdata, // añadir el objeto FormData al parámetro data
     processData: false, // indicar a jquery que no procese datos
     contentType: false, // indicar a jquery que no establezca content-type
     success: function (response, status, jqxhr) {
          // manejar éxito
     },
     error: function (jqxhr, status, errorMessage) {
          // manejar error
     }
});
```

Establecemos las propiedades `processData` y `contentType` a `false`. Esto se hace para que los archivos puedan ser enviados al servidor y ser procesados por el servidor correctamente.

Capítulo 17: Casilla de verificación Seleccionar todo con marcar/desmarcar automáticamente en otro cambio de casilla

He utilizado varios ejemplos y respuestas de Stackoverflow para llegar a este ejemplo realmente sencillo sobre cómo gestionar la casilla de verificación "seleccionar todo" junto con un check/uncheck automático si cambia el estado de alguna de las casillas de verificación del grupo.

Restricción: El id de "seleccionar todo" debe coincidir con los nombres de entrada para crear el grupo de seleccionar todo. En el ejemplo, el ID de la entrada "seleccionar todo" es cbGroup1. Los nombres de entrada también son cbGroup1.

El código es muy corto, no abunda la sentencia if (consume tiempo y recursos).

Sección 17.1: Seleccionar todas las casillas de verificación con las casillas de verificación de grupo correspondientes

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
<p>
<input id="cbGroup1" type="checkbox">Select all
<input name="cbGroup1" type="checkbox" value="value1_1">Group1 value 1
<input name="cbGroup1" type="checkbox" value="value1_2">Group1 value 2
<input name="cbGroup1" type="checkbox" value="value1_3">Group1 value 3
</p>
<p>
<input id="cbGroup2" type="checkbox">Select all
<input name="cbGroup2" type="checkbox" value="value2_1">Group2 value 1
<input name="cbGroup2" type="checkbox" value="value2_2">Group2 value 2
<input name="cbGroup2" type="checkbox" value="value2_3">Group2 value 3
</p>
<script type="text/javascript" language="javascript">
     $("input").change(function() {
          $('input[name=\''+this.id+'\']').not(this).prop('checked', this.checked);
          $('#'+this.name).prop('checked', $('input[name=\''+this.name+'\']').length ===
          $('input[name=\''+this.name+'\']').filter(':checked').length);
     });
</script>
```

Capítulo 18: Plugins

Sección 18.1: Plugins – Introducción

La API jQuery puede ampliarse añadiendo elementos a su prototipo. Por ejemplo, la API existente ya dispone de muchas funciones como `.hide()`, `.fadeIn()`, `.hasClass()`, etc.

El prototipo jQuery se expone a través de `$.fn`, el código fuente contiene la línea

```
jQuery.fn = jQuery.prototype
```

Añadir funciones a este prototipo permitirá que esas funciones estén disponibles para ser llamadas desde cualquier objeto jQuery construido (lo que se hace implícitamente con cada llamada a jQuery, o cada llamada a `$` si lo prefieres).

Un objeto jQuery construido contendrá un array interno de elementos basado en el selector que se le haya pasado. Por ejemplo, `$('.active')` construirá un objeto jQuery que contendrá elementos con la clase active en el momento de la llamada (es decir, no se trata de un conjunto de elementos activos).

El valor **`this`** dentro de la función plugin se referirá al objeto jQuery construido. Como resultado, esto se utiliza para representar el conjunto coincidente.

Plugin básico:

```
$.fn.highlight = function() {
     this.css({ background: "yellow" });
};
// Utiliza el ejemplo:
$("span").highlight();
```

Ejemplo de jsFiddle

Encadenabilidad y reutilización

A diferencia del ejemplo anterior, se espera que los plugins de jQuery sean **encadenables**.

Lo que esto significa es la posibilidad de encadenar múltiples Métodos a una misma Colección de Elementos como `$(".warn").append("¡ATENCIÓN! ").css({color:"red"})` (mira como usamos el método `.css()` después del `.append()`, ambos métodos se aplican sobre la misma Colección `.warn`)

Permitir el uso del mismo plugin en diferentes colecciones con diferentes opciones de personalización juega un papel importante en la **Personalización / Reutilización**.

```
(function($) {
     $.fn.highlight = function( custom ) {
          // Ajustes por defecto
          var settings = $.extend({
               color : "", // Color de texto por defecto
               background : "yellow" // Por defecto, fondo amarillo
          }, custom);
          return this.css({ // `return this` mantiene la encadenabilidad de los métodos
               color : settings.color,
               backgroundColor : settings.background
          });
     };
}( jQuery ));
// Utilizar la configuración predeterminada
$("span").highlight(); // puedes encadenar otros métodos
// Utilizar ajustes personalizados
$("span").highlight({
     background: "#f00",
     color: "white"
});
```

Demostración de jsFiddle

Libertad

Los ejemplos anteriores están en el ámbito de la comprensión básica de la creación de plugins. No hay que restringir al usuario a un conjunto limitado de opciones de personalización.

Digamos por ejemplo que usted quiere construir un plugin `.highlight()` donde usted puede pasar una cadena de **texto** deseada que será resaltada y permitir la máxima libertad con respecto a los estilos:

```
//...
// Ajustes por defecto
var settings = $.extend({
     text : "", // texto a resaltar
     class : "highlight" // referencia a la clase CSS
}, custom);
return this.each(function() {
     // tu lógica de resaltado de palabras aquí
});
//...
```

el usuario puede ahora pasar un **texto** deseado y tener un control total sobre los estilos añadidos utilizando una clase CSS personalizada:

```
$("#content").highlight({
     text : "hello",
     class : "makeYellowBig"
});
```

Ejemplo de jsFiddle

Créditos

Muchas gracias a todas las personas de Stack Overflow Documentation que ayudaron a proporcionar este contenido, más cambios pueden ser enviados a web@petercv.com para que el nuevo contenido sea publicado o actualizado.

Traductor al español

rortegag

A.J	Capítulos 1, 4 y 8
acdcjunior	Capítulos 1 y 4
Alex	Capítulo 15
Alex Char	Capítulo 10
Alon Eitan	Capítulo 5
amflare	Capítulos 1 y 5
Andrew Brooke	Capítulo 16
Anil	Capítulo 1
Arun Prasad E S	Capítulo 16
Ashiquzzaman	Capítulo 15
Ashkan Mobayen Khiabani	Capítulos 9, 11, 12, 13 y 16
Assimilater	Capítulo 7
Athafoud	Capítulo 16
ban17	Capítulo 4
Ben H	Capítulo 16
Bipon	Capítulos 3 y 11
Brandt Solovij	Capítulo 9
Brock Davis	Capítulo 7
bwegs	Capítulo 1
Castro Roy	Capítulo 2
charlietfl	Capítulo 6
csbarnes	Capítulo 16
Darshak	Capítulo 11
David	Capítulo 2
DefyGravity	Capítulo 7
DelightedD0D	Capítulo 2
Deryck	Capítulos 7 y 11
devlin carnate	Capítulo 2
dlsso	Capítulo 8
Dr. J. Testington	Capítulo 16
Emanuel Vintilă	Capítulo 5
empiric	Capítulos 11 y 12
Flyer53	Capítulo 11
Fueled By Coffee	Capítulo 1
Gone Coding	Capítulos 6 y 15
hasan	Capítulos 2 y 18
Horst Jahns	Capítulo 6
Iceman	Capítulo 2
Igor Raush	Capítulo 1
J F	Capítulo 5
j08691	Capítulo 9
Jatniel Prinsloo	Capítulo 6
jkdev	Capítulos 1 y 5

JLF	Capítulo 2
John C	Capítulos 1 y 16
John Slegers	Capítulo 2
Jonathan Michalik	Capítulo 9
Joram van den Boezem	Capítulo 5
kapantzak	Capítulo 2
Kevin Katzke	Capítulo 1
Keyslinger	Capítulo 2
Lacrioque	Capítulo 16
Liam	Capítulo 5
Luca Putzu	Capítulos 1 y 6
Mark Schultheiss	Capítulos 5 y 7
mark.hch	Capítulo 8
martincarlin87	Capítulo 7
Matas Vaitkevicius	Capítulo 1
Melanie	Capítulo 2
Mottie	Capítulo 1
Neal	Capítulo 1
Nhan	Capítulo 5
ni8mr	Capítulo 1
Nico Westerdale	Capítulo 5
Nirav Joshi	Capítulo 16
NotJustin	Capítulo 6
Nux	Capítulo 2
ochi	Capítulo 4
Ozan	Capítulo 16
Pranav C Balan	Capítulo 12
Proto	Capítulo 11
Renier	Capítulo 3
rmondesilva	Capítulo 8
Roko C. Buljan	Capítulos 1, 9 y 18
Rupali Pemare	Capítulo 10
Scimonster	Capítulos 4 y 5
secelite	Capítulo 5
SGS Venkatesh	Capítulo 8
Shaunak D	Capítulos 1, 2 y 16
Shekhar Pankaj	Capítulo 2
Shlomi Haver	Capítulo 9
Simplans	Capítulo 14
Sorangwala Abbasali	Capítulos 2 y 9
ssb	Capítulo 2
still_learning	Capítulo 7
sucil	Capítulo 8
Suganya	Capítulo 1
Sunny R Gupta	Capítulo 6
Sverri M. Olsen	Capítulo 8
TheDeadMedic	Capítulo 5
Theodore K.	Capítulo 10
The_Outsider	Capítulos 8 y 10
Travis J	Capítulos 1, 2 y 18
Upal Roy	Capítulo 5
user1851673	Capítulo 17
user2314737	Capítulo 10

VJS	Capítulo 14
Washington Guedes	Capítulo 6
WOUNDEDStevenJones	Capítulo 2
Yosvel Quintero	Capítulos 1 y 16
Zakaria Acharki	Capítulo 6
Zaz	Capítulos 2 y 10