**EBook Gratis**

# APRENDIZAJE jQuery

Free unaffiliated eBook created from
**Stack Overflow contributors.**

#jquery

# Tabla de contenido

# Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: jquery

It is an unofficial and free jQuery ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official jQuery.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

# Capítulo 1: Empezando con jQuery

## Observaciones

jQuery es una biblioteca de JavaScript que simplifica las operaciones DOM, el manejo de eventos, AJAX y las animaciones. También se encarga de muchos problemas de compatibilidad del navegador en los motores DOM y JavaScript subyacentes.

Cada versión de jQuery se puede descargar desde https://code.jquery.com/jquery/ en formatos comprimidos (minified) y sin comprimir.

## Versiones

| Versión | Notas | Fecha de lanzamiento |
|---|---|---|
| 1.0 | Primer lanzamiento 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 del módulo ajax | 2011-01-31 |
| 1.6 | Ganancias de rendimiento significativas 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 incorporaron correcciones de errores y diferencias reportadas de los ciclos beta 1.9 y 2.0 | 2013-05-24 |
| 1.11 | | 2014-01-24 |
| 1.12 | | 2016-01-08 |
| 2.0 | Se eliminó el soporte de IE 6–8 para mejorar el rendimiento | 2013-04-18 |

| Versión | Notas | Fecha de lanzamiento |
|---|---|---|
| | y reducir el tamaño | |
| 2.1 | | 2014-01-24 |
| 2.2 | | 2016-01-08 |
| 3.0 | Aceleraciones masivas para algunos selectores personalizados jQuery | 2016-06-09 |
| 3.1 | No más errores silenciosos | 2016-07-07 |

# Examples

## jQuery Namespace ("jQuery" y "$")

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

`$` es un alias para `jQuery` y los dos pueden intercambiarse entre sí (excepto cuando se ha utilizado `jQuery.noConflict();` consulte Evitar colisiones de espacios de nombres ).

Suponiendo que tenemos este fragmento de HTML -

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

Podríamos usar jQuery para agregar algo de contenido de texto a este div. Para hacer esto podríamos usar la función jQuery `text()` . Esto podría escribirse usando `jQuery` o `$` . es decir

```
jQuery("#demo_div").text("Demo Text!");
```

O

```
$("#demo_div").text("Demo Text!");
```

Ambos resultarán en el mismo HTML final -

```
<div id="demo_div" class="demo">Demo Text!</div>
```

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

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

### Empezando

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

```
<!DOCTYPE html>
<html>
<head>
    <title>Hello, World!</title>
</head>
<body>
    <div>
        <p id="hello">Some random text</p>
    </div>
    <script src="https://code.jquery.com/jquery-2.2.4.min.js"></script>
    <script>
        $(document).ready(function() {
            $('#hello').text('Hello, World!');
        });
    </script>
</body>
</html>
```

Demo en vivo en JSBin

Abra este archivo en un navegador web. Como resultado, verá una página con el texto: `Hello, World!`

## Explicación del código

1. Carga la biblioteca jQuery desde el CDN 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.

   > *Tenga en cuenta que uno de los errores más comunes que se cometen al incluir jQuery es no cargar la biblioteca ANTES de otros scripts o bibliotecas que puedan depender o hacer uso de ella.*

2. Ofrece una función que se ejecutará cuando se detecte que el DOM ( Modelo de objeto de documento ) está "listo" por jQuery:

   ```
   // When the `document` is `ready`, execute this function `...`
   $(document).ready(function() { ... });

   // A commonly used shorthand version (behaves the same as the above)
   $(function() { ... });
   ```

3. Una vez que el DOM está listo, jQuery ejecuta la función de devolución de llamada que se muestra arriba. Dentro de nuestra función, solo hay una llamada que hace 2 cosas principales:

1. Obtiene el elemento con el atributo `id` igual a `hello` (nuestro selector `#hello` ). Usar un selector como el argumento pasado es el núcleo de la funcionalidad y el nombramiento de jQuery; la biblioteca entera esencialmente evolucionó a partir de la extensión document.querySelectorAll MDN .

2. Establezca el `text()` dentro del elemento seleccionado en `Hello, World!` .

```
#    ↓ - Pass a `selector` to `$` jQuery, returns our element
$('#hello').text('Hello, World!');
#             ↑ - Set the Text on the element
```

Para más información consulte la página de documentación de jQuery .

## Incluir etiqueta de script en el encabezado de la página HTML

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


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


Copie el código completo (o haga clic en el icono de copia) y péguelo en el `<head>` o `<body>` de su html.

La mejor práctica es cargar cualquier biblioteca de JavaScript externa en la etiqueta de cabecera con el atributo `async` . Aquí hay una demostración:

```
<!DOCTYPE html>
    <html>
      <head>
         <title>Loading jquery-2.2.4</title>
         <script src="https://code.jquery.com/jquery-2.2.4.min.js" async></script>
      </head>
      <body>
          <p>This page is loaded with jquery.</p>
      </body>
   </html>
```

Cuando utilice un atributo `async` tenga en cuenta que las bibliotecas de javascript se cargan y ejecutan de forma asíncrona tan pronto como estén disponibles. Si se incluyen dos bibliotecas donde la segunda biblioteca depende de la primera, en este caso, si la segunda biblioteca se carga y se ejecuta antes de la primera biblioteca, puede generar un error y la aplicación puede fallar.

## Evitar las colisiones entre espacios de nombres.

Las bibliotecas que no sean jQuery también pueden usar `$` como alias. Esto puede causar interferencia entre esas bibliotecas y jQuery.

Para liberar `$` para usar con otras bibliotecas:

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

Después de llamar a esta función, `$` ya no es un alias para `jQuery` . Sin embargo, aún puede usar la variable `jQuery` para acceder a las funciones de jQuery:

```
jQuery('#hello').text('Hello, World!');
```

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

```
var jqy = jQuery.noConflict();
jqy('#hello').text('Hello, World!');
```

A la inversa, para evitar que otras bibliotecas interfieran con jQuery, puede ajustar su código jQuery en una expresión de función invocada de inmediato (IIFE) y pasar `jQuery` como argumento:

```
(function($) {
    $(document).ready(function() {
        $('#hello').text('Hello, World!');
    });
})(jQuery);
```

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

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

```
jQuery(function( $ ) { // DOM is ready
   // You're now free to use $ alias
   $('#hello').text('Hello, World!');
});
```

Para resumir,

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

Ahora, existe un tercer escenario: ¿qué **`jQuery2`** si queremos que jQuery esté disponible **solo en `jQuery2`** ? Utilizar,

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

Esto da como resultado que ni `$` ni `jQuery` refieran a jQuery.

Esto es útil cuando se deben 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>
    // Here, jQuery1 refers to jQuery 1.12.4 while, $ and jQuery refers to jQuery 3.1.0.
</script>
```

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

### Cargando jQuery vía consola en una página que no lo tiene.

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

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

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

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

### El objeto jQuery

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

```
// Define a local copy of jQuery
jQuery = function( selector, context ) {

    // The jQuery object is actually just the init constructor 'enhanced'
    // Need init if jQuery is called (just allow error to be thrown if not included)
    return new jQuery.fn.init( selector, context );
}
```

Internamente, jQuery se refiere a su prototipo como `.fn` , y el estilo utilizado aquí para crear instancias internas de un objeto jQuery permite que ese prototipo quede expuesto sin el uso explícito de `new` por parte del llamante.

Además de configurar una instancia (que es la forma en que se expone la API de jQuery, como `.each` , `children` , `filter` , etc.), jQuery interno también creará una estructura similar a una matriz para que coincida con el resultado del selector (siempre que algo más que nada, `undefined` , `null` o similar fue pasado como argumento). En el caso de un solo elemento, esta estructura similar a una matriz mantendrá solo ese elemento.

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

```
var $div = $("#myDiv");//populate the jQuery object with the result of the id selector
var div = $div[0];//access array-like structure of jQuery object to get the DOM Element
```

## Cargando plugins jQuery con espacios de nombre

Normalmente, al cargar complementos, asegúrese de incluir siempre el complemento *después de* jQuery.

```
<script src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
<script src="some-plugin.min.js"></script>
```

Si *debe* usar más de una versión de jQuery, asegúrese de cargar el (los) complemento (s) *después de* la versión requerida de jQuery seguido de un código para configurar `jQuery.noConflict(true)` ; luego cargue la próxima versión de jQuery y sus complementos asociados:

```
<script src="https://code.jquery.com/jquery-1.7.0.min.js"></script>
<script src="plugin-needs-1.7.min.js"></script>
<script>
// save reference to jQuery v1.7.0
var $oldjq = jQuery.noConflict(true);
</script>
<script src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
<script src="newer-plugin.min.js"></script>
```

Ahora, al inicializar los complementos, deberá utilizar la versión jQuery asociada.

```
<script>
// newer jQuery document ready
jQuery(function($){
  // "$" refers to the newer version of jQuery
  // inside of this function

  // initialize newer plugin
  $('#new').newerPlugin();
});

// older jQuery document ready
$oldjq(function($){
  // "$" refers to the older version of jQuery
  // inside of this function

  // initialize plugin needing older jQuery
  $('#old').olderPlugin();
});
</script>
```

Es posible usar solo una función de documento listo para inicializar ambos complementos, pero para evitar confusiones y problemas con cualquier código jQuery adicional dentro de la función de documento listo, sería mejor mantener las referencias separadas.

Lea Empezando con jQuery en línea: https://riptutorial.com/es/jquery/topic/211/empezando-con-jquery

# Capítulo 2: Adjuntar

## Sintaxis

1. $ (selector) .append (contenido)
2. $ (contenido) .appendTo (selector)

## Parámetros

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

## Observaciones

- .append () & .after () puede potencialmente ejecutar código. Esto puede ocurrir mediante la inyección de etiquetas de script o el uso de atributos HTML que ejecutan código (por ejemplo,). No utilice estos métodos para insertar cadenas obtenidas de fuentes no confiables, como parámetros de consulta de URL, cookies o entradas de formularios. Si lo hace, puede introducir vulnerabilidades de scripts entre sitios (XSS). Elimine o elimine cualquier entrada del usuario antes de agregar contenido al documento.
- jQuery no soporta oficialmente SVG. Usando métodos jQuery en SVG
  Los documentos, a menos que estén documentados explícitamente para ese método, pueden causar comportamientos inesperados. Ejemplos de métodos que soportan SVG a partir de
  jQuery 3.0 son addClass y removeClass.

## Examples

### Anexando un elemento a un contenedor

**Solución 1:**

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

**Solución 2:**

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

Ambas soluciones están agregando el elemento `#child` (agregando al final) al elemento `#parent` .

Antes de:

```
<div id="parent">
  <span>other content</span>
</div>
<div id="child">

</div>
```

Después:

```
<div id="parent">
  <span>other content</span>
  <div id="child">

  </div>
</div>
```

---

**Nota:** cuando agregue contenido que ya exista en el documento, este contenido se eliminará de su contenedor principal original y se agregará al nuevo contenedor principal. Por lo tanto, no puede utilizar `.append()` o `.appendTo()` para clonar un elemento. Si necesita un clon, use `.clone()` -> [ http://api.jquery.com/clone/◆◆1]

### Uso eficiente y consecutivo de .append ()

Empezando:

# HTML

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

# JS

```
var data = [
    { type: "Name", content: "John Doe" },
    { type: "Birthdate", content: "01/01/1970" },
    { type: "Salary", content: "$40,000,000" },
    // ...300 more rows...
    { type: "Favorite Flavour", content: "Sour" }
];
```

---

**Anexando dentro de un bucle**

Acabas de recibir una gran variedad de datos. Ahora es el momento de recorrer y renderizarlo en la página.

Tu primer pensamiento puede ser hacer algo como esto:

```
var i;                          // <- the current item number
var count = data.length;        // <- the total
var row;                        // <- for holding a reference to our row object

// Loop over the array
for ( i = 0; i < count; ++i ) {
    row = data[ i ];

    // Put the whole row into your table
    $('#my-table').append(
        $('<tr></tr>').append(
            $('<td></td>').html(row.type),
            $('<td></td>').html(row.content)
        )
    );
}
```

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

# No hagas esto.

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

**Cada uno** forzará al navegador a volver a calcular el ancho, la altura y los valores de 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>` ), ellos no pueden.

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

---

**Mejores opciones**

## 1. Agregar a una matriz separada, agregar después de que se complete el bucle

```
/**
 * Repeated DOM traversal (following the tree of elements down until you reach
 * what you're looking for - like our <table>) should also be avoided wherever possible.
 */

// Keep the table cached in a variable then use it until you think it's been removed
var $myTable = $('#my-table');

// To hold our new <tr> jQuery objects
var rowElements = [];

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

```
var row;

// Loop over the array
for ( i = 0; i < count; ++i ) {
    rowElements.push(
        $('<tr></tr>').append(
            $('<td></td>').html(row.type),
            $('<td></td>').html(row.content)
        )
    );
}

// Finally, insert ALL rows at once
$myTable.append(rowElements);
```

De estas opciones, esta es la que más confía en jQuery.

---

## 2. Usando métodos modernos de Array. *

```
var $myTable = $('#my-table');

// Looping with the .map() method
//  - This will give us a brand new array based on the result of our callback function
var rowElements = data.map(function ( row ) {

    // Create a row
    var $row = $('<tr></tr>');

    // Create the columns
    var $type = $('<td></td>').html(row.type);
    var $content = $('<td></td>').html(row.content);

    // Add the columns to the row
    $row.append($type, $content);

    // Add to the newly-generated array
    return $row;
});

// Finally, put ALL of the rows into your table
$myTable.append(rowElements);
```

Funcionalmente equivalente al anterior, solo más fácil de leer.

---

## 3. Uso de cadenas de HTML (en lugar de los métodos incorporados 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;
});

// Using .join('') here combines all the separate strings into one
$myTable.append(rowElements.join(''));
```

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

---

## 4. Crear elementos manualmente, añadir al fragmento del documento.

```
var $myTable = $(document.getElementById('my-table'));

/**
 * Create a document fragment to hold our columns
 * - after appending this to each row, it empties itself
 *   so we can re-use it in the next iteration.
 */
var colFragment = document.createDocumentFragment();

/**
 * Loop over the array using .reduce() this time.
 * We get a nice, tidy output without any side-effects.
 *  - In this example, the result will be a
 *    document fragment holding all the <tr> elements.
 */
var rowFragment = data.reduce(function ( fragment, row ) {

    // Create a row
    var rowEl = document.createElement('tr');

    // Create the columns and the inner text nodes
    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);

    // Add the columns to the column fragment
    // - this would be useful if columns were iterated over separately
    //   but in this example it's just for show and tell.
    colFragment.appendChild(typeEl);
    colFragment.appendChild(contentEl);

    rowEl.appendChild(colFragment);

    // Add rowEl to fragment - this acts as a temporary buffer to
    // accumulate multiple DOM nodes before bulk insertion
    fragment.appendChild(rowEl);
```

```
    return fragment;
}, document.createDocumentFragment());

// Now dump the whole fragment into your table
$myTable.append(rowFragment);
```

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

---

# Bucear más profundo

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

## jQuery anexar

HTML

```
<p>This is a nice </p>
<p>I like </p>

<ul>
  <li>List item 1</li>
  <li>List item 2</li>
  <li>List item 3</li>
</ul>

<button id="btn-1">Append text</button>
<button id="btn-2">Append list item</button>
```

Guión

```
$("#btn-1").click(function(){
    $("p").append(" <b>Book</b>.");
});
$("#btn-2").click(function(){
    $("ul").append("<li>Appended list item</li>");
 });
});
```

Lea Adjuntar en línea: https://riptutorial.com/es/jquery/topic/1910/adjuntar

# Capítulo 3: Ajax

## Sintaxis

- var jqXHR = $ .ajax (url [, configuración])
- var jqXHR = $ .ajax ([configuración])
- jqXHR.done (función (data, textStatus, jqXHR) {});
- jqXHR.fail (función (jqXHR, textStatus, errorThrown) {});
- jqXHR.always (function (jqXHR) {});

## Parámetros

| Parámetro | Detalles |
|---|---|
| url | Especifica la URL a la que se enviará la solicitud. |
| ajustes | Un objeto que contiene numerosos valores que afectan el comportamiento de la solicitud. |
| tipo | El método HTTP que se utilizará para la solicitud. |
| datos | Datos a ser enviados por la solicitud. |
| éxito | Una función de devolución de llamada que se llamará si la solicitud se realiza correctamente |
| error | Una devolución de llamada para manejar el error |
| código de estado | Un objeto de códigos y funciones HTTP numéricos que deben llamarse cuando la respuesta tiene el código correspondiente |
| tipo de datos | El tipo de datos que esperas del servidor. |
| tipo de contenido | Tipo de contenido de los datos a enviar al servidor. El valor predeterminado es "application / x-www-form-urlencoded; charset = UTF-8" |
| contexto | Especifica el contexto que se usará dentro de las devoluciones de llamada, generalmente `this` que se refiere al objetivo actual. |

## Observaciones

AJAX significa **A** avaScript **J** avaScript **a** nd **X** ML. AJAX permite que una página web realice una solicitud HTTP asíncrona (AJAX) al servidor y reciba una respuesta, sin necesidad de volver a cargar toda la página.

# Examples

## Manejo de códigos de respuesta HTTP con $ .ajax ()

Además de `.done` , `.fail` y. `.always` prometen devoluciones de llamada, que se activan en función de si la solicitud fue exitosa o no, existe la opción de activar una función cuando se devuelve un Código de Estado HTTP específico desde el servidor. Esto se puede hacer usando el parámetro `statusCode` .

```
$.ajax({
    type: {POST or GET or PUT etc.},
    url:  {server.url},
    data: {someData: true},
    statusCode: {
        404: function(responseObject, textStatus, jqXHR) {
            // No content found (404)
            // This code will be executed if the server returns a 404 response
        },
        503: function(responseObject, textStatus, errorThrown) {
            // Service Unavailable (503)
            // This code will be executed if the server returns a 503 response
        }
    }
})
.done(function(data){
    alert(data);
})
.fail(function(jqXHR, textStatus){
    alert('Something went wrong: ' + textStatus);
})
.always(function(jqXHR, textStatus) {
   alert('Ajax request was finished')
});
```

Como indica la documentación oficial de jQuery:

> Si la solicitud es exitosa, las funciones del código de estado toman los mismos parámetros que la devolución de llamada exitosa; si da como resultado un error (incluida la redirección 3xx), toman los mismos parámetros que la devolución de llamada de `error` .

## Uso de Ajax para enviar un formulario

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

Supongamos que tiene esta forma simple -

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

```
</form>
```

Se puede usar el siguiente código jQuery (dentro de `$(document).ready` call) -

```
$('#ajax_form').submit(function(event){
  event.preventDefault();
  var $form = $(this);

  $.ajax({
      type: 'POST',
      url: $form.attr('action'),
      data: $form.serialize(),
      success: function(data) {
        // Do something with the response
      },
      error: function(error) {
        // Do something with the error
      }
  });
});
```

**Explicación**

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

## Enviando datos JSON

jQuery hace que el manejo de las *respuestas* jSON sea indoloro, pero se requiere un poco más de trabajo cuando una solicitud determinada desea *enviar* datos en formato JSON:

```
 $.ajax("/json-consuming-route", {
      data: JSON.stringify({author: {name: "Bullwinkle J. Moose",
                                     email: "bullwinkle@example.com"} }),
      method: "POST",
      contentType: "application/json"
   });
```

Observe que estamos especificando el tipo de `contentType` correcto para los datos que estamos enviando; esta es una buena práctica en general y puede ser requerida por la API a la que está publicando, pero *también* tiene el efecto secundario de indicar a jQuery que no realice la conversión predeterminada de `%20` a `+` , lo que haría si `contentType` fuera se deja en el valor predeterminado de `application/x-www-form-urlencoded` . Si, por algún motivo, debe dejar

contentType configurado en el valor predeterminado, asegúrese de establecer `processData` en false para evitarlo.

La llamada a `JSON.stringify` podría evitarse aquí, pero su uso nos permite proporcionar los datos en forma de un objeto de JavaScript (evitando así los vergonzosos errores de sintaxis de JSON, como no citar nombres de propiedades).

## Todo en uno ejemplos

**Ajax consigue:**

**Solución 1:**

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

**Solución 2:**

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

**Ajax Load:** otro método ajax get creado para simplcity

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

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

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

Si se llama a .load con un método de devolución de llamada, la solicitud al servidor será una publicación

```
$('#update-box').load('url.php', {data: "something"}, function(resolve){
    //do something
});
```

**Ajax Post:**

**Solución 1:**

```
$.post('url.php',
```

```
    {date1Name: data1Value, date2Name: data2Value},  //data to be posted
    function(data){
        $('#update-box').html(data);
    }
);
```

### Solución 2:

```
$.ajax({
    type: 'Post',
    url: 'url.php',
    data: {date1Name: data1Value, date2Name: data2Value}  //data to be posted
}).done(function(data){
    $('#update-box').html(data);
}).fail(function(jqXHR, textStatus){
    alert('Error occured: ' + textStatus);
});
```

### Ajax Post JSON:

```
var postData = {
    Name: name,
    Address: address,
    Phone: phone
};

$.ajax({
     type: "POST",
     url: "url.php",
     dataType: "json",
     data: JSON.stringfy(postData),
     success: function (data) {
         //here variable data is in JSON format
     }
 });
```

### Ajax Get JSON:

### Solución 1:

```
$.getJSON('url.php', function(data){
    //here variable data is in JSON format
});
```

### Solución 2:

```
 $.ajax({
      type: "Get",
      url: "url.php",
      dataType: "json",
      data: JSON.stringfy(postData),
      success: function (data) {
          //here variable data is in JSON format
      },
      error: function(jqXHR, textStatus){
          alert('Error occured: ' + textStatus);
```

```
      }
  });
```

### Ajax abortar una llamada o solicitud

```
var xhr = $.ajax({
    type: "POST",
    url: "some.php",
    data: "name=John&location=Boston",
    success: function(msg){
       alert( "Data Saved: " + msg );
    }
});
```

// mata la solicitud

```
xhr.abort()
```

### Archivos Ajax

# 1. Un ejemplo simple completo

Podríamos usar este código de muestra para cargar los archivos seleccionados por el usuario cada vez que se realiza una nueva selección de archivos.

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

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

    $.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, //add the FormData object to the data parameter
        processData: false, //tell jquery not to process data
        contentType: false, //tell jquery not to set content-type
        success: function (response, status, jqxhr) {
            //handle success
        },
        error: function (jqxhr, status, errorMessage) {
            //handle error
        }
```

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

Ahora vamos a desglosarlo e inspeccionarlo parte por parte.

# 2. Trabajar con entradas de archivos

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

Antes de que subamos los archivos, primero tenemos que darle al usuario una manera de seleccionar los archivos que desea cargar. Para ello utilizaremos un `file input` . La propiedad `multiple` permite seleccionar más de un archivo, puede eliminarlo si desea que el usuario seleccione un archivo a la vez.

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

Vamos a utilizar el `change event` de entrada para capturar los archivos.

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

Dentro de la función de controlador, accedemos a los archivos a través de la propiedad de archivos de nuestra entrada. Esto nos da una Lista de archivos , que es un objeto similar a una matriz.

# 3. Creando y llenando los datos de formulario

Para cargar archivos con Ajax vamos a utilizar FormData .

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

FileList que obtuvimos en el paso anterior es una matriz similar a un objeto y se puede iterar usando varios métodos, incluyendo for loop , for ... of loop y jQuery.each . Nos quedaremos con el jQuery en este ejemplo.

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

Utilizaremos el método de adición de FormData para agregar los archivos a nuestro objeto formdata.

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

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

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

# 4. Enviando los archivos con Ajax

```
$.ajax({
    url: "/Test/Url",
    type: "post",
    data: fdata, //add the FormData object to the data parameter
    processData: false, //tell jquery not to process data
    contentType: false, //tell jquery not to set content-type
    success: function (response, status, jqxhr) {
        //handle success
    },
    error: function (jqxhr, status, errorMessage) {
        //handle error
    }
});
```

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

Lea Ajax en línea: https://riptutorial.com/es/jquery/topic/316/ajax

# Capítulo 4: Atravesando DOM

## Examples

### Selecciona los hijos del elemento.

Para seleccionar los elementos secundarios de un elemento, puede utilizar el método `children()` .

```
<div class="parent">
    <h2>A headline</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` :

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

El método acepta un argumento `selector` opcional que se puede usar para filtrar los elementos que se devuelven.

```
// Only get "p" children
$('.parent').children("p").css("color", "green");
```

### Iterando sobre la lista de elementos jQuery

Cuando necesite iterar sobre la lista de elementos jQuery.

Considere esta estructura DOM:

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

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

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

*Consejo: la* `key` es el índice del elemento `div.red` que estamos iterando actualmente, dentro de su elemento primario. `ele` es el elemento HTML, por lo que podemos crear un objeto jQuery a partir de él usando `$()` o `jQuery()` , así: `$(ele)` . Después, podemos llamar a cualquier método jQuery en el objeto, como `css()` u `hide()` etc. En este ejemplo, simplemente extraemos el texto del objeto.

## Seleccionando hermanos

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

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

```
<ul class="menu">
    <li class="selected">Home</li>
    <li>Blog</li>
    <li>About</li>
</ul>
```

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

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

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

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

## método más cercano ()

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

HTML

```
<div id="abc" class="row">
    <div id="xyz" class="row">
    </div>
    <p id="origin">
      Hello
    </p>
</div>
```

jQuery

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

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

SALIDA

```
"Closest row: abc"
```

```
"Closest p: origin"
```

**Método first ():** el primer método devuelve el primer elemento del conjunto de elementos coincidentes.

HTML

```
<div class='.firstExample'>
   <p>This is first paragraph in a div.</p>
   <p>This is second paragraph in a div.</p>
   <p>This is third paragraph in a div.</p>
   <p>This is fourth paragraph in a div.</p>
   <p>This is fifth paragraph in a div.</p>
</div>
```

JQuery

```
var firstParagraph = $("div p").first();
console.log("First paragraph:", firstParagraph.text());
```

Salida:

```
First paragraph: This is first paragraph in a div.
```

## Obtener el siguiente elemento

Para obtener el siguiente elemento puede usar el método `.next()` .

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

Si está de pie en el elemento "Anna" y desea obtener el siguiente elemento, "Paul", el método `.next()` le permitirá hacerlo.

```
// "Paul" now has green text
$(".anna").next().css("color", "green");
```

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

```
// Next element is a "li", "Paul" now has green text
$(".anna").next("li").css("color", "green");
```

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

```
// Next element is not a ".mark", nothing will be done in this case
$(".anna").next(".mark").css("color", "green");
```

### Obtener elemento anterior

Para obtener el elemento anterior puede usar el método `.prev()` .

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

Si se encuentra en el elemento "Anna" y desea obtener el elemento anterior, "Mark", el método `.prev()` le permitirá hacerlo.

```
// "Mark" now has green text
$(".anna").prev().css("color", "green");
```

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

```
// Previous element is a "li", "Mark" now has green text
$(".anna").prev("li").css("color", "green");
```

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

```
// Previous element is not a ".paul", nothing will be done in this case
$(".anna").prev(".paul").css("color", "green");
```

### Filtrar una selección

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

El método se llama en una selección y devuelve una nueva selección. Si el filtro coincide con un elemento, entonces se agrega a la selección devuelta, de lo contrario se ignora. Si no hay ningún elemento que coincida, se devuelve una selección vacía.

## El HTML

Este es el HTML que usaremos.

```
<ul>
    <li class="zero">Zero</li>
    <li class="one">One</li>
    <li class="two">Two</li>
    <li class="three">Three</li>
</ul>
```

## Selector

Filtrar usando selectores es una de las formas más simples de filtrar una selección.

```
$("li").filter(":even").css("color", "green"); // Color even elements green
$("li").filter(".one").css("font-weight", "bold"); // Make ".one" bold
```

## Función

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

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

```
var selection = $("li").filter(function (index, element) {
    // "index" is the position of the element
    // "element" is the same as "this"
    return $(this).hasClass("two");
});
selection.css("color", "green"); // ".two" will be colored green
```

## 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(three).css("color", "green");
```

## Selección

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

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

### método encontrar ()

El método .find () nos permite buscar a través de los descendientes de estos elementos en el árbol DOM y construir un nuevo objeto jQuery a partir de los elementos correspondientes.

HTML

```
     <div class="parent">
        <div class="children" name="first">
            <ul>
```

```
                <li>A1</li>
                <li>A2</li>
                <li>A3</li>
            </ul>
        </div>
        <div class="children" name="second">
            <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**

Lea Atravesando DOM en línea: https://riptutorial.com/es/jquery/topic/1189/atravesando-dom

# Capítulo 5: Atributos

## Observaciones

La función jQuery `.attr()` obtiene el valor de un atributo para el **primer** elemento en el conjunto de elementos coincidentes o establece uno o más atributos para **cada** elemento coincidente.

Vale la pena señalar que al obtener el valor de un atributo, solo lo obtiene del primer elemento que coincide con el selector (es decir, `$("input").attr("type");` solo obtendría el tipo de la primera entrada , si hay más de uno)

Sin embargo, al configurar un atributo, se aplicará a todos los elementos coincidentes.

## Examples

### Obtener el valor de atributo de un elemento HTML

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

Sintaxis:

```
$([selector]).attr([attribute name]);
```

Ejemplo:

HTML:

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

jQuery:

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

**Obteniendo atributos de `data` :**

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

Sintaxis:

```
$([selector]).data([attribute name]);
```

Ejemplo:

HTML:

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

jQuery:

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

**Nota:**

> El método data () de jQuery le dará acceso a los atributos data- *, PERO, obstruye el caso del nombre del atributo. Referencia

## Valor de ajuste del atributo HTML

Si desea agregar un atributo a algún elemento, puede usar la función `attr(attributeName, attributeValue)` . Por ejemplo:

```
$('a').attr('title', 'Click me');
```

Este ejemplo agregará el texto `"Click me"` mouse `"Click me"` en todos los enlaces de la página.

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

## Quitando atributo

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

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

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

## Diferencia entre attr () y prop ()

`attr()` obtiene / establece el atributo HTML usando 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 establecer una casilla de verificación como está marcado:

```
$('#tosAccept').prop('checked', true); // using attr() won't work properly here
```

Para eliminar una propiedad puede usar el método `removeProp()` . De forma similar, `removeAttr()` elimina los atributos.

Lea Atributos en línea: https://riptutorial.com/es/jquery/topic/4429/atributos

# Capítulo 6: Cada funcion

## Examples

### Uso basico

```
// array
var arr = [
   'one',
   'two',
   'three',
   'four'
];
$.each(arr, function (index, value) {
  console.log(value);

  // Will stop running after "three"
  return (value !== 'three');
});
// Outputs: one two three
```

### jQuery cada función

HTML:

```
<ul>
  <li>Mango</li>
  <li>Book</li>
</ul>
```

Guión:

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

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

0: Mango

1: Libro

Lea Cada funcion en línea: https://riptutorial.com/es/jquery/topic/10853/cada-funcion

# Capítulo 7: Casilla de verificación Seleccionar todo con la opción de marcar / desmarcar automáticamente otro cambio de casilla de verificación

## Introducción

He usado varios ejemplos y respuestas de Stackoverflow para llegar a este ejemplo realmente simple sobre cómo administrar la casilla de verificación "seleccionar todos" junto con una casilla de verificación / deselección automática si alguno de los cambios en el estado de la casilla de verificación del grupo. Restricción: la identificación "seleccionar todo" debe coincidir con los nombres de entrada para crear el grupo seleccionar todo. En el ejemplo, la entrada seleccionar todo ID es cbGroup1. Los nombres de entrada también son cbGroup1

El código es muy corto, no muy abundante si el enunciado (consume tiempo y recursos).

## Examples

### 2 seleccionar todas las casillas de verificación con las correspondientes casillas de verificación de grupo

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>

<p>
<input id="cbGroup1" type="checkbox">Select all
<input name="cbGroup1" type="checkbox" value="value1_1">Group1 value 1
<input name="cbGroup1" type="checkbox" value="value1_2">Group1 value 2
<input name="cbGroup1" type="checkbox" value="value1_3">Group1 value 3
</p>
<p>
<input id="cbGroup2" type="checkbox">Select all
<input name="cbGroup2" type="checkbox" value="value2_1">Group2 value 1
<input name="cbGroup2" type="checkbox" value="value2_2">Group2 value 2
<input name="cbGroup2" type="checkbox" value="value2_3">Group2 value 3
</p>

<script type="text/javascript" language="javascript">
    $("input").change(function() {
      $('input[name=\''+this.id+'\']').not(this).prop('checked', this.checked);
      $('#'+this.name).prop('checked', $('input[name=\''+this.name+'\']').length ===
$('input[name=\''+this.name+'\']').filter(':checked').length);
    });
</script>
```

Lea Casilla de verificación Seleccionar todo con la opción de marcar / desmarcar automáticamente otro cambio de casilla de verificación en línea:

https://riptutorial.com/es/jquery/topic/10076/casilla-de-verificacion-seleccionar-todo-con-la-opcion-de-marcar---desmarcar-automaticamente-otro-cambio-de-casilla-de-verificacion

# Capítulo 8: Complementos

## Examples

### Plugins - Primeros pasos

La API de jQuery se puede ampliar agregando a su prototipo. Por ejemplo, la API existente ya tiene muchas funciones disponibles, como `.hide()` , `.fadeIn()` , `.hasClass()` , etc.

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

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

Agregar 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 prefiere).

Un objeto jQuery construido tendrá una matriz interna de elementos en función del selector que se le haya pasado. Por ejemplo, `$('.active')` construirá un objeto jQuery que contiene elementos con la clase activa, en el momento de la llamada (como en, esto no es un conjunto vivo de elementos).

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

**Plugin básico** :

```
$.fn.highlight = function() {
    this.css({ background: "yellow" });
};

// Use example:
$("span").highlight();
```

Ejemplo de jsFiddle

#### Encadenamiento y reutilización

**A diferencia del ejemplo anterior** , se espera que los Complementos de jQuery sean **Chainable** .
Lo que esto significa es la posibilidad de encadenar múltiples Métodos a una misma Colección de Elementos como `$(".warn").append("WARNING! ").css({color:"red"})` (vea cómo usamos el `.css()` Método `.css()` después de `.append()` , ambos métodos se aplican en la misma colección `.warn` )

Permitir que uno use el mismo complemento en diferentes colecciones, las diferentes opciones de personalización desempeñan un papel importante en la **personalización / reutilización**

```
(function($) {
  $.fn.highlight = function( custom ) {

    // Default settings
    var settings = $.extend({
        color : "",              // Default to current text color
        background : "yellow"    // Default to yellow background
    }, custom);

    return this.css({     // `return this` maintains method chainability
        color : settings.color,
        backgroundColor : settings.background
    });

  };
}( jQuery ));


// Use Default settings
$("span").highlight();    // you can chain other methods

// Use Custom settings
$("span").highlight({
    background: "#f00",
    color: "white"
});
```

demo jsFiddle

**Libertad**

Los ejemplos anteriores están en el ámbito de la comprensión de la creación básica de complementos. Tenga en cuenta que no debe restringir a un usuario a un conjunto limitado de opciones de personalización.

Digamos, por ejemplo, que desea crear un `.highlight()` donde pueda pasar una cadena de **texto** que se destacará y permitir la máxima libertad con respecto a los estilos:

```
//...
// Default settings
var settings = $.extend({
   text : "",          // text to highlight
   class : "highlight" // reference to CSS class
}, custom);

return this.each(function() {
   // your word highlighting logic here
});
//...
```

el usuario ahora puede pasar un **texto** deseado y tener control completo sobre los estilos agregados usando una clase de CSS personalizada:

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

```
});
```

Ejemplo de jsFiddle

## Método jQuery.fn.extend ()

Este método extiende el objeto prototipo jQuery ($ .fn) para proporcionar nuevos métodos personalizados que se pueden encadenar a la función jQuery ().

Por ejemplo:

```
<div>Hello</div>
<div>World!</div>

<script>
jQuery.fn.extend({
  // returns combination of div texts as a result
  getMessage: function() {
    var result;
    // this keyword corresponds result array of jquery selection
    this.each(function() {
        // $(this) corresponds each div element in this loop
        result = result + " " + $(this).val();
    });
    return result;
  }
});

// newly created .getMessage() method
var message = $("div").getMessage();

// message = Hello World!
console.log(message);
</script>
```

Lea Complementos en línea: https://riptutorial.com/es/jquery/topic/1805/complementos

# Capítulo 9: evento listo para documentos

## Examples

### ¿Qué es el documento listo y cómo debo usarlo?

El código jQuery a menudo se envuelve en `jQuery(function($) { ... });` de modo que solo se ejecuta después de que el DOM haya terminado de cargarse.

```
<script type="text/javascript">
  jQuery(function($) {
    // this will set the div's text to "Hello".
    $("#myDiv").text("Hello");
  });
</script>

<div id="myDiv">Text</div>
```

Esto es importante porque jQuery (y JavaScript en general) no puede seleccionar un elemento DOM que no se haya procesado en la página.

```
<script type="text/javascript">
  // no element with id="myDiv" exists at this point, so $("#myDiv") is an
  // empty selection, and this will have no effect
  $("#myDiv").text("Hello");
</script>

<div id="myDiv">Text</div>
```

Tenga en cuenta que puede asignar un alias al espacio de nombres jQuery pasando un controlador personalizado al método `.ready()` . Esto es útil para los casos en que otra biblioteca JS usa el mismo `$` alias acortado que *jQuery* , lo que crea un conflicto. Para evitar este conflicto, debe llamar a `$.noConflict();` - Esto le obliga a usar solo el espacio de nombres *jQuery* predeterminado (en lugar del `$` alias corto).
Al pasar un controlador personalizado al controlador `.ready()` , podrá elegir el nombre de alias para usar *jQuery* .

```
$.noConflict();

jQuery( document ).ready(function( $ ) {
   // Here we can use '$' as jQuery alias without it conflicting with other
   // libraries that use the same namespace
   $('body').append('<div>Hello</div>')
});

jQuery( document ).ready(function( jq ) {
   // Here we use a custom jQuery alias 'jq'
   jq('body').append('<div>Hello</div>')
});
```

En lugar de simplemente colocar su código jQuery en la parte inferior de la página, el uso de la función `$(document).ready` garantiza que todos los elementos HTML se hayan procesado y que todo el Modelo de objetos de documento (DOM) esté listo para que se ejecute el código JavaScript.

### jQuery 2.2.3 y anteriores

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

```
$(function() {
  // code
});

$().ready(function() {
  // code
});

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

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

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

### jQuery 3.0

## Notación

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

```
jQuery(function($) {
  // Run when document is ready
  // $ (first argument) will be internal reference to jQuery
  // Never rely on $ being a reference to jQuery in the global namespace
});
```

Todos los demás manejadores de documentos están en desuso en jQuery 3.0.

## Asincrónico

A partir de jQuery 3.0, el controlador listo siempre se llamará de forma asíncrona . Esto significa que en el código a continuación, el registro 'manejador externo' siempre se mostrará primero, sin importar si el documento estaba listo en el punto de ejecución.

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

> manejador externo
> manejador interno

## Diferencia entre $ (documento) .ready () y $ (ventana) .load ()

`$(window).load()` quedó **en desuso en la versión 1.8 de jQuery (y se eliminó por completo de jQuery 3.0 )** y, como tal, ya no debe utilizarse. Las razones de la desaprobación se indican en la página jQuery sobre este evento.

> Advertencias del evento de carga cuando se utiliza con imágenes.
>
> Un desafío común que los desarrolladores intentan resolver con el `.load()` abreviado `.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. Estos son:
>
> - No funciona de forma coherente ni fiable entre navegadores
> - No se dispara correctamente en WebKit si la imagen `src` está configurada en la misma `src` que antes
> - No burbujea correctamente el árbol DOM
> - Puede dejar de disparar para imágenes que ya viven en el caché del navegador.

Si aún desea usar `load()` está documentado a continuación:

---

`$(document).ready()` espera hasta que el DOM completo esté disponible: todos los elementos en el HTML se han analizado y están en el documento. Sin embargo, es posible que recursos como las imágenes no se hayan cargado completamente en este punto. Si es importante esperar hasta que se carguen todos los recursos, `$(window).load()` **y es consciente de las limitaciones significativas de este evento** , en su lugar se puede usar lo siguiente:

```
$(document).ready(function() {
  console.log($("#my_large_image").height()); // may be 0 because the image isn't available
});

$(window).load(function() {
  console.log($("#my_large_image").height()); // will be correct
});
```

## Adjuntando eventos y manipulando el DOM dentro de ready ()

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

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

```
$(document).ready(function() {
  $("button").click(function() {
    // Code for the click function
  });
});
```

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

```
jQuery(function($) {
// set the value of an element.
   $("#myElement").val("Hello");
});
```

3. **Manipular la estructura DOM cargada**
   Por ejemplo: oculte un `div` cuando la página se carga por primera vez y muéstrelo en el evento de clic de un botón

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

## Diferencia entre jQuery (fn) y la ejecución de su código antes

El uso del evento preparado para documentos puede tener pequeños inconvenientes en el rendimiento , con una ejecución demorada de hasta ~ 300 ms. A veces se puede lograr el mismo comportamiento mediante la ejecución del código justo antes de la etiqueta de cierre `</body>` :

```
<body>
  <span id="greeting"></span> world!
  <script>
    $("#greeting").text("Hello");
  </script>
</body>
```

producirá un comportamiento similar, pero tendrá un rendimiento más rápido que el que no espera el desencadenante de evento preparado para el documento como lo hace en:

```
<head>
  <script>
    jQuery(function($) {
      $("#greeting").text("Hello");
    });
  </script>
</head>
<body>
  <span id="greeting"></span> world!
</body>
```

Haga hincapié en el hecho de que el primer ejemplo se basa en su conocimiento de su página y

la ubicación del script justo antes de la etiqueta de cierre `</body>` y específicamente después de la etiqueta `span` .

Lea evento listo para documentos en línea: https://riptutorial.com/es/jquery/topic/500/evento-listo-para-documentos

# Capítulo 10: Eventos

## Observaciones

jQuery maneja internamente eventos a través de la función addEventListener . Esto significa que es perfectamente legal tener más de una función vinculada al mismo evento para el mismo elemento DOM.

## Examples

### Adjuntar y separar controladores de eventos

# Adjuntar un controlador de eventos

Desde la versión 1.7 jQuery tiene el evento API `.on()` . De esta manera, cualquier evento javascript estándar o evento personalizado se puede vincular al elemento jQuery seleccionado actualmente. Existen accesos directos como `.click()` , pero `.on()` le ofrece más opciones.

---

## HTML

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

## jQuery

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

---

# Separar un controlador de eventos

Naturalmente, también tiene la posibilidad de separar eventos de sus objetos jQuery. Lo hace utilizando `.off( events [, selector ] [, handler ] )` .

---

## HTML

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

## jQuery

```
$('#hello').on('click', function(){
    console.log('hello world!');
    $(this).off();
});
```

Al hacer clic en el botón `$(this)` se referirá al objeto jQuery actual y eliminará de él todos los controladores de eventos adjuntos. También puede especificar qué controlador de eventos debe eliminarse.

## jQuery

```
$('#hello').on('click', function(){
    console.log('hello world!');
    $(this).off('click');
});

$('#hello').on('mouseenter', function(){
    console.log('you are about to click');
});
```

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

### Eventos delegados

Vamos a empezar con el ejemplo. Aquí hay un ejemplo muy simple de HTML.

# Ejemplo HTML

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

# El problema

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

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

---

# Información de fondo - propagación de eventos

Los eventos delegados solo son posibles debido a la propagación de eventos (a menudo llamada propagación de eventos). Cada vez que se activa un evento, se propagará hasta el final (a la raíz del documento). *Delegan* el manejo de un evento a un elemento ancestral que no cambia, de ahí el nombre de eventos "delegados".

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

- una
- li
- ul
- cuerpo
- html
- Raiz del documento

---

# Solución

Sabiendo qué hace el burbujeo de eventos, podemos detectar uno de los eventos deseados que se están propagando a través de nuestro HTML.

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

```
$('ul').on('click', 'a', function () {
  console.log(this.href); // jQuery binds the event function to the targeted DOM element
                          // this way `this` refers to the anchor and not to the list
  // Whatever you want to do when link is clicked
});
```

En lo anterior:

- Tenemos 'ul' que es el receptor de este detector de eventos.
- El primer parámetro ('clic') define qué eventos estamos tratando de detectar.
- El segundo parámetro ('a') se usa para declarar desde dónde se debe *originar* el evento (de todos los elementos secundarios bajo el destinatario del oyente del evento, ul).
- Por último, el tercer parámetro es el código que se ejecuta si se cumplen los requisitos del primer y segundo parámetro.

# En detalle cómo funciona la solución.

1. El usuario hace clic en el elemento `<a>`
2. Eso activa el evento de clic en el elemento `<a>` .
3. El evento comienza a burbujear hacia la raíz del documento.
4. El evento burbujea primero en el elemento `<li>` y luego en el elemento `<ul>` .
5. La escucha de eventos se ejecuta ya que el elemento `<ul>` tiene adjunta la escucha de eventos.
6. El detector de eventos primero detecta el evento desencadenante. El evento de propagación es "clic" y el oyente tiene "clic", es un pase.
7. Las comprobaciones de escucha intentan hacer coincidir el segundo parámetro ('a') con cada elemento de la cadena de burbujas. Como el último elemento de la cadena es una 'a', esto coincide con el filtro y también es un pase.
8. El código en el tercer parámetro se ejecuta utilizando el elemento coincidente, ya que es `this` . Si la función no incluye una llamada a `stopPropagation()` , el evento continuará propagándose hacia la raíz ( `document` ).

Nota: Si un antecesor adecuado que no cambia no está disponible / conveniente, debe usar el `document` . Como hábito no use `'body'` por las siguientes razones:

- `body` tiene un error, que tiene que ver con el estilo, que puede significar que los eventos del mouse no le hagan burbujas. Esto depende del navegador y puede ocurrir cuando la altura del cuerpo calculada es 0 (por ejemplo, cuando todos los elementos secundarios tienen posiciones absolutas). Los eventos del ratón siempre burbujean para `document` .
- `document` *siempre* existe en su secuencia de comandos, por lo que puede adjuntar controladores delegados para `document` fuera de un *controlador listo para DOM* y asegurarse de que todavía funcionarán.

## Evento de carga de documentos .load ()

Si desea que su secuencia de comandos espere hasta que se cargue un determinado recurso, como una imagen o un PDF, puede usar `.load()` , que es un atajo para el acceso directo de `.on( "load", handler)` .

**HTML**

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

### jQuery

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

## Eventos para repetir elementos sin usar ID's.

### Problema

Hay una serie de elementos repetidos en la página en los que necesita saber en qué evento ocurrió para hacer algo con esa instancia específica.

### Solución

- Dar a todos los elementos comunes una clase común
- Aplicar el detector de eventos a una clase. `this` controlador de eventos interno es el elemento selector correspondiente en el que ocurrió el evento
- Recorra el contenedor externo más repetitivo para esa instancia comenzando en `this`
- Use `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="delete">Delete</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="delete">Delete</button>
   </div>
</div>
```

jQuery

```
$(function() {
  $('.delete').on('click', function() {
    // "this" is element event occured on
    var $btn = $(this);
    // traverse to wrapper container
    var $itemWrap = $btn.closest('.item-wrapper');
    // look within wrapper to get person for this button instance
    var person = $itemWrap.find('.person').text();
    // send delete to server and remove from page on success of ajax
    $.post('url/string', { id: $itemWrap.data('item_id')}).done(function(response) {
      $itemWrap.remove()
    }).fail(function() {
      alert('Ooops, not deleted at server');
    });
  });
```

```
});
```

### originalEvento

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

# Obtener la dirección de desplazamiento

```
$(document).on("wheel",function(e){
    console.log(e.originalEvent.deltaY)
    // Returns a value between -100 and 100 depending on the direction you are scrolling
})
```

### Activar y desactivar eventos específicos a través de jQuery. (Oyentes nombrados)

A veces desea desactivar todos los oyentes registrados previamente.

```
//Adding a normal click handler
$(document).on("click",function(){
    console.log("Document Clicked 1")
});
//Adding another click handler
$(document).on("click",function(){
    console.log("Document Clicked 2")
});
//Removing all registered handlers.
$(document).off("click")
```

Un problema con este método es que TODOS los escuchas vinculados en el `document` por otros complementos, etc., también serían eliminados.

**La mayoría de las veces, deseamos separar a todos los oyentes conectados solo por nosotros.**

Para lograr esto, podemos unir a oyentes nombrados como,

```
//Add named event listener.
$(document).on("click.mymodule",function(){
    console.log("Document Clicked 1")
});
$(document).on("click.mymodule",function(){
    console.log("Document Clicked 2")
});

//Remove named event listener.
$(document).off("click.mymodule");
```

Esto asegura que cualquier otro detector de clics no se modifique inadvertidamente.

Lea Eventos en línea: https://riptutorial.com/es/jquery/topic/1321/eventos

# Capítulo 11: jQuery .animate () Método

## Sintaxis

1. (selector) .animate ({styles}, {options})

## Parámetros

| Parámetro | Detalles |
|---|---|
| propiedades | Un objeto de propiedades y valores CSS que la animación moverá hacia |
| duración | (predeterminado: 400) Una cadena o número que determina la duración de la animación |
| facilitando | (predeterminado: swing) Una cadena que indica qué función de aceleración usar para la transición |
| completar | Una función para llamar una vez que se completa la animación, llamada una vez por elemento coincidente. |
| comienzo | Especifica una función para ser ejecutada cuando comienza la animación. |
| paso | Especifica una función para ser ejecutada para cada paso en la animación. |
| cola | un valor booleano que especifica si colocar o no la animación en la cola de efectos. |
| Progreso | Especifica una función para ser ejecutada después de cada paso en la animación. |
| hecho | Especifica una función para ser ejecutada cuando finaliza la animación. |
| fallar | especifica una función que se ejecutará si la animación no se completa. |
| especialEasing | un mapa de una o más propiedades CSS del parámetro de estilos, y sus correspondientes funciones de aceleración. |
| siempre | especifica una función que se ejecutará si la animación se detiene sin completar. |

## Examples

### Animación con devolución de llamada

A veces necesitamos 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 automáticamente para mejorar la atracción de nuestro sitio web o aplicaciones web. JQuery ayuda mucho con este concepto usando `fadeIn()`, `hide()`, `slideDown()` pero su funcionalidad es limitada y solo realizó la tarea específica que se le asigna.

Jquery soluciona este problema proporcionando un método sorprendente y flexible llamado `.animate()` . Este método permite establecer animaciones personalizadas que se utilizan las propiedades de css que dan permiso para volar sobre los bordes. por ejemplo si damos propiedad de estilo css como `width:200;` y la posición actual del elemento DOM es 50, el método animar 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 lo manejará.

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js"></script>
<script>
    $("#btn1").click(function(){
        $("#box").animate({width: "200px"});
    });
</script>

<button id="btn1">Animate Width</button>
<div id="box" style="background:#98bf21;height:100px;width:100px;margin:6px;"></div>
```

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

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

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

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

### Facilitación especificada en el método `.animate()` .

| |
|---|
| "oscilación" |
| "lineal" |

Aquí hay algunos ejemplos con complejas opciones de animación.

Ej. 1:

```
$( "#book" ).animate({
  width: [ "toggle", "swing" ],
  height: [ "toggle", "swing" ],
```

```
  opacity: "toggle"
 }, 5000, "linear", function() {
    $( this ).after( "<div>Animation complete.</div>" );
});
```

Ejemplo 2:

```
  $("#box").animate({
     height: "300px",
     width: "300px"
     }, {
     duration: 5000,
     easing: "linear",
     complete: function(){
        $(this).after("<p>Animation is complete!</p>");
     }
  });
```

Lea jQuery .animate () Método en línea: https://riptutorial.com/es/jquery/topic/5064/jquery--animate----metodo

# Capítulo 12: jQuery Objetos diferidos y Promesas

## Introducción

Las promesas de jQuery son una forma inteligente de encadenar operaciones asíncronas de manera constructiva. Esto reemplaza la anidación de devoluciones de llamada en la vieja escuela, que no se reorganizan tan fácilmente.

## Examples

### Creación de promesa básica

Aquí hay un ejemplo muy simple de una función que " *promete* continuar cuando transcurra un tiempo determinado". Lo hace creando un nuevo objeto `Deferred` , que se resuelve más tarde y devuelve la promesa del aplazado:

```
function waitPromise(milliseconds){

   // Create a new Deferred object using the jQuery static method
   var def = $.Deferred();

   // Do some asynchronous work - in this case a simple timer
   setTimeout(function(){

       // Work completed... resolve the deferred, so it's promise will proceed
       def.resolve();
   }, milliseconds);

   // Immediately return a "promise to proceed when the wait time ends"
   return def.promise();
}
```

Y usar así:

```
waitPromise(2000).then(function(){
     console.log("I have waited long enough");
});
```

### Promesas asíncronas de encadenamiento

Si tiene varias tareas asíncronas que deben ocurrir una después de la otra, deberá encadenar sus objetos de promesa. Aquí hay un ejemplo simple:

```
function First() {
    console.log("Calling Function First");
    return $.get("/ajax/GetFunction/First");
}
```

```
function Second() {
    console.log("Calling Function Second");
    return $.get("/ajax/GetFunction/Second");
}

function Third() {
    console.log("Calling Function Third");
    return $.get("/ajax/GetFunction/Third");
}

function log(results){
    console.log("Result from previous AJAX call: " + results.data);
}

First().done(log)
       .then(Second).done(log)
       .then(Third).done(log);
```

## jQuery ajax () éxito, error VS.done (), .fail ()

**éxito y error:** una devolución de llamada **exitosa** que se invoca al completar con éxito una solicitud Ajax.

Una devolución de llamada **fallida** que se invoca en caso de que haya algún error al realizar la solicitud.

**Ejemplo:**

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

**.done () y .fail ():**

.ajax (). done (función (datos, estado de texto, jqXHR) {}); Reemplaza el método .success () que estaba en desuso en jQuery 1.8. Esta es una construcción alternativa para la función de devolución de llamada exitosa anterior.

.ajax (). fail (función (jqXHR, textStatus, errorThrown) {}); Reemplaza el método .error () que estaba en desuso en jQuery 1.8. Esta es una construcción alternativa para la función de devolución de llamada completa anterior.

**Ejemplo:**

```
$.ajax({
    url: 'URL',
    type: 'POST',
    data: yourData,
```

```
    datatype: 'json'
})
.done(function (data) { successFunction(data); })
.fail(function (jqXHR, textStatus, errorThrown) { serrorFunction(); });
```

### Obtener el estado actual de una promesa.

Por defecto, el estado de una promesa está pendiente cuando se crea. El estado de una promesa cambia cuando el objeto diferido que creó la promesa, ya sea que la resuelva o la rechace.

```
var deferred = new $.Deferred();
var d1= deferred.promise({
    prop: "value"
});
var d2= $("div").promise();
var d3= $("div").hide(1000).promise();

console.log(d1.state()); // "pending"
console.log(d2.state()); // "resolved"
console.log(d3.state()); // "pending"
```

Lea jQuery Objetos diferidos y Promesas en línea:
https://riptutorial.com/es/jquery/topic/8308/jquery-objetos-diferidos-y-promesas

# Capítulo 13: Manipulación de CSS

## Sintaxis

- .css (cssProperty) // Obtiene el valor de la propiedad CSS representada
- .css ([cssProperty, ...]) // Obtener valores de la matriz de cssProperties
- .css (cssProperty, valor) // Establecer valor
- .css ({cssProperty: value, ...}) // Establecer propiedades y valores
- .css (cssProperty, function) // Exponer cssProperty a una función de devolución de llamada

## Observaciones

### Valores renderizados

Si se usa una unidad de respuesta (como `"auto"` , `"%"` , `"vw"` etc.), `.css()` devolverá el valor real renderizado en `px`

```
.myElement{ width: 20%; }
```

```
var width = $(".myElement").css("width"); // "123px"
```

### Formateo de propiedades y valores

**Las propiedades** se pueden definir usando **el formato CSS estándar como String** o usando **camelCase**

```
"margin-bottom"
marginBottom
```

**Los valores** deben expresarse en String. Los valores numéricos se tratan como unidades `px` internamente por jQuery

```
.css(fontSize: "1em")
.css(fontSize: "16px")
.css(fontSize: 16)      // px will be used
```

### A partir de jQuery 3, evite utilizar `.show()` y `.hide()`

De acuerdo con esta publicación del blog de jQuery , debido a problemas de sobrecarga y rendimiento, ya no debe usar `.show()` o `.hide()` .

> Si tiene elementos en una hoja de estilo que están configurados para `display: none` , el método `.show()` ya no lo reemplazará. Por lo tanto, la regla más importante para pasar a jQuery 3.0 es la siguiente: no use una hoja de estilo para establecer el valor

predeterminado de `display: none` y luego intente usar `.show()` - o cualquier método que muestre elementos, como `.slideDown()` y `.fadeIn()` - para hacerlo visible. Si necesita que un elemento esté oculto de manera predeterminada, la mejor manera es agregar un nombre de clase como "oculto" al elemento y definir esa clase para `display: none` en una hoja de estilo. Luego puede agregar o eliminar esa clase utilizando los `.addClass()` y `.removeClass()` jQuery para controlar la visibilidad. Alternativamente, puede hacer que un controlador `.ready()` llame a `.hide()` en los elementos antes de que se muestren en la página. O, si realmente debe conservar el valor predeterminado de la hoja de estilo, puede usar `.css("display", "block")` (o el valor de pantalla apropiado) para anular la hoja de estilo.

# Examples

## Establecer propiedad CSS

Configuración de un solo estilo:

```
$('#target-element').css('color', '#000000');
```

Configurando múltiples estilos al mismo tiempo:

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

## Obtener propiedad CSS

Para obtener la propiedad CSS de un elemento, puede usar el método `.css(propertyName)` :

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

## Incremento / Decremento de propiedades numéricas

Las propiedades numéricas de CSS se pueden incrementar y disminuir con la sintaxis `+=` y `-=` , respectivamente, usando el método `.css()` :

```
  // Increment using the += syntax
  $("#target-element").css("font-size", "+=10");

  // You can also specify the unit to increment by
  $("#target-element").css("width", "+=100pt");
  $("#target-element").css("top", "+=30px");
  $("#target-element").css("left", "+=3em");

  // Decrementing is done by using the -= syntax
  $("#target-element").css("height", "-=50pt");
```

# CSS Getter

La función de **captador** `.css()` se puede aplicar a todos los elementos DOM en la página, como los siguientes:

```
// Rendered width in px as a string. ex: `150px`
// Notice the `as a string` designation - if you require a true integer,
// refer to `$.width()` method
$("body").css("width");
```

Esta línea devolverá el **ancho computado** del elemento especificado, cada propiedad CSS que proporcione entre paréntesis dará el valor de la propiedad para este elemento DOM `$("selector")` , si solicita un atributo CSS que no existe obtendrá `undefined` como respuesta.

También puede llamar al **captador de CSS** con una matriz de atributos:

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

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

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

# CSS Setter

El `.css()` método **setter** también se puede aplicar a cada elemento DOM en la página.

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

Esta declaración establece el `width` del `$("selector")` a `500px` y devuelve el objeto jQuery para que pueda encadenar más métodos al selector especificado.

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

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

Todos los cambios que hizo el colocador se agregan a la propiedad de `style` elemento DOM, lo que afecta los estilos de los elementos (a menos que el valor de la propiedad de estilo ya esté definido como `!important` en otro lugar en los estilos)

Lea Manipulación de CSS en línea: https://riptutorial.com/es/jquery/topic/2732/manipulacion-de-css

# Capítulo 14: Manipulación de DOM

## Examples

### Creando elementos DOM

La función `jQuery` (normalmente con un alias como `$` ) se puede usar tanto para seleccionar elementos como para crear nuevos elementos.

```
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 desea crear. En este ejemplo es un ancla pero podría ser cualquier cosa en esta lista . Ver la especificación de referencia de la `a` elemento.

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

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

### Manipular las clases de elementos.

Asumiendo que la página incluye un elemento HTML como:

```
<p class="small-paragraph">
  This is a small <a href="https://en.wikipedia.org/wiki/Paragraph">paragraph</a>
  with a <a class="trusted" href="http://stackexchange.com">link</a> inside.
</p>
```

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

```
$('p').hasClass('small-paragraph'); // true
$('p').hasClass('large-paragraph'); // false

// Add a class to all links within paragraphs
$('p a').addClass('untrusted-link-in-paragraph');

// Remove the class from a.trusted
$('a.trusted.untrusted-link-in-paragraph')
.removeClass('untrusted-link-in-paragraph')
.addClass('trusted-link-in-paragraph');
```

### Alternar una clase

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

```
$(".small-paragraph").toggleClass("pretty");
```

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");}
else {
   $(".small-paragraph").addClass("pretty"); }
```

alternar dos clases:

```
$(".small-paragraph").toggleClass("pretty cool");
```

Booleano para agregar / eliminar clases:

```
$(".small-paragraph").toggleClass("pretty",true); //cannot be truthy/falsey

$(".small-paragraph").toggleClass("pretty",false);
```

Función para alternar la clase (vea el ejemplo más abajo para evitar un problema)

```
$( "div.surface" ).toggleClass(function() {
  if ( $( this ).parent().is( ".water" ) ) {
    return "wet";
  } else {
    return "dry";
  }
});
```

Utilizado en ejemplos:

```
// functions to use in examples
function stringContains(myString, mySubString) {
  return myString.indexOf(mySubString) !== -1;
}
function isOdd(num) { return num % 2;}
var showClass = true; //we want to add the class
```

Ejemplos:

Usa el índice del elemento para alternar entre las clases impar / par

```
$( "div.gridrow" ).toggleClass(function(index,oldClasses, false), showClass ) {
  showClass
  if ( isOdd(index) ) {
    return "wet";
```

```
  } else {
    return "dry";
  }
});
```

**Más complejo ejemplo de `toggleClass` , dado un marcado de cuadrícula simple**

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

Funciones simples para nuestros ejemplos:

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

function stringContains(myString, mySubString) {
  return myString.indexOf(mySubString) !== -1;
}
var showClass = true; //we want to add the class
```

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

```
$("div.gridrow").toggleClass(function(index, oldClasses, showThisClass) {
  if (isOdd(index)) {
    return "odd";
  } else {
    return "even";
  }
  return oldClasses;
}, showClass);
```

Si la fila tiene una clase de `gridfooter` , elimine las clases impares / pares, conserve el resto.

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

Las clases que se devuelven son las que se efectúan. Aquí, si un elemento no tiene un `gridfooter` , agregue una clase para par / impar. Este ejemplo ilustra el retorno de la lista de clases OLD. Si esto `else return oldClasses;` se elimina, solo se agregan las nuevas clases, por lo tanto, la fila con una clase de `gridfooter` tendría todas las clases eliminadas si no hubiéramos devuelto las anteriores; de lo contrario, se habrían cambiado (eliminado).

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

## Otros métodos API

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

El primero es el método .empty () .

Imagina el siguiente marcado:

```
<div id="content">
  <div>Some text</div>
</div>
```

Llamando a `$('#content').empty();` , el div interior sería eliminado. Esto también podría lograrse utilizando `$('#content').html('');` .

Otra función útil es la función .closest () :

```
<tr id="row_1">
  <td><button type="button" class="delete">Delete</button>
</tr>
```

Si desea encontrar la fila más cercana a un botón en el que se hizo clic dentro de una de las celdas de la fila, puede hacer esto:

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

Como probablemente habrá varias filas, cada una con sus propios botones de `delete` , usamos `$(this)` dentro de la función .click () para limitar el alcance al botón en el que realmente hicimos clic.

Si desea obtener el `id` de la fila que contiene el botón `Delete` que hizo clic, podría hacer algo como esto:

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

Por lo general, se considera una buena práctica prefijar las variables que contienen objetos jQuery con un `$` (signo de dólar) para aclarar qué es la variable.

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 () también:

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

`.parent()` solo sube un nivel del árbol DOM, por lo que es bastante inflexible, si tuviera que cambiar el botón de eliminar para que esté contenido dentro de un `span` por ejemplo, el selector jQuery se rompería.

## .html ()

Puede usar este método para reemplazar todo el HTML dentro del selector. Asumiendo que tienes un elemento html como este

```
<div class="row">
    <div class="col-md-12">
        <div id="information">
            <p>Old message</p>
        </div>
    </div>
</div>
```

Podrías usar `.html()` . para eliminar y agregar una alerta o texto informativo para alertar a los usuarios con una sola línea.

```
$("#information").html("<p>This is the new alert!</p>");
```

## Elementos de clasificación

Para ordenar los elementos de manera eficiente (todos a la vez y con una mínima interrupción de DOM), necesitamos:

1. **Encontrar** los elementos
2. **Clasificación** basada en una condición establecida
3. **Insertar de** nuevo en el DOM

```
<ul id='my-color-list'>
    <li class="disabled">Red</li>
```

```
    <li>Green</li>
    <li class="disabled">Purple</li>
    <li>Orange</li>
</ul>
```

1. Encuéntralos - `.children()` o `.find()`

   Esto nos devolverá un objeto similar a un Array para jugar.

   ```
   var $myColorList = $('#my-color-list');

   // Elements one layer deep get .children(), any deeper go with .find()
   var $colors = $myColorList.children('li');
   ```

2. Re-organizarlos - `Array.prototype.sort()`

   Actualmente está configurado para devolver los elementos en orden ascendente según el contenido HTML (también conocido como sus colores).

   ```
   /**
    * Bind $colors to the sort method so we don't have to travel up
    * all these properties more than once.
    */
   var sortList = Array.prototype.sort.bind($colors);

   sortList(function ( a, b ) {

       // Cache inner content from the first element (a) and the next sibling (b)
       var aText = a.innerHTML;
       var bText = b.innerHTML;

       // Returning -1 will place element `a` before element `b`
       if ( aText < bText ) {
           return -1;
       }

       // Returning 1 will do the opposite
       if ( aText > bText ) {
           return 1;
       }

       // Returning 0 leaves them as-is
       return 0;
   });
   ```

3. `.append()` - `.append()`

   *Tenga en cuenta que no es necesario separar los elementos primero: `append()` moverá los elementos que ya existen en el DOM, por lo que no tendremos copias adicionales*

   ```
   // Put it right back where we got it
   $myColorList.append($colors);
   ```

# Hazlo lindo

## Añadir un botón de clasificación

```
<!-- previous HTML above -->
<button type='button' id='btn-sort'>
    Sort
</button>
```

## Establecer el valor inicial de la dirección de clasificación

```
var ascending = true;
```

## `sortList()` caché nuestros elementos DOM y `sortList()` aquí para minimizar nuestro procesamiento DOM

```
var $myColorList = $('#my-color-list');
var $colors = $myColorList.children('li');
var sortList = Array.prototype.sort.bind($colors);
```

## Envuelve todo en una función `doSort()`

```
// Put the sortList() and detach/append calls in this portable little thing
var doSort = function ( ascending ) {

    sortList(function ( a, b ) {
        // ...
    });

    $myColorList.append($colors);
};
```

## Agregar el controlador de clic para `$('#btn-sort')`

```
$('#btn-sort').on('click', function () {
    // Run the sort and pass in the direction value
    doSort(ascending);

    // Toggle direction and save
    ascending = !ascending;
});
```

# Todos juntos ahora

```
var ascending = true;

var $myColorList = $('#my-color-list');
var $colors = $myColorList.children('li');
var sortList = Array.prototype.sort.bind($colors);

var doSort = function ( ascending ) {

    sortList(function ( a, b ) {

        var aText = a.innerHTML;
        var bText = b.innerHTML;

        if ( aText < bText ) {
            return ascending ? -1 : 1;
        }

        if ( aText > bText ) {
            return ascending ? 1 : -1;
        }

        return 0;
    });

    $myColorList.append($colors);

};

$('#btn-sort').on('click', function () {
    doSort(ascending);
    ascending = !ascending;
});
```

---

**Prima**

# Clasificación multinivel (agrupación de elementos ordenados)

```
// ...

var doSort = function ( ascending ) {

    sortList(function ( a, b ) {

        // ...initial sorting...

    }).sort(function ( a, b ) {

        // We take the returned items and sort them once more
        var aClass = a.className;
        var bClass = b.className;

        // Let's group the disabled items together and put them at the end

        /**
         * If the two elements being compared have the same class
         * then there's no need to move anything.
```

```
         */
        if ( aClass !== bClass ) {
            return aClass === 'disabled' ? 1 : -1;
        }
        return 0;
    });

    // And finalize with re-insert
    $myColorList.append($colors);
};

// ...
```

**¿Puedes ir un paso más allá?**

## Agregar otro botón para alternar la clasificación de grupos deshabilitados

MDN - Array.prototype.sort ()

Lea Manipulación de DOM en línea: https://riptutorial.com/es/jquery/topic/512/manipulacion-de-dom

# Capítulo 15: Obtención y configuración del ancho y alto de un elemento.

## Examples

### Obtención y configuración de ancho y alto (ignorando borde)

Obtener ancho y alto:

```
var width = $('#target-element').width();
var height = $('#target-element').height();
```

Establecer ancho y alto:

```
$('#target-element').width(50);
$('#target-element').height(100);
```

### Obtención y configuración de InternalWidth y innerHeight (ignorando el relleno y el borde)

Obtener ancho y alto:

```
var width = $('#target-element').innerWidth();
var height = $('#target-element').innerHeight();
```

Establecer ancho y alto:

```
$('#target-element').innerWidth(50);
$('#target-element').innerHeight(100);
```

### Obtención y configuración de la anchura exterior y la altura externa (incluido el relleno y el borde)

Obtener ancho y alto (excluyendo margen):

```
var width = $('#target-element').outerWidth();
var height = $('#target-element').outerHeight();
```

Obtener ancho y alto (incluyendo margen):

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

Establecer ancho y alto:

```
$('#target-element').outerWidth(50);
$('#target-element').outerHeight(100);
```

Lea Obtención y configuración del ancho y alto de un elemento. en línea: https://riptutorial.com/es/jquery/topic/2167/obtencion-y-configuracion-del-ancho-y-alto-de-un-elemento-

# Capítulo 16: Prepend

## Examples

### Prependiendo un elemento a un contenedor

**Solución 1:**

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

**Solución 2:**

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

Ambas soluciones están anteponiendo el elemento `#child` (agregando al principio) al elemento `#parent` .

Antes de:

```
<div id="parent">
  <span>other content</span>
</div>
<div id="child">

</div>
```

Después:

```
<div id="parent">
  <div id="child">

  </div>
  <span>other content</span>
</div>
```

### Método prepend

`prepend()` : inserta contenido, especificado por el parámetro, al principio de cada elemento en el conjunto de elementos coincidentes.

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

```
// with html string
jQuery('#parent').prepend('<span>child</span>');
// or you can use jQuery object
jQuery('#parent').prepend($('#child'));
// or you can use comma seperated multiple elements to prepend
jQuery('#parent').prepend($('#child1'),$('#child2'));
```

**2.** `prepend(function)`

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

```
jQuery('#parent').prepend(function(i,oldHTML){
     // return the value to be prepend
     return  '<span>child</span>';
});
```

Lea Prepend en línea: https://riptutorial.com/es/jquery/topic/1909/prepend

# Capítulo 17: Selectores

## Introducción

Los selectores de jQuery seleccionan o encuentran un elemento DOM (modelo de objeto de documento) en un documento HTML. Se utiliza para seleccionar elementos HTML basados en id, nombre, tipos, atributos, clase, etc. Se basa en selectores de CSS existentes.

## Sintaxis

- Etiqueta: Sin marcador, usa la etiqueta directamente
- Id: `#id`
- Clase: `.className`
- Atributo: `[attributeName]`
- Atributo con valor: `[attributeName ='value']`
- El atributo comienza con el valor `^=` : `[attributeName ^= 'value']`
- El atributo termina con el valor `$=` : `[attributeName $='value']`
- El atributo contiene el valor `*=` : `[attributeName *= 'value']`
- Pseudo-selector `:pseudo-selector`
- Cualquier descendiente: espacio entre selectores.
- Niños directos: `>` entre selectores
- Hermano adyacente siguiendo el primero: `+`
- Hermano no adyacente después de la primera: `~`
- O: `,` (coma) entre el selector

## Observaciones

Al escribir `selectors` para la `class` o `id` o `attribute` que contiene algunos caracteres especiales como

```
! " # $ % & ' ( ) * + , . / : ; < = > ? @ [ \ ] ^ { | } ~
```

los caracteres deben escaparse utilizando dos barras `\\` inversas `\\` .

p.ej.

```
<span id="temp.foobar"><span>
```

los selectores serán enmarcados como,

```
$('#temp\\.foobar')
```

## Examples

## Tipos de selectores

En jQuery puede seleccionar elementos en una página usando muchas propiedades diferentes del elemento, incluyendo:

- Tipo
- Clase
- CARNÉ DE IDENTIDAD
- Posesión de Atributo
- Valor de atributo
- Selector indexado
- Seudoestado

Si conoce los selectores de CSS , notará que los selectores en jQuery son los mismos (con pequeñas excepciones).

Tome el siguiente HTML, por ejemplo:

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

### Seleccionando por tipo:

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

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

### Seleccionando por clase

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

```
$(".example")
```

### Seleccionando por ID

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

```
$("#second-link")
```

### Seleccionando por Posesión de Atributo

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

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

### Selección por valor de atributo

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

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

### Selección por posición *indexada* ( *Selector indexado* )

El siguiente selector de jQuery seleccionará solo 1, el segundo `<a>` es decir. el `second-link` porque el índice proporcionado es `1` como `eq(1)` (Tenga en cuenta que el índice comienza en `0` por lo que el segundo se seleccionó aquí).

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

### Selección con exclusión excluida

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

Lo siguiente selecciona elementos `<a>` , excepto que con el `example` clase, que es 1

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

### Seleccionando con Exclusión

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

Lo siguiente selecciona elementos `<a>` , excepto aquellos con el `example` clase, que son 1 y 2.

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

### Seleccionando por pseudo-estado

También puede seleccionar en jQuery usando pseudo-estados, incluyendo `:first-child` `:last-child` `:first-of-type` `:last-of-type` , etc.

El siguiente selector de jQuery solo seleccionará el primer elemento `<a>` : número 1.

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

### Combinando selectores jQuery

También puede aumentar su especificidad combinando múltiples selectores de 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='something'][attr3='something']:first-of-type:first-child")
```

Esto seleccionaría un elemento `<a>` que:

- Tiene las siguientes clases: `class1, class2, and class3`
- Tiene el siguiente ID: `someID`
- Tiene el siguiente atributo: `attr1`
- Tiene los siguientes atributos y valores: `attr2` con `something` valor, `attr3` con `something` valor
- 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 niños y hermanos**

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

- Para seleccionar un hijo no directo, use un espacio
- Para seleccionar un hijo directo, use un `>`
- Para seleccionar un hermano adyacente después del primero, use a `+`
- Para seleccionar un hermano no adyacente después del primero, use un `~`

---

**Selección comodín**

Puede haber casos en los que deseamos seleccionar todos los elementos, pero no hay una propiedad común para seleccionar (clase, atributo, etc.). En ese caso podemos usar el selector `*` que simplemente selecciona todos los elementos:

```
$('#wrapper *')    // Select all elements inside #wrapper element
```

## Combinando selectores

Considere seguir la estructura DOM

```
<ul class="parentUl">
    <li> Level 1
        <ul class="childUl">
            <li>Level 1-1 <span> Item - 1 </span></li>
            <li>Level 1-1 <span> Item - 2 </span></li>
        </ul>
    </li>
    <li> Level 2
```

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

---

# Descendientes y selectores infantiles

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

1. **`$('parent child')`** Simple **`$('parent child')`**

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

   Esto hace que todos los descendientes coincidentes del antepasado especificado *bajen todos los niveles* .

2. `>` - **`$('parent > child')`**

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

   Esto encuentra a todos los niños que emparejan ( *solo 1er nivel abajo* ).

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

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

   Esto funciona igual que 1. arriba.

4. `find()` - **`$('parent').find('child')`**

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

   Esto funciona igual que 1. arriba.

5. `children()` - **`$('parent').find('child')`**

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

   Esto funciona igual que el 2 de arriba.

---

# Otros combinadores

### Selector de grupo: ","

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

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

### Selector de múltiplos: "" (sin carácter)

Seleccione todos los elementos `<ul>` con la clase `parentUl` :

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

### Selector de hermanos adyacente: "+"

Seleccione todos los elementos `<li>` que se colocan inmediatamente después de otro elemento `<li>` :

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

### Selector general de hermanos: "~"

Seleccione todos los elementos `<li>` que son hermanos de otros elementos `<li>` :

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

### Visión general

Los elementos pueden seleccionarse por jQuery utilizando los selectores jQuery . La función devuelve un elemento o una lista de elementos.

## Selectores basicos

```
$("*")                      // All elements
$("div")                    // All <div> elements
$(".blue")                  // All elements with class=blue
$(".blue.red")              // All elements with class=blue AND class=red
$(".blue,.red")             // All elements with class=blue OR class=red
$("#headline")              // The (first) element with id=headline
$("[href]")                 // All elements with an href attribute
$("[href='example.com']")   // All elements with href=example.com
```

## Operadores relacionales

```
$("div span")          // All <span>s that are descendants of a <div>
$("div > span")        // All <span>s that are a direct child of a <div>
$("a ~ span")          // All <span>s that are siblings following an <a>
$("a + span")          // All <span>s that are immediately after an <a>
```

### Selectores de almacenamiento en caché

Cada vez que usa un selector en jQuery, el DOM busca elementos que coincidan con su consulta. Hacer esto con demasiada frecuencia o repetidamente disminuirá el rendimiento. Si se refiere a un selector específico más de una vez, debe agregarlo a la memoria caché asignándolo a una variable:

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

Esto reemplazaría:

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

El almacenamiento en caché de este selector podría resultar útil si su sitio web necesita mostrar / ocultar este elemento a menudo. Si hay varios elementos con el mismo selector, la variable se convertirá en una matriz de estos elementos:

```
<div class="parent">
    <div class="child">Child 1</div>
    <div class="child">Child 2</div>
</div>

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

    // output: "Child 1"
</script>
```

**NOTA:** El elemento debe existir en el DOM en el momento de su asignación a una variable. Si no hay ningún elemento en el DOM con una clase llamada `child` , almacenará una matriz vacía en esa variable.

```
<div class="parent"></div>

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

    // output: []

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

    // output: "Child 1"
</script>
```

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

**Nota** : Al almacenar en memoria caché los selectores, muchos desarrolladores iniciarán el

nombre de la variable con un `$` para indicar que la variable es un objeto jQuery así:

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

## Elementos DOM como selectores

jQuery acepta una amplia variedad de parámetros, y uno de ellos es un elemento DOM real. Pasar un elemento DOM a jQuery hará que la estructura subyacente similar a una matriz 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 devoluciones de llamada, donde el elemento actual se pasa al constructor jQuery para obtener acceso a la API jQuery.

Como en `each` devolución de llamada (nota: cada una es una función de iterador).

```
$(".elements").each(function(){
    //the current element is bound to `this` internally by jQuery when using each
    var currentElement = this;

    //at this point, currentElement (or this) has access to the Native API

    //construct a jQuery object with the currentElement(this)
    var $currentElement = $(this);

    //now $currentElement has access to the jQuery API
});
```

## 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 similar a una matriz del objeto jQuery contenga el HTML generado resultante.

jQuery usa expresiones regulares para determinar si la cadena que se pasa al constructor es una cadena HTML, y también que *debe* comenzar con `<` . Esa expresión regular se define como `rquickExpr = /^(?:\s*(<[\w\W]+>)[^>]*|#([\w-]*))$/` ( explicación en regex101.com ).

El uso más común de una cadena HTML como selector es cuando los conjuntos de elementos DOM deben crearse solo en código, a menudo esto lo usan las bibliotecas para cosas como las ventanas emergentes modales.

Por ejemplo, una función que devolvió una etiqueta de ancla envuelta en un div como una plantilla

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

Devolvería un objeto jQuery sosteniendo.

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

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

Lea Selectores en línea: https://riptutorial.com/es/jquery/topic/389/selectores

# Capítulo 18: Visibilidad de los elementos

## Parámetros

| Parámetro | Detalles |
|---|---|
| Duración | Cuando se pasan, los efectos de `.hide()` , `.show()` y `.toggle()` se animan; el elemento (s) se desvanecerá gradualmente dentro o fuera. |

## Examples

### Visión general

```
$(element).hide()            // sets display: none
$(element).show()            // sets display to original value
$(element).toggle()          // toggles between the two
$(element).is(':visible')    // returns true or false
$('element:visible')         // matches all elements that are visible
$('element:hidden')          // matches all elements that are hidden

$('element').fadeIn();          // display the element
$('element').fadeOut();          // hide the element

$('element').fadeIn(1000);          // display the element using timer
$('element').fadeOut(1000);          // hide the element using timer

// display the element using timer and a callback function
$('element').fadeIn(1000, function(){
 // code to execute
});

// hide the element using timer and a callback function
$('element').fadeOut(1000, function(){
   // code to execute
});
```

### Posibilidades de alternar

#### Caso simple `toggle()`

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

#### Con *duración* específica

```
function toggleDuration() {
  $(".target2").toggle("slow"); // A millisecond duration value is also acceptable
}
```

### ... y *devolución de llamada*

```
function toggleCallback() {
  $(".target3").toggle("slow",function(){alert('now do something');});
}
```

### ... o con *flexibilización* y devolución de llamada.

```
function toggleEasingAndCallback() {
  // You may use jQueryUI as the core only supports linear and swing easings
  $(".target4").toggle("slow","linear",function(){alert('now do something');});
}
```

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

```
function toggleWithOptions() {
  $(".target5").toggle(
    { // See all possible options in: api.jquery.com/toggle/#toggle-options
      duration:1000, // milliseconds
      easing:"linear",
      done:function(){
        alert('now do something');
      }
    }
  );
}
```

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

```
function toggleSlide() {
  $(".target6").slideToggle(); // Animates from top to bottom, instead of top corner
}
```

### ... o desvanecerse / desaparecer cambiando la opacidad con `fadeToggle()`

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

### ... o alternar una clase con `toggleClass()`

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

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

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

Todos los ejemplos anteriores se pueden encontrar aquí

Lea Visibilidad de los elementos en línea: https://riptutorial.com/es/jquery/topic/1298/visibilidad-de-los-elementos

# Creditos

| S. No | Capítulos | Contributors |
|---|---|---|
| 1 | Empezando con jQuery | A.J, acdcjunior, amflare, Anil, bwegs, Community, DGS, empiric, Fueled By Coffee, hairboat, Hirshy, Iceman, Igor Raush, J F, jkdev, John C, Kevin B, Kevin Katzke, Kevin Montrose, Luca Putzu, Matas Vaitkevicius, mico, Mottie, Neal, ni8mr, Prateek, RamenChef, Rion Williams, Roko C. Buljan, secelite, Shaunak D, Stephen Leppik, Suganya, Sverri M. Olsen, the12, Travis J, user2314737, Velocibadgery, Yosvel Quintero |
| 2 | Adjuntar | Ashkan Mobayen Khiabani, bipon, Community, Darshak, Deryck, empiric, Flyer53, J F, JF it, Paul Roub, Pranav C Balan, Proto, Shaunak D |
| 3 | Ajax | Alon Eitan, amflare, Andrew Brooke, Ashkan Mobayen Khiabani, Athafoud, atilacamurca, Ben H, Cass, Community, csbarnes, Dr. J. Testington, Edathadan Chief aka Arun, empiric, hasan, joe_young, John C, kapantzak, Kiren Siva, Lacrioque, Marimba, Nirav Joshi, Ozan, shaN, Shaunak D, Teo Dragovic, Yosvel Quintero |
| 4 | Atravesando DOM | A.J, charlietfl, Community, dlsso, mark.hch, rmondesilva, SGS Venkatesh, sucil, Sverri M. Olsen, The_Outsider, Zaz |
| 5 | Atributos | A.J, acdcjunior, ban17, ochi, Scimonster |
| 6 | Cada funcion | bipon, Renier |
| 7 | Casilla de verificación Seleccionar todo con la opción de marcar / desmarcar automáticamente otro cambio de casilla de verificación | user1851673 |
| 8 | Complementos | hasan, Roko C. Buljan, Travis J |
| 9 | evento listo para documentos | Adjit, Alon Eitan, amflare, charlietfl, Emanuel Vintilă, Igor Raush, J F, jkdev, Joram van den Boezem, Liam, Mark Schultheiss, Melanie, Nhan, Nico Westerdale, Scimonster, secelite, TheDeadMedic, the-noob, URoy |

| | | |
|---|---|---|
| 10 | Eventos | Adjit, charlietfl, DelightedD0D, doydoy44, empiric, Gone Coding, Horst Jahns, Jatniel Prinsloo, Kevin B, Louis, Luca Putzu, Marimba, NotJustin, SGS Venkatesh, Stephen Leppik, Sunny R Gupta, Washington Guedes, WMios, Zakaria Acharki |
| 11 | jQuery .animate () Método | RamenChef, Rust in Peace, Simplans, VJS |
| 12 | jQuery Objetos diferidos y Promesas | Alex, Ashiquzzaman, Gone Coding |
| 13 | Manipulación de CSS | abaracedo, Ashkan Mobayen Khiabani, Brandt Solovij, J F, j08691, Jonathan Michalik, Kevin B, Petroff, Roko C. Buljan, ScientiaEtVeritas, Shlomi Haver, Sorangwala Abbasali, Stephen Leppik, Sverri M. Olsen |
| 14 | Manipulación de DOM | Angelos Chalaris, Assimilater, Brock Davis, DawnPaladin, DefyGravity, Deryck, Marimba, Mark Schultheiss, martincarlin87, Neal, Paul Roub, Shaunak D, still_learning |
| 15 | Obtención y configuración del ancho y alto de un elemento. | Ashkan Mobayen Khiabani |
| 16 | Prepend | Ashkan Mobayen Khiabani, empiric, J F, Pranav C Balan |
| 17 | Selectores | alepeino, Alon Eitan, Brock Davis, Castro Roy, David, DelightedD0D, devlin carnate, dlsso, hasan, Iceman, James Donnelly, JLF, John Slegers, kapantzak, Kevin B, Keyslinger, Matas Vaitkevicius, Melanie, MikeC, Nux, Petr R., rbashish, Scimonster, Shaunak D, Shekhar Pankaj, Sorangwala Abbasali, ssb, Sverri M. Olsen, Travis J, whales, WOUNDEDStevenJones, Zaz |
| 18 | Visibilidad de los elementos | Alex Char, Paul Roub, Rupali Pemare, The_Outsider, Theodore K., user2314737, Zaz |