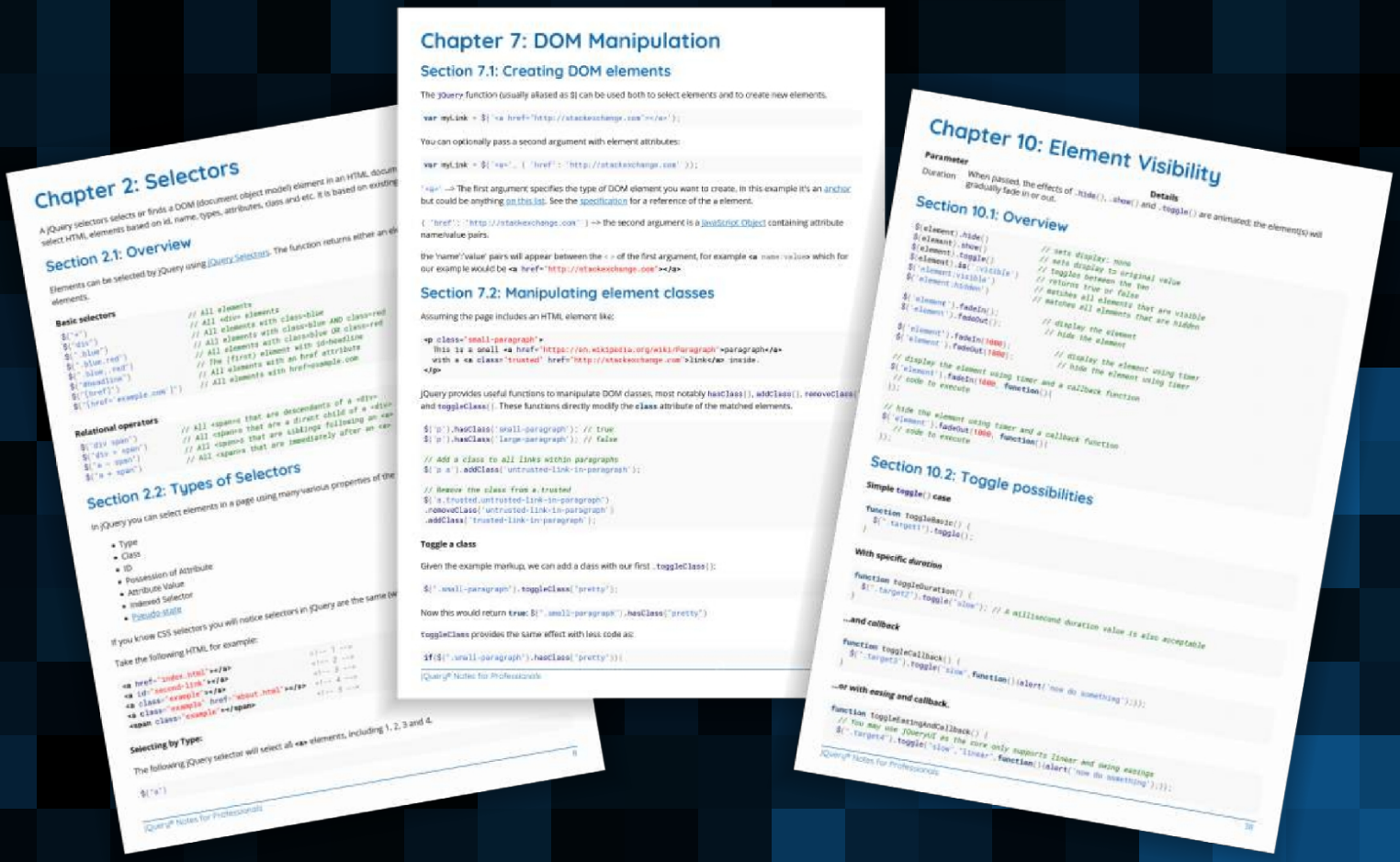
jQuery®

Notas para los profesionales



Más de 50 páginas

de consejos y trucos profesionales gratuitos

GoalKicker.com
Libros de programación

Contenido

Acerca de 1
Capítulo 1: Introducción a jQuery 2
Sección 1.1: Primeros pasos 2
Sección 1.2: Evitar colisiones entre espacios de nombres 3
Sección 1.3: Espacio de nombres jQuery ("jQuery" y "$") 4
Sección 1.4: Carga de jQuery por consola en una página que no lo tiene 5
Sección 1.5: Incluir la etiqueta script en la cabecera de la página HTML 5
Sección 1.6: El objeto jQuery 7
Capítulo 2: Selectores 8
Sección 2.1: Visión general 8
Sección 2.2: Tipos de selectores 8
Sección 2.3: Selectores de caché 10
Sección 2.4: Combinación de selectores 11
Sección 2.5: Elementos DOM como selectores 13
Sección 2.6: Cadenas HTML como selectores 13
Capítulo 3: Cada función 15
Sección 3.1: jQuery cada función 15
Capítulo 4: Atributos 16
Sección 4.1: Diferencia entre attr() y prop() 16
Sección 4.2: Obtener el valor de atributo de un elemento HTML 16
Apartado 4.3: Establecer el valor del atributo HTML 17
Sección 4.4: Eliminar atributo 17
Capítulo 5: evento document-ready 18
Apartado 5.1: ¿Qué es el document-ready y cómo debo utilizarlo? 18
Sección 5.2: jQuery 2.2.3 y versiones anteriores 18
Sección 5.3: jQuery 3.0 19
Sección 5.4: Adjuntar eventos y manipular el DOM dentro de ready() 19
Sección 5.5: Diferencia entre $(document).ready() y $(window).load() 20
Sección 5.6: Diferencia entre jQuery(fn) y ejecutar tu código antes de </body> 21
Capítulo 6: Eventos 22
Sección 6.1: Actos delegados 22
Sección 6.2: Manejadores de eventos Attach y Detach 23
Sección 6.3: Activación y desactivación de eventos específicos mediante jQuery. (Receptores con nombre) 24
Sección 6.4: originalEvent 25
Sección 6.5: Eventos de repetición de elementos sin utilizar IDs 25
Sección 6.6: Evento de carga del documento .load() 26
Capítulo 7: Manipulación del DOM 27
Sección 7.1: Creación de elementos DOM 27
Sección 7.2: Manipulación de clases de elementos 27
Sección 7.3: Otros métodos de la API 29
Capítulo 8: DOM Traversing 31
Sección 8.1: Seleccionar hijos de un elemento 31
Sección 8.2: Obtener el siguiente elemento 31
Sección 8.3: Obtener el elemento anterior 31
Sección 8.4: Filtrar una selección 32
Sección 8.5: método find() 33
Sección 8.6: Iterar sobre una lista de elementos jQuery 34
Sección 8.7: Selección de hermanos 34
Sección 8.8: Método closest() 34

Capítulo 9: Manipulación de CSS 36
Sección 9.1: CSS - Getters y Setters 36
Sección 9.2: Incremento/Decremento de propiedades numéricas 36
Sección 9.3: Establecer propiedad CSS 37
Sección 9.4: Obtener propiedad CSS 37
Capítulo 10: Visibilidad de los elementos 38
Sección 10.1: Visión general 38
Sección 10.2: Posibilidades de alternancia 38
Capítulo 11: Añadir 40
Sección 11.1: Uso consecutivo eficiente de .append() 40
Sección 11.2: jQuery append 43
Sección 11.3: Añadir un elemento a un contenedor 43
Capítulo 12: Prepender 45
Sección 12.1: Añadir un elemento a un contenedor 45
Apartado 12.2: Método Prepend 45
Capítulo 13: Obtener y definir la anchura y la altura de un elemento 47
Sección 13.1: Obtención y configuración de la anchura y la altura (sin tener en cuenta el borde) 47
Sección 13.2: Obtener y establecer innerWidth e innerHeight (ignorando padding y border) 47
Sección 13.3: Obtención y configuración de outerWidth y outerHeight (incluyendo relleno y borde) 47
Capítulo 14: Método jQuery .animate() 48
Sección 14.1: Animación con devolución de llamada 48
Capítulo 15: Objetos diferidos y promesas de jQuery 50
Sección 15.1: jQuery ajax() éxito, error VS .done(), .fail() 50
Sección 15.2: Creación de promesas básicas 50
Capítulo 16: Ajax 52
Sección 16.1: Manejo de códigos de respuesta HTTP con $.ajax() 52
Sección 16.2: Usar Ajax para enviar un formulario 53
Sección 16.3: Ejemplos todo en uno 53
Sección 16.4: Carga de ficheros Ajax 55
Capítulo 17: Casilla de verificación Seleccionar todo con marcación/desmarcación automática en otra casilla de verificación
cambiar 58
Sección 17.1: 2 seleccione todas las casillas de verificación con las casillas de verificación de grupo correspondientes 58
Capítulo 18: Plugins 59
Sección 18.1: Plugins - Primeros pasos 59
Créditos 61
También le puede gustar 64

Acerca de

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 web@petercv.com

Capítulo 1: Introducción a jQuery

Versión	Notas	Fecha de publicación
1.0	Primera versión estable	2006-08-26
1.1		2007-01-14
1.2		2007-09-10
1.3	Sizzle introducido en el núcleo	2009-01-14
1.4		2010-01-14
1.5	Gestión de devolución de llamada diferida, reescritura de módulo ajax	2011-01-31
1.6	Importantes mejoras de rendimiento en los métodos `attr()` y `val()`	2011-05-03
1.7	Nuevas API de eventos: `on()` y `off()`.	2011-11-03
1.8	Sizzle reescrito, animaciones mejoradas y`$ (html, props)` flexibilidad.	2012-08-09
1.9	Eliminación de interfaces obsoletas y limpieza de código	2013-01-15
1.10	Se han incorporado correcciones de errores y diferencias notificadas en los ciclos beta 1.9 y 2.0	2013-05-24
1.11		2014-01-24
1.12		2016-01-08
2.0	Se ha eliminado la compatibilidad con IE 6-8 para mejorar el rendimiento y reducir el tamaño.	2013-04-18
2.1		2014-01-24
2.2		2016-01-08
3.0	Aumento masivo de la velocidad de algunos selectores personalizados de jQuery	2016-06-09
3.1	Se acabaron los errores silenciosos	2016-07-07
3.2	Se acabaron los errores silenciosos	2017-03-16
3.3	Se acabaron los errores silenciosos	2018-01-19

Sección 1.1: Primeros pasos

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

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

Demostración en directo en JSBin

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

Explicación del código

1. Carga la librería jQuery desde el 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 librería ANTES que cualquier otro script o librería que pueda depender o hacer uso de ella.*

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

```
// Cuando el `documento` esté `listo`, 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.querySelectorAllMDN.
   2. Establece el `texto()` dentro del elemento seleccionado a `¡Hola, Mundo!`.

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

Para más información, consulte la página jQuery - Documentación.

Sección 1.2: Evitar colisiones entre espacios de nombres

Otras 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, puedes seguir utilizando la propia variable `jQuery` para acceder a las funciones de jQuery:

```
jQuery('#hello').text('¡Hola, mundo!');
```

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

```
var jqy= jQuery.noConflict();
```

```
jqy('#hello').text('¡Hola, mundo!');
```

Por el contrario, para evitar que otras bibliotecas 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 puede utilizar el alias $
   $('#hola').text('¡Hola, mundo!');
});
```

Resumiendo,

- `jQuery.noConflict()` :`$` ya no hace referencia a jQuery, mientras que la variable `jQuery` sí.
- `var jQuery2= jQuery.noConflict()` -`$` 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 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 sustituirse (excepto en los casos en que `jQuery.noConflict();` - ver 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 algún contenido de texto a este div. Para ello podríamos utilizar el jQuery `text()` función. Esto podría ser escrito usando `jQuery` o`$` . i.e. -

```
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 Demo!</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 ver 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.

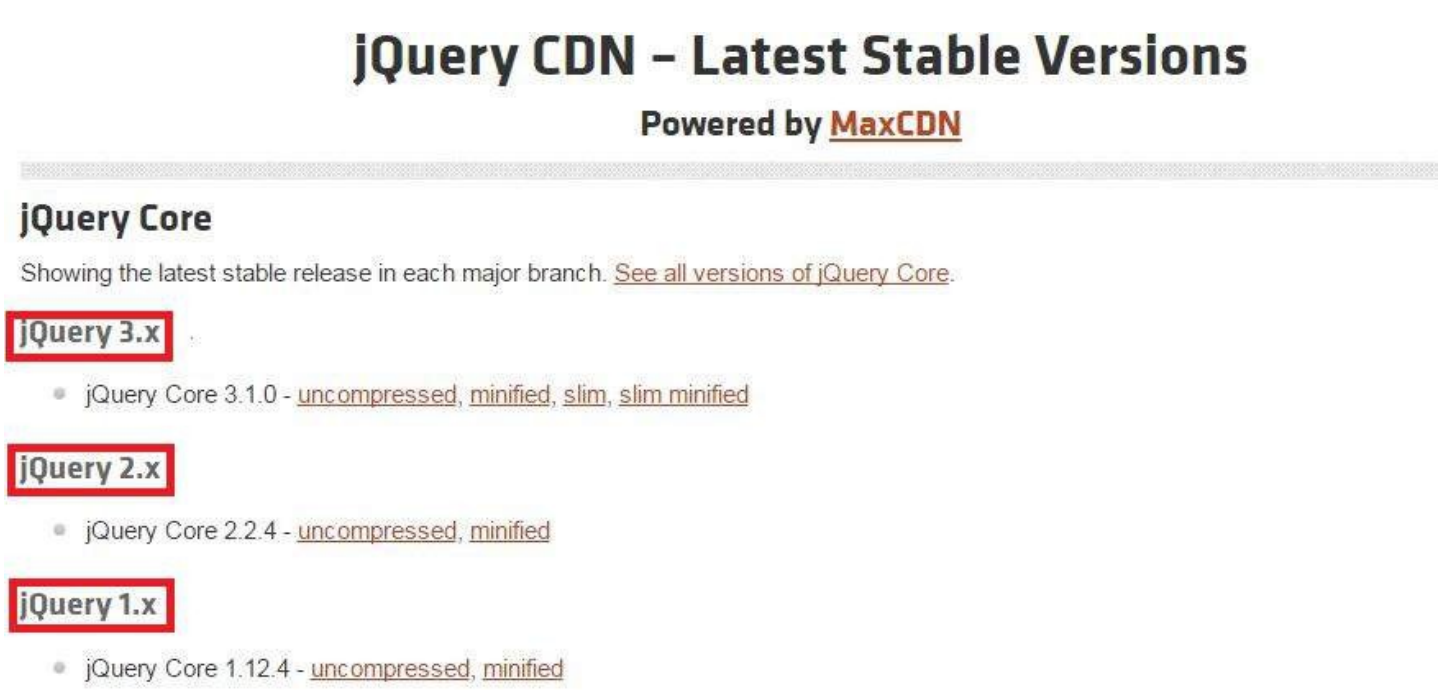
En tales situaciones se puede utilizar la consola `de Chrome Developer Tools` (F12) para añadir manualmente `jQuery` en una página cargada

```
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);
```

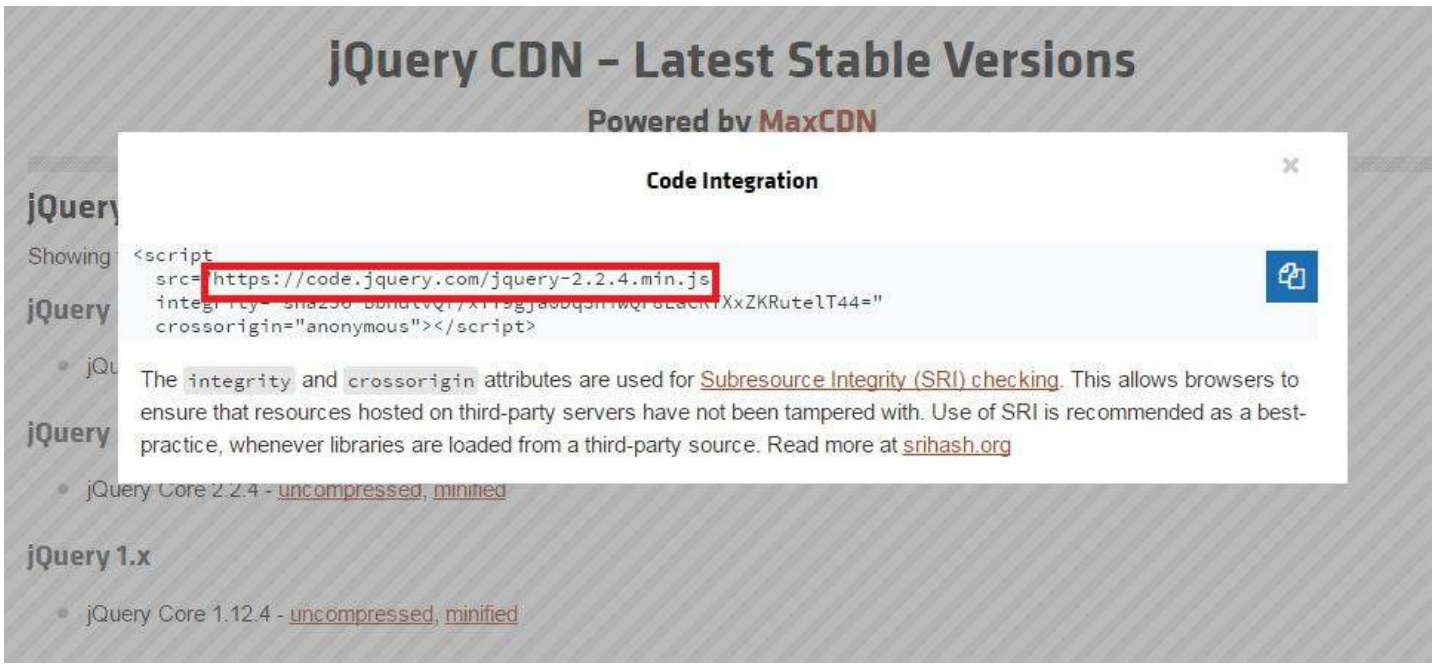
La versión que desea puede diferir de la anterior (`1.12.4`), puede obtener el enlace a la que necesita aquí.

Sección 1.5: Incluir la etiqueta script en la cabecera de la página HTML

Para cargar **jQuery** desde la CDN oficial, vaya al sitio web de jQuery. Verás 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 **sin comprimir** 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 es 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 está cargada con jquery.</p>
      </body>
   </html>
```

Cuando se utiliza el atributo `async` hay que tener en cuenta que las librerías javascript se cargan de forma asíncrona y se ejecutan tan pronto como están disponibles. Si se incluyen dos bibliotecas donde la segunda biblioteca depende de la primera, en este caso, si la segunda biblioteca 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 **`nueva`** 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 permite 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 parte de quien llama.

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 algo distinto de nothing, `undefined`, **`null`**, o similar). En el caso de un único elemento, esta estructura tipo array contendrá sólo ese elemento.

Una demostración sencilla 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");//poblar el objeto jQuery con el resultado del selector id
var div =$ div[0];//accede 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 basados en id, nombre, tipos, atributos, clase y etc. Se basa en los selectores CSS existentes.

Sección 2.1: Visión general

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

Selectores básicos

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

Operadores relacionales

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

Sección 2.2: Tipos de selectores

En jQuery puedes seleccionar elementos en una página usando 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 iguales (con pequeñas excepciones).

Tomemos como ejemplo el siguiente HTML:

```
<a href= "index.html"></a>                  <!-- 1 -->
<a id= "segundo-enlace"></a>                <!-- 2 -->
<a clase= "ejemplo"></a>                    <!-- 3 -->
<a class= "example" href= "about.html"></a>  <!-- 4 -->
<span class= "ejemplo"></span>              <!-- 5 -->
```

Seleccionar por tipo:

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

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

Seleccionar por clase

El siguiente selector jQuery seleccionará todos los elementos de la clase `ejemplo` (incluyendo 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, incluyendo 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>` , el `segundo enlace` porque el índice suministrado es `1` como `eq(1)` (Obsérvese que el índice empieza en `0`, de ahí que se haya seleccionado aquí el segundo).

```
$("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, utilice `:not()`

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

```
$("a:no(.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>`: el número 1.

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

Combinación de selectores jQuery

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

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

Esto seleccionaría un elemento `<a>` que:

- Tiene las siguientes clases: `class1, class2 y class3`
- Tiene el siguiente ID: `someID`
- 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 diferentes selectores con una coma:

```
$("a, .class1, #someID")
```

Esto seleccionaría:

- Todos los elementos `<a>`
- Todos los elementos que tienen la clase `class1`
- Un elemento con el id `#someID`

Selección de hijos y hermanos

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

- Para seleccionar un hijo no directo, utilice un espacio
- Para seleccionar un hijo directo, utilice un `>`
- Para seleccionar un hermano adyacente después del primero, utilice una tecla `+`
- Para seleccionar un hermano no adyacente a continuación del primero, utilice una tecla `~`

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

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

Esto reemplazaría:

```
$('#navegación').show();
```

El almacenamiento en caché de este selector podría 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 una matriz de estos elementos:

```
<div clase= "padre">
    <div clase= "niño"> Niño 1</div>
    <div clase= "niño"> Niño 2</div>
</div>

<script>
    var children= $('.child');
    var firstChildText= children[0].text();
    console.log(firstChildText);

    // salida: "Niño 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 estarás almacenando un array vacío en esa variable.

```
<div clase= "padre"></div>

<script>
    var padre     = $('.parent');
    var children = $('.child');
    console.log(children);

    // salida: []

    parent.append('<div class= "child"> Child 1</div>');
    children = $('.child'); console.log(children[0].text());

    // salida: "Niño 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 `$`para indicar que se trata de un objeto jQuery:

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

Sección 2.4: Combinación de selectores

Considere la siguiente estructura DOM

```
<ul class= "parentUl">
    <li> Nivel 1
        <ul class= "childUl">
            <li>Nivel 1-1 <span> Artículo - 1 </span></li>
            <li>Nivel 1-1 <span> Artículo - 2 </span></li>
        </ul>
    </li> <li>
```

```
    <li> Nivel 2
        <ul class= "childUl">
            <li>Nivel 2-1 <span> Artículo - 1 </span></li>
            <li>Nivel 2-1 <span> Artículo - 1 </span></li>
        </ul>
    </li> <li>
</ul>
```

Selectores descendientes e hijos

Dado un padre `<ul>` - `parentUl` encontrar sus descendientes `(<li>`),

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

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

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

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

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

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

3. Selector basado en el contexto -`$ ('child','parent')`

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

   Funciona igual que punto 1.

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

   ```
   >>$ ('ul.parentUl').find('li')
   ```

   Funciona igual que punto 1.

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

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

   Funciona igual que el 2. anterior.

Otros combinadores
Selector de grupo : ","

Seleccionar todos los elementos `<ul>` Y todos los elementos `<li>` Y todos los elementos `<span>` :

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

Selector de múltiplos : "" (sin carácter)

Selecciona todos los elementos `<ul>` con clase `parentUl` :

```
$('ul.parentUl')
```

Selector de hermanos adyacentes : "+"

Selecciona todos los elementos `<li>` que estén colocados inmediatamente después de otro elemento `<li>`:

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

Selector general de hermanos : "~"

Selecciona 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 DOM real. Pasar un elemento DOM a jQuery hará que la estructura subyacente tipo array 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 llamadas de retorno, donde el elemento actual se pasa al constructor de jQuery con el fin de obtener acceso a la API de jQuery.

Como en la llamada de retorno `each` (nota: each es una función iteradora).

```
$(".elements").each(function(){
    //el elemento actual está vinculado a `this` internamente por jQuery cuando se utiliza cada
    var currentElement= this;

    //en este , currentElement (o this) tiene acceso a la API Nativa

    //construye un objeto jQuery con el elemento currentElement(this)
    var$ currentElement =$ (this);

    //ahora $currentElement tiene acceso a la API jQuery
});
```

Sección 2.6: Cadenas HTML como selectores

jQuery acepta una amplia variedad de parámetros como "selectores", y uno de ellos es una cadena HTML. Pasar una cadena HTML a jQuery hará que la estructura subyacente tipo array del objeto jQuery contenga 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](regex101.com)).

El uso más común de una cadena HTML como selector es cuando conjuntos de elementos DOM necesitan ser creados sólo en código, a menudo esto es usado por librerías para cosas como popouts modales.

Por ejemplo, una función que devolviera una etiqueta anchor 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: Cada función

Sección 3.1: jQuery cada función

HTML:

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

Guión:

```
$( "li" ).each(function( index ) {
  console.log( index+ ": " +$ ( 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: Diferencia 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 ocasionalmente se necesita uno sobre el otro.

Para marcar una casilla de verificación:

```
$('#tosAccept').prop('checked', true); // usar 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 del atributo]);
```

Por
ejemplo:

HTML:

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

jQuery:

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

Obtención de atributos de `datos`:

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

Sintaxis:

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

Ejemplo:

Html:

```
<articulo datos-columna= "3"></articulo>
```

jQuery:

```
$("artículo").data("columna")
```

Nota:

> El método data() de jQuery le dará acceso a los atributos data-*, PERO, no tiene en cuenta las mayúsculas y minúsculas del nombre del atributo. Referencia

Apartado 4.3: Establecer el valor del atributo HTML

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

```
$('a').attr('title', 'Hazme clic');
```

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

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

Sección 4.4: Eliminación de atributos

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

```
$('#home').removeAttr('title');
```

Esto eliminará el atributo `title` del elemento con ID `home`.

Capítulo 5: evento "document-ready

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

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

```
<script type= "text/javascript">
  jQuery(function($) {
    // esto establecerá el texto del div a "Hola".
    $("#myDiv").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 haya sido renderizado en la página.

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

<div id= "miDiv"> Texto</div>
```

Ten en cuenta que puedes poner un alias al espacio de nombres de jQuery pasando un manejador personalizado al método `.ready()`. Esto es útil para casos en los que otra librería JS está usando el mismo alias acortado`$` que *jQuery*, lo que crea un conflicto. Para evitar este , debes llamar a`$ .noConflict();` - Esto te obliga a usar sólo el espacio de nombres por defecto de *jQuery* (En lugar del alias acortado`$`).
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 otros
   // bibliotecas que utilizan 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 al final 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.

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 la primera es la forma recomendada, la siguiente es una versión de que con el `jQuery` en lugar de`$` , que produce 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 confíe en que$ sea una referencia a jQuery en el espacio de nombres global
});
```

Todos los demás gestores de documentos listos están obsoletos en jQuery 3.0.

Asíncrono

A partir de jQuery 3.0, siempre será llamado de forma asíncronael manejador ready . Esto significa que en el código de abajo, el registro 'fuera manejador' siempre se mostrará en primer lugar, independientemente de si el documento estaba listo en el punto de ejecución.

```
$(function() {
  console.log("inside handler");
});
console.log("gestor externo");
```

> manipulador externo
> dentro del manipulador

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

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

1. **Adjuntar controladores de eventos**
   Adjuntar manejadores de eventos jQuery

```
$(document).ready(function() {
  $("botón").click(function() {
```

```
    // Código para 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();
  $("botón").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 jQuery 3.0)**, por lo que ya no debería utilizarse. Las razones de la eliminación se indican en la página de jQuery sobre este evento

> 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 esto que deben tenerse en cuenta. Estas son:
>
> - No funciona de forma coherente ni fiable en todos los navegadores.
> - No se dispara correctamente en WebKit si `el src` de la imagen se establece en el mismo `src` que antes• No burbujea correctamente 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:

`$(document).ready()` espera hasta que el DOM completo esté disponible -- todos los elementos del HTML han sido analizados y están en el documento. Sin embargo, es posible que recursos como las imágenes no se hayan cargado completamente en este punto. Si es esperar hasta que se hayan cargado todos los recursos,`$ (window).load()` **y es consciente de las importantes limitaciones de este**

puede utilizar lo siguiente:

```
$(document).ready(function() {
  console.log($ ("#my_large_image").height()); // puede ser 0 porque la imagen no está disponible
});

$(window).load(function() { console.log($
  ("#my_large_image").height()); // será correcto
});
```

Sección 5.6: Diferencia entre jQuery(fn) y ejecutar tu 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 mediante la ejecución de 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 al disparo del evento document ready como hace en:

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

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= "some_url/"> Enlace 1</a>
            </li> <li>
            <li>
                <a href= "some_url/"> Enlace 2</a>
            </li> <li>
            <li>
                <a href= "some_url/"> Enlace 3</a>
            </li> <li>
        </ul>
    </body>
</html>
```

El problema

Ahora, en este , queremos añadir un escuchador 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 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 sucesos

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

Así, en el ejemplo anterior, al hacer clic en el enlace del elemento `<a>` se desencadenará el evento 'clic' en

estos elementos en este orden:• a
• li• ul
•
cuerpo
• html
• raíz del documento

Solución

Sabiendo lo que hace el burbujeo de eventos, podemos capturar uno de los eventos deseados que se propagan hacia arriba a través de nuestro 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 de destino
                          // de esta forma `this` se refiere al ancla y no a la lista
  // Lo que quieras hacer cuando se pulse el enlace
});
```

Arriba:

- 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 receptor de eventos, ul).
- Por último, el tercer parámetro es el código que se ejecuta si se cumplen los requisitos de los parámetros primero y segundo.

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 elemento de la cadena es una "a", coincide con el filtro y también es un aprobado.
8. El código en el tercer parámetro se ejecuta utilizando el elemento coincidente como `este`. Si la función no incluye una llamada a `stopPropagation()`, el evento continuará propagándose hacia arriba, hacia la raíz (`documento`).

Nota: Si no hay disponible/conveniente un antepasado adecuado que no cambie, debe `utilizar documento`. Como hábito no utilice `'body'` por las siguientes razones:

- `body` tiene un error, relacionado con el estilo, que puede hacer que los eventos del ratón no lleguen a él. Esto depende del navegador y puede ocurrir cuando la altura calculada del cuerpo es 0 (por ejemplo, cuando todos los elementos hijos tienen posiciones absolutas). Los eventos del ratón siempre llegan al `documento`.
- *siempre* existe para tu script, por lo que puedes adjuntar manejadores delegados `al documento` fuera de un *manejador listo para DOM* y estar seguro de que seguirán funcionando.

Sección 6.2: Manejadores de eventos Attach y Detach

Adjuntar un controlador de eventos

Desde la versión 1.7 jQuery tiene la API de eventos `.on()`. De esta forma, cualquier evento javascript estándar o personalizado puede ser vinculado al elemento jQuery actualmente 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() ); //barra
```

```
});
```

Separar un manejador de eventos

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

HTML

```
<button id= "hola"> hola</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 ser eliminado.

jQuery

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

$('#hola').on('mouseenter', function(){
    console.log('estás 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. (Receptores con nombre)

A veces desea desconectar a todos los oyentes registrados previamente.

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

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

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

Para conseguirlo, podemos enlazar escuchas con nombre como,

```
//Agregar receptor de eventos con nombre.
```

```
$(document).on("click.mymodule",function(){
    console.log("Documento pulsado 1")
});
$(document).on("click.mymodule",function(){
    console.log("Documento clicado 2")
});

//Eliminar el receptor de eventos nombrado.
$(document).off("click.mymodule");
```

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 `Evento.originalEvento`

Obtener dirección de desplazamiento

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

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

Problema

Hay una serie de elementos que se repiten en la página y necesitas saber en cuál de ellos ocurrió un evento para hacer algo con esa instancia específica.

Solución

- Dar a todos los elementos comunes una clase común
- Aplica un receptor de eventos a una clase. `Este` controlador de eventos interno es el elemento selector coincidente en el que se produjo el evento.
- Recorrer hasta el contenedor más externo que se repite para esa instancia comenzando en `esto`
- Utilice `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= "person"> Fred</span></div>
   <div class= "item-toolbar">
      <button class= "eliminar"> Eliminar</button>
   </div>
</div>
<div class= "item-wrapper" data-item_id= "393">
   <div clss= "item"><span class= "person"> Wilma</span></div>
   <div class= "item-toolbar">
      <button class= "eliminar"> Eliminar</button>
   </div>
</div>
```

jQuery

```
$(function() {
  $('.delete').on('click', function() {
```

```
    // "this" es el elemento en el que ocurrió el evento
    var btn$=$  (this);
    // pasar al contenedor envoltorio
    var$ itemWrap =$ btn.closest('.item-wrapper');
    // buscar dentro de la envoltura para obtener la persona para esta instancia de botón
    var persona =$ itemWrap.find('.persona').text();
    // enviar borrado 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 para abreviar `.on( "load", handler)`.

HTML

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

jQuery

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

Capítulo 7: Manipulación del DOM

Sección 7.1: Creación de elementos DOM

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

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

Opcionalmente puede pasar un segundo argumento con atributos de elementos:

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

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

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

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

Apartado 7.2: Manipulación de clases de elementos

Suponiendo que la página incluye un elemento HTML como:

```
<p class= "pequeño-párrafo">
  Se trata de un pequeño <a href= "https://en.wikipedia.org/wiki/Paragraph"> párrafo</a>
  con un <a class= "trusted" href= "http://stackexchange.com"> enlace</a> en su interior.
</p>
```

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

```
$('p').hasClass('pequeño-párrafo'); // true
$('p').hasClass('párrafo-grande'); // false

// Añadir una clase a todos los enlaces dentro de los párrafos
$('p a').addClass('untrusted-link-in-paragraph');

// Eliminar la clase de a.trusted
$('a.trusted.untrusted-link-in-paragraph')
.removeClass('untrusted-link-in-paragraph')
.addClass('trusted-link-in-paragraph');
```

Alternar una clase

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

```
$(".pequeño-párrafo").toggleClass("bonito");
```

Ahora esto devolvería `true`:`$ (".small-paragraph").hasClass("pretty")`

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

```
if($ (".small-paragraph").hasClass("pretty")){
```

```
   $(".small-paragraph").removeClass("pretty");}
si no {
   $(".pequeño-párrafo").addClass("bonito"); }
```

alternar Dos clases:

```
$(".pequeño-párrafo").toggleClass("bastante guay");
```

Booleano para añadir/eliminar clases:

```
$(".small-paragraph").toggleClass("pretty",true); //no puede ser        truthy/falsey

$(".pequeño-párrafo").toggleClass("bonito",false);
```

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

```
$( "div.surface" ).toggleClass(function() {
  if ($ ( this ).parent().is( ".water" ) ) {
    devuelve "mojado";
  } else {
    devuelve "seco";
  }
});
```

Utilizado en ejemplos:

```
// funciones a utilizar en los ejemplos
  function cadenaContiene(miCadena, miSubCadena) {
    return miCadena.indexOf(miCadenaSub) !== -1;
}
function isOdd(num) { return num % 2;}
var showClass= true; //queremos añadir la clase
```

Ejemplos:

Utilice el índice del elemento para alternar clases pares/impares

```
$( "div.gridrow" ).toggleClass(function(index,oldClasses, false), showClass ) { showClass
  if ( isOdd(index) ) {
    devuelve "mojado";
  } else {
    devuelve "seco";
  }
});
```

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

```
<div clase= "cuadrícula">
  <div clase= "gridrow"> fila</div>
  <div clase= "gridrow"> fila</div>
  <div clase= "gridrow"> fila</div>
  <div clase= "gridrow"> fila</div>
  <div clase= "gridrow"> fila</div>
  <div class= "gridrow gridfooter"> row but I am footer!</div>
</div>
```

Funciones sencillas para nuestros ejemplos:

```
function isOdd(num) {
  devolver num % 2;
}

 function cadenaContiene(miCadena, miSubCadena) {
   return miCadena.indexOf(miSubCadena) !== -1;
}
  var showClass= true; //queremos añadir la clase
```

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

```
$("div.gridrow").toggleClass(function(index, oldClasses, showThisClass) {
  if (isOdd(index)) {
    devuelve "impar";
  } else {
    devolver "incluso";
  }
  return clasesantiguas;
}, showClass);
```

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

```
$("div.gridrow").toggleClass(function(index, oldClasses, showThisClass) {
  var isFooter= stringContains(oldClasses, "gridfooter");
  if (isFooter) {
    oldClasses= oldClasses.replace('even', ' ').replace('odd', ' ');
    $(this).toggleClass("par impar", false);
  }
  return clasesantiguas;
}, showClass);
```

Las clases que se devuelven son las que se ven afectadas. Aquí, si un elemento no tiene un `gridfooter`, añade una clase para even/odd. Este ejemplo ilustra el retorno de la lista de clases OLD. Si este `else return oldClasses;` se elimina, sólo se añaden las nuevas clases, por lo que la fila con una clase gridfooter tendría todas las clases eliminadas si no hubiéramos devuelto las antiguas - de lo contrario se habrían activado (eliminado).

```
$("div.gridrow").toggleClass(function(index, oldClasses, showThisClass) {
  var isFooter= stringContains(oldClasses, "gridfooter");
  if (!isFooter) {
    if (isOdd(index)) {
      devolver "oddLight";
    } else {
      return "evenLight";
    }
  } else return oldClasses;
}, showClass);
```

Sección 7.3: Otros métodos de la API

jQuery ofrece una variedad de métodos que pueden ser utilizados para la

manipulación del DOM. El primero es el método .empty().

Imagine el siguiente marcado:

```
<div id= "contenido">
  <div>Algo de texto</div>
```

```
</div>
```

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

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

```
<tr id= "fila_1">
  <td><button type= "button" class= "delete"> Delete</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:

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

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

Si quisieras obtener el `id` de la fila que contiene el botón `Borrar` sobre el que has hecho clic, podrías hacer algo como esto

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

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

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

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

y también hay una función .parent():

```
$('.delete').click(function() {
  var$ row =$ (this).parent().parent();
  var id =$ row.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: DOM Traversing

Sección 8.1: Seleccionar hijos de un elemento

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

```
<div clase= "padre">
    <h2>Un titular</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", "verde");
```

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

```
// Sólo obtener hijos "p
$('.padre').children("p").css("color", "verde");
```

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 "Ana" y quieres obtener el siguiente elemento, "Pablo", el método `.next()` te permitirá hacerlo.

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

El método toma un argumento `selector` opcional, que se puede utilizar 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", "verde");
```

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

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

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.

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

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", "verde");
```

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", "verde");
```

Sección 8.4: Filtrar una selección

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

El método se ejecuta sobre una selección y devuelve una nueva selección. Si el filtro coincide con un elemento, éste se añade a la selección devuelta; en caso 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"); // Color even elements green
$("li").filter(".one").css("font-weight", "bold"); // Poner ".one" en negrita
```

Función

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

La función se ejecuta para cada elemento de la selección. Si devuelve un `verdadero`, el elemento se añadirá a la selección devuelta.

```
var selection =$ ("li").filter(function (index, element) {
    // "índice" es la posición del elemento
    // "elemento" es lo mismo que "esto"
    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 three= document.getElementsByClassName("three");
$("li").filter(tres).css("color", "verde");
```

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", "verde");
```

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 clase= "padre">
       <div clase= "niños" nombre= "primero">
           <ul>
               <li>A1</li>
               <li>A2</li>
               <li>A3</li>
           </ul>
       </div>
       <div clase= "niños" nombre= "segundo">
           <ul>
               <li>B1</li>
               <li>B2</li>
               <li>B3</li>
           </ul>
       </div>
      </div>
```

jQuery

```
$('.parent').find('.children[name="second"] 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.

Considere esta estructura DOM:

```
<div clase= "contenedor">
    <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 actualmente, 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 sobre el objeto, como `css()` u `hide()` etc. En este ejemplo, sólo sacamos 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= "selected"> Inicio</li>
    <li>Blog</li>
    <li>Acerca de</li>
</ul>
```

Cuando el usuario hace clic en un elemento del menú, la clase `seleccionada` 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("selected");
});
```

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("selected");
```

Sección 8.8: Método closest()

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

HTML

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

jQuery

```
var target= $ ('#origin').closest('.row');
console.log("Fila más cercana:", target.attr('id')
);

var target2= $ ('#origin').closest('p');
console.log("P más cercano:", target2.attr('id') );
```

Salida

```
"Fila más cercana: abc"
"P más cercana: origen"
```

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

HTML

```
<div clase= '.primerEjemplo'>
   <p>Este es el primer párrafo en un div.</p>
   <p>Este es el segundo párrafo en un div.</p>
   <p>Este es el tercer párrafo en un div.</p>
   <p>Este es el cuarto párrafo en un div.</p>
   <p>Este es el quinto párrafo en un div.</p>
</div>
```

JQuery

```
var firstParagraph= $ ("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

Obtención de 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. ej: `150px`.
// Fíjate en la designación `as a string` - si necesitas un entero verdadero,
// hacer referencia al método `$.width()`.
$("body").css("anchura");
```

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

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

```
$("body").css(["animación", "anchura"]);
```

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

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

Ajustador CSS

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

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

Esta sentencia establece el `ancho` del`$ ("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: Incremento/Decremento de propiedades numéricas

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

```
// Incremento utilizando la sintaxis +=
$("#elemento-objetivo").css("font-size", "+=10");

// También puede especificar la unidad de incremento
$("#elemento-objetivo").css("anchura", "+=100pt");
$("#elemento-objetivo").css("top", "+=30px");
$("#elemento-objetivo").css("izquierda", "+=3em");
```

```
// La disminución se realiza utilizando la sintaxis -=.
$("#elemento-objetivo").css("altura", "-=50pt");
```

Sección 9.3: Establecer propiedad CSS

Fijar un solo estilo:

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

Configurar varios estilos al mismo :

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

Sección 9.4: Obtener propiedad CSS

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

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

Capítulo 10: Visibilidad de los elementos

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

```
$(elemento).ocultar()          // establece display: none
$(elemento).show()             // restablece el valor original de la pantalla
$(elemento).toggle()           // alterna entre los dos
$(elemento).is(':visible')     // devuelve verdadero o falso
$('element:visible')           // coincide con todos los elementos visibles
$('element:hidden')            // coincide con todos los elementos ocultos

$('elemento').fadeIn();        // mostrar el elemento
$('elemento').fadeOut();       // ocultar el elemento

$('elemento').fadeIn(1000);    // mostrar el elemento con temporizador
$('elemento').fadeOut(1000);   // ocultar el elemento mediante temporizador

// mostrar el elemento usando un temporizador y una función callback
$('elemento').fadeIn(1000, function(){
 // código a ejecutar
});

// ocultar el elemento usando un temporizador y una función callback
$('elemento').fadeOut(1000, function(){
   // código a ejecutar
});
```

Sección 10.2: Posibilidades de alternancia

Caso simple de `toggle()`

```
function toggleBasic() {
  $(".target1").toggle();
}
```

Con *duración específica*

```
function toggleDuration() {
  $(".target2").toggle("slow"); // También es aceptable un valor de duración en milisegundos
}
```

...y *devolución de llamada*

```
function toggleCallback() {
  $(".target3").toggle("lento",function(){alert('ahora haz algo');});
}
```

...o con *flexibilización* y rellamada.

```
function toggleEasingAndCallback() {
  // Usted puede utilizar jQueryUI como el núcleo sólo es compatible con lineal y swing easings
  $(".target4").toggle("lento", "lineal",function(){alert('ahora haz algo');});
```

```
}
```

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

```
function toggleWithOptions() {
  $(".target5").toggle(
    { // Ver todas las opciones posibles en: api.jquery.com/toggle/#toggle-options
      duration:1000, // milisegundos
      easing: "lineal",
      done:function(){
        alert('ahora haz algo');
      }
    }
  );
}
```

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

```
function toggleSlide() {
  $(".target6").slideToggle(); // Se anima de arriba a abajo, en lugar de la esquina superior
}
```

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

```
function toggleFading() {
  $( ".target7" ).fadeToggle("lento")
}
```

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

```
function toggleClass() {
  $(".target8").toggleClass('active');
}
```

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

```
function toggleX() {
  $(".targetX").toggle("lento");
}
```

Todos los ejemplos anteriores pueden consultarse aquí

Capítulo 11: Añadir

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

Sección 11.1: Uso consecutivo eficiente de .append()

:

HTML

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

JS

```
var datos= [
    { tipo: "Nombre", contenido: "John Doe" },
    { tipo: "Fecha de nacimiento", contenido: "01/01/1970" },
    { tipo: "Salario", contenido: "$40,000,000" },
    // ...300 filas más...
    { tipo: "Sabor Favorito", contenido: "Sour" }
];
```

Añadir dentro de un bucle

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

página. Su primer pensamiento puede ser hacer algo como esto:

```
var i;                          // <- el número de artículo actual
var count= data.length;         // <- el total
var fila;                       // <- para mantener una referencia a nuestro
                                objeto fila

// Bucle sobre el array
for ( i= 0; i< count;++ i ) {
    row=  data[ i ];

    // Pon 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 una matriz separada, añadir después de completar el bucle**

```
/**
 * Recorrido repetido del DOM (siguiendo el árbol de elementos hacia abajo hasta llegar a
 * como nuestra <table>) también debe evitarse siempre que sea posible.
 */

// Guarda la tabla en caché en una variable y luego úsala hasta que creas que ha sido eliminada
var$ miTabla =$ ('#mi-tabla');

// Para contener nuestros nuevos objetos <tr> jQuery.
var rowElements= [];

var count= data.length;
var i;
var fila;

// Bucle sobre el array
for ( i= 0; i< count;++ i ) {
    rowElements.push(
        $('<tr></tr>').append(
            $('<td></td>').html(fila.tipo),
            $('<td></td>').html(fila.contenido)
        )
    );
}

// Por último, inserte TODAS las filas a la vez
$myTable.append(rowElements);
```

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 rowElements= data.map(function ( row ) {

    // Crear una fila
    var$ row =$ ('<tr></tr>');

    // Crear las columnas
    var$ type =$ ('<td></td>').html(row.type);
    var$ content =$ ('<td></td>').html(row.content);

    // Añadir las columnas a la fila
    $row.append($ type,$ content);

    // Añadir a la matriz recién generada
    devolver$ fila;
});

// Finalmente, pon TODAS las filas en tu tabla
$myTable.append(rowElements);
```

Funcionalmente equivalente anterior, sólo que más fácil de leer.

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

```
// ...
var rowElements= data.map(function ( row ) {
    var rowHTML= '<tr><td>';
    rowHTML+=  row.type;
    rowHTML+=  '</td><td>';
    rowHTML+=  row.content;
    rowHTML+=  '</td></tr>';
    return rowHTML;
});

// Usando .join('') aquí se combinan todas las cadenas separadas en una sola
$myTable.append(rowElements.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 $myTable =$ (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 colFragment= 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
 *     que contiene todos los elementos <tr>.
 */
var rowFragment= data.reduce(function ( fragment, row ) {

    // Crear una fila
    var rowEl= document.createElement('tr');

    // Crear las columnas y los nodos de texto interiores
    var typeEl= document.createElement('td');
    var typeText= document.createTextNode(row.type);
    typeEl.appendChild(typeText);

    var contentEl= document.createElement('td');
    var contentText= document.createTextNode(row.content);
    contentEl.appendChild(contentText);

    // Añadir las columnas al fragmento de columna
    // - esto sería útil si las columnas se iteraran por separado
    //    colFragment.appendChild(typeEl);
    colFragment.appendChild(contentEl);
```

```
    rowEl.appendChild(colFragment);

    // Añadir rowEl al fragmento - esto actúa como un buffer temporal para
    // acumular varios nodos DOM antes de la inserción masiva
    fragment.appendChild(filaEl);

    devolver fragmento;
}, document.createDocumentFragment());

// Ahora vuelca todo el fragmento en tu tabla
$myTable.append(filaFragmento);
```

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

Profundizar

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

Sección 11.2: jQuery append

HTML

```
<p>Este es un bonito </p>
<p>Me gusta </p>

<ul>
  <li>Punto 1 de la lista</li>
  <li>Lista 2</li>
  <li>Lista 3</li>
</ul>

<button id= "btn-1"> Añadir texto</button>
<button id= "btn-2"> Añadir elemento a la lista</button>
```

Guión

```
$("#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:

```
$('#parent').append($ ('#child'));
```

Solución 2:

```
$('#child').appendTo($ ('#parent'));
```

Ambas soluciones anexan el elemento `#child` (añadiendo al final) al elemento `#parent`. Antes:

```
<div id= "padre">
  <span>otros contenidos</span>
</div>
<div id= "niño">

</div>
```

Después:

```
<div id= "padre">
  <span>otros contenidos</span>
  <div id= "niño">

  </div>
</div>
```

Nota: Cuando añada contenido que ya existe en el documento, este contenido será eliminado de su contenedor padre original y añadido al nuevo contenedor padre. Así que no puedes usar `.append()` o `.appendTo()` para clonar un elemento. Si necesita un clon use `.clone()` -> [http://api.jquery.com/clone/][1]

Capítulo 12: Prepender

Sección 12.1: Añadir un elemento a un contenedor

Solución 1:

```
$('#parent').prepend($ ('#child'));
```

Solución 2:

```
$('#child').prependTo($ ('#parent'));
```

Ambas soluciones anteponen el elemento `#child` (añadiendo al principio) al elemento `#parent`. Antes:

```
<div id= "padre">
  <span>otros contenidos</span>
</div>
<div id= "niño">

</div>
```

Después:

```
<div id= "padre">
  <div id= "niño">

  </div>
  <span>otros contenidos</span>
</div>
```

Apartado 12.2: Método Prepend

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

1. `prepend( contenido [, contenido ] )`

```
// con cadena html
jQuery('#parent').prepend('<span>child</span>');
// o puedes usar el objeto jQuery
jQuery('#parent').prepend($ ('#child'));
// o puede utilizar elementos múltiples separados por comas para anteponer
jQuery('#parent').prepend($ ('#child1'),$ ('#child2'));
```

2. `prepend(function)`

Versión de JQuery*: 1.4* en adelante puedes usar la función callback 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('#parent').prepend(function(i,oldHTML){
      // devuelve el valor a añadir
      return '<span>hijo</span>';
```

```
});
```

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

Sección 13.1: Obtención y configuración de la anchura y la altura (sin tener en cuenta el borde)

Obtener anchura y altura:

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

Establece la anchura y la altura:

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

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

Obtener anchura y altura:

```
var width =$ ('#elemento-objetivo').innerWidth();
var height =$ ('#elemento-objetivo').innerHeight();
```

Establece la anchura y la altura:

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

Sección 13.3: Obtención y configuración de outerWidth y outerHeight (incluyendo relleno y borde).

Obtener anchura y altura (sin margen):

```
var width =$ ('#elemento-objetivo').outerWidth();
var height =$ ('#elemento-objetivo').outerHeight();
```

Obtener anchura y altura (incluido el margen):

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

Establece la anchura y la altura:

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

Capítulo 14: Método jQuery .animate()

Parámetro	Detalles
propiedades	Un objeto de propiedades y valores CSS hacia los que se moverá la animación duración
	(por defecto: 400) Una cadena o número que determina cuánto durará la animación easing
	(por defecto: swing) Una cadena que indica la función easing a utilizar para la transición
complete	Una función a llamar una vez que la animación se ha , llamada una vez por cada elemento
emparejado. start	especifica una función que se ejecutará cuando comience la animación.
paso	especifica una función que se ejecutará en cada paso de la animación.
cola	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 .
terminado	especifica una función que se ejecutará cuando finalice la animación.
fallo	especifica una función que se ejecutará si la animación no se completa.
specialEasing	un mapa de una o más propiedades CSS del parámetro styles, y su correspondiente easing siempre funciones.
	especifica una función a ejecutar si la animación se detiene sin completarse.

Sección 14.1: Animación con devolución de llamada

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 solucionar este problema proporcionando un método increíble y flexible llamado `.animate()`. Este método permite establecer animaciones personalizadas que se utilizan propiedades css que dan permiso para volar por encima de las fronteras. por ejemplo, si damos la propiedad de estilo css como `width:200;` y la posición actual del elemento DOM es 50, el método animate reduce el valor de la posición actual del valor css dado y anima ese elemento a 150. Pero no tenemos que 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(){
        $("#box").animate({ancho: "200px"});
    });
</script>

<button id= "btn1"> Animar Ancho</button>
<div id= "box" 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()`.

```
milisegundos (Ej: 100, 1000, 5000, etc.),
"lento",
```

```
"rápido"
```

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

"swing"
"lineal"

He aquí algunos ejemplos con opciones de animación

complejas. Ej. 1:

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

Ej. 2:

```
  $("#box").animate({
     altura:  "300px",
     anchura: "300px"
     }, {
     duración: 5000, easing:
     "linear", complete:
     function(){
        $(this).after("<p>¡La animación ha terminado!</p>");
     }
  });
```

Capítulo 15: Objetos diferidos y promesas de jQuery

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

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

é**xito y Error :** Un callback de é**xito** que se invoca cuando se completa con éxito una petición Ajax. Una llamada de

retorno **de error** que se invoca en caso de que se produzca algún error al realizar la solicitud.

Por ejemplo:

```
$.ajax({
        url: 'URL',
        tipo: 'POST',
        datos: tusDatos,
        tipo de datos:
        'json',
        éxito: function (datos) { éxitoFunción(datos); },
        error: function (jqXHR, textStatus, errorThrown) { errorFunction(); }
    });
```

.done() y .fail() :

.ajax().done(function(data, textStatus, jqXHR){}); Sustituye al método .success() que fue obsoleto en jQuery 1.8.Esta es una construcción alternativa para la función de devolución de llamada de éxito anterior.

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

Por ejemplo:

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

Sección 15.2: Creación de promesas básicas

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

```
function esperarPromesa(milisegundos){

   // Crear un nuevo objeto Deferred usando el método estático de jQuery
   var def =$ .Deferred();

   // Hacer algún trabajo asíncrono - en este caso un simple temporizador
   setTimeout(function(){
```

```
        // Trabajo completado... resuelve el diferido, para que su promesa proceda
        def.resolve();
    }, milisegundos);

    // Devuelve inmediatamente una "promesa de proceder cuando termine el tiempo de
    espera"
    return def.promise();
}
```

Y úsalo así:

```
waitPromise(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
configuración	un objeto que contiene numerosos valores que afectan al comportamiento del tipo de
solicitud	El método HTTP que se utilizará para la solicitud
datos	Datos que debe enviar la solicitud
éxito	Una función de devolución de llamada que se llamará si la
solicitud tiene éxito error	Una llamada de retorno 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".
contexto	Especifica el contexto que se utilizará dentro de las retrollamadas, normalmente `este` que se refiere al objetivo actual.

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

Además de las devoluciones de llamada 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({
    tipo: {POST o GET o PUT etc.},
    url: {servidor.url},
    datos: {algunosDatos: 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(datos){
    alert(datos);
})
.fail(function(jqXHR, textStatus){ alert('Algo ha
    ido 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 la devolución de llamada de éxito; si resulta en un error (incluida la redirección 3xx), toman los mismos parámetros que la devolución de llamada de `error`.

Sección 16.2: Usar Ajax para enviar un formulario

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

que tiene este simple formulario -

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

  $.ajax({
      tipo: 'POST',
      url:$ form.attr('action'),
      datos:$ form.serialize(),
      éxito: function(datos) {
        // 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 y usarlo para la propiedad "url".
- `data:$ form.serialize(),` - Convierte las entradas 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){
    $('#update-box').html(datos);
});
```

Solución 2:

```
$.ajax({
     tipo: 'GET',
     url: 'url.php',
}).done(function(data){
     $('#update-box').html(datos);
}).fail(function(jqXHR, textStatus){ alert('Se ha
     producido un error: '+ textStatus);
});
```

Carga Ajax: Otro método ajax get creado para simplcity

```
$('#update-box').load('url.html');
```

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

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

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

```
$('#update-box').load('url.php', {data: "algo"}, function(resolver){
    //hacer algo
});
```

Ajax Post:

Solución 1:

```
$.post('url.php',
    {date1Name: data1Value, date2Name: data2Value},    //datos a contabilizar
    function(datos){
        $('#update-box').html(datos);
    }
);
```

Solución 2:

```
$.ajax({
    tipo: 'Post',
    url: 'url.php',
    datos: {date1Name: data1Value, date2Name: data2Value}    //datos a contabilizar
}).done(function(data){
    $('#update-box').html(datos);
}).fail(function(jqXHR, textStatus){ alert('Se ha
    producido un error: '+ textStatus);
});
```

Ajax Post JSON:

```
var postData = {
    Nombre: nombre,
    Dirección:
    dirección,
    Teléfono: teléfono
};

$.ajax({
     tipo: "POST",
     url: "url.php",
```

```
    dataType: "json",
    datos: JSON.stringfy(postData), éxito:
    function (datos) {
        /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({
     tipo: "Get",
     url: "url.php",
     dataType: "json",
     datos: JSON.stringfy(postData),
     éxito: function (datos) {
         /aquí los datos variables están en formato JSON
     },
     error: function(jqXHR, textStatus){ alert('Se
         ha producido un error: '+ textStatus);
     }
 });
```

Sección 16.4: Carga 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 una nueva selección de archivos.

```
<tipo de entrada= "archivo" id= "archivo-entrada" múltiple>
```

```
archivos var;
var fdata= new FormData();
$("#archivo-entrada").on("cambio", function (e) {
    archivos=  this.archivos;

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

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

    $.ajax({
        url: "/Test/Url",
        type: "post",
        data: fdata, //añade el objeto FormData al parámetro data
        processData: false, //indicar a jquery que no procese los datos
```

```
        contentType: false, //indicar a jquery que no establezca content-type
        éxito: function (respuesta, estado, jqxhr) {
            /éxito del mango
        },
        error: function (jqxhr, status, errorMessage) {
            /emitir error
        }
    });
});
```

Ahora a desglosar esto e inspeccionarlo parte por parte.

2. Trabajar con entradas de archivos

Este documento MDN (Using files from web applications) 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 una `entrada de archivo`. La propiedad `multiple` permite seleccionar más de un archivo, puedes quitarla si quieres que el usuario seleccione un archivo a la vez.

```
<tipo de entrada= "archivo" id= "archivo-entrada" múltiple>
```

Utilizaremos `el evento de cambio` de entrada para capturar los archivos.

```
archivos var;
$("#archivo-entrada").on("cambio",
    function(e){archivos=  this.archivos;
});
```

Dentro de la función handler, accedemos a los ficheros a través de la propiedad files de nuestra entrada. Esto nos da un FileList, que es una matriz 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(ficheros, function(i, fichero) {
  //...
});
```

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("Sexo", "Masculino");
fdata.append("Edad", "24");
//...
```

4. Envío de archivos con Ajax

```
$.ajax({
    url: "/Test/Url",
    type: "post",
    data: fdata, //añade el objeto FormData al parámetro data
    processData: false, //indica a jquery que no procese los datos
    contentType: false, //indica a jquery que no establezca el tipo de
    contenido success: function (response, status, jqxhr) {
        /éxito del mango
    },
    error: function (jqxhr, status, errorMessage) {
        /emitir 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 marcación/desmarcación automática al cambiar otra casilla de verificación

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 cualquiera de las casillas de verificación del grupo.
Restricción: El id "select all" debe coincidir con los nombres de entrada para crear el grupo select all. En el ejemplo, el ID de la entrada select all 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: 2 seleccione 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"> Seleccionar todo
<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"> Seleccionar todo
<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">
    $("entrada").change(function() {
      $('input[name='+this.id+'\']').not(this).prop('checked', this.checked);
      $('#'+this.name).prop('checked', $('input[name=''+this.name+'\']').===
$('input[name=''+this.name+'\']').filter(':checked').length);
    });
</script>
```

Capítulo 18: Plugins

Sección 18.1: Plugins - Primeros pasos

La API de jQuery puede ampliarse añadiendo funciones 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 haya pasado. Por ejemplo, `$('.active')` construirá un objeto jQuery que contiene elementos con la clase active, en el momento de la llamada (es decir, no se trata de un conjunto vivo de elementos).

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({ fondo: "amarillo" });
};

// Ejemplo de uso:
$("span").highlight();
```

jsFiddle ejemplo

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 por ejemplo `$(".warn").append("¡ATENCIÓN! ").css({color: "rojo"})` (véase cómo hemos utilizado el método `.css()` después del `.append()`, ambos métodos se aplican en 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 ) {

    // Configuración por defecto
    var settings =$ .extend({
        color : "",               // Por defecto el color del texto
        actual
        fondo : "amarillo"        // Por defecto fondo amarillo
    }, personalizado);

    return this.css({       // `return this` mantiene la encadenabilidad del
        color : settings.color;  método
```

```
        backgroundColor : settings.background
    });

  };
}( jQuery ));


// Utilizar la configuración por defecto
$("span").highlight();   // puedes encadenar otros métodos

// Utilizar ajustes personalizados
$("span").highlight({ fondo:
    "#f00", color: "blanco"
});
```

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:

```
//...
// Configuración por
defecto
var settings =$ .extend({// texto a resaltar
   text : ""
   class : "highlight" // referencia a la clase CSS
}, personalizado);

return this.each(function() {
   // su 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({
    texto : "hola",
    clase : "makeYellowBig"
});
```

jsFiddle ejemplo

Créditos

Muchas gracias a todas las personas de Stack Overflow Documentation que ayudaron a proporcionar este contenido, se pueden enviar más cambios aweb@petercv.com para que se publiquen o actualicen nuevos contenidos.

A.J	Capítulos 1, 4 y 8
acdcjunior	Capítulos 1 y 4
Alex	Capítulo 15
Alex Char	Capítulo 10
Alon Eitan	Capitulo 5
amflare	Capítulos 1 y 5
Andrew Brooke	Capitulo 16
Anil	Capítulo 1
Arun Prasad E S	Capítulo 16
Ashiquzzaman	Capítulo 15
Ashkan Mobayen Khiabani	Capítulos 9, 11, 12, 13 y 16
Asimilador	Capítulo 7
Athafoud	Capítulo 16
prohibición17	Capítulo 4
Ben H	Capítulo 16
Bipon	Capítulos 3 y 11
Brandt Solovij	Capítulo 9
Brock Davis	Capitulo 7
bwegs	Capítulo 1
Castro Roy	Capítulo 2
charlietfl	Capítulo 6
csbarnes	Capítulo 16
Darshak	Capítulo 11
David	Capítulo 2
DefyGravity	Capitulo 7
EncantadoD0D	Capítulo 2
Deryck	Capítulos 7 y 11
devlin carnate	Capítulo 2
dlsso	Capítulo 8
Dr. J. Testington	Capítulo 16
Emanuel Vintilă	Capitulo 5
empírico	Capítulos 11 y 12
Flyer53	Capítulo XI
El café como combustible	Capítulo I
Codificación	Capítulos 6 y 15
hasan	Capítulos 2 y 18
Horst Jahns	Capítulo 6
Hombre de Hielo	Capitulo 2
Igor Raush	Capitulo 1
J F	Capítulo 5
j08691	Capítulo 9
Jatniel Prinsloo	Capítulo 6
jkdev	Capítulos 1 y 5
JLF	Capítulo 2
Juan 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	Capitulo 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	Capitulo 2
Mottie	Capitulo 1
Neal	Capitulo 1
Nhan	Capítulo 5
ni8mr	Capítulo 1
Nico Westerdale	Capítulo 5
Nirav Joshi	Capítulo 16
NotJustin	Capitulo 6
Nux	Capitulo 2
ochi	Capitulo 4
Ozan	Capítulo 16
Pranav C Balan	Capítulo 12
Proto	Capitulo 11
Renier	Capitulo 3
rmondesilva	Capítulo 8
Roko C. Buljan	Capítulos 1, 9 y 18
Rupali Pemare	Capítulo 10
Scimonster	Capítulos 4 y 5
secelite	Capitulo 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
aún_aprendiendo	Capitulo 7
sucil	Capítulo 8
Suganya	Capítulo 1
Sunny R Gupta	Capítulo 6
Sverri M. Olsen	Capítulo 8
TheDeadMedic	Capitulo 5
Theodore K.	Capítulo 10
El_Externo	Capítulos 8 y 10
Travis J	Capítulos 1, 2 y 18
Upal Roy	Capitulo 5
usuario1851673	Capítulo 17
usuario2314737	Capítulo 10
VJS	Capítulo 14
Washington Guedes	Capítulo 6
HERIDOStevenJones	Capítulo 2
Yosvel Quintero	Capítulos 1 y 16

Zakaria Acharki	Capítulo 6
Zaz	Capítulos 2 y 10

También le puede interesar