



ÍNDICE

1. OBJETIVOS	2
2. ENCADENAMIENTO O CHAINING	3
3. MÉTODOS TRANSVERSALES DEL DOM (DOM TRAVERSING)	4
MÉTODOS DE FILTRADO.....	4
MÉTODOS DE RECORRIDO DEL ÁRBOL DOM	7
OTROS MÉTODOS TRANSVERSALES DEL DOM	9
4. MANIPULAR EL HTML Y EL CSS DE NUESTRA PÁGINA	12
MANIPULANDO LOS ELEMENTOS DEL DOM.....	12
CAMBIANDO EL HTML	13
OBTENIENDO EL TEXTO DEL OBJETO	13
AÑADIENDO HTML DENTRO DEL ELEMENTO	14
AÑADIENDO HTML ANTES O DESPUÉS DE UN ELEMENTO.....	15
REEMPLAZANDO ELEMENTOS	16
ENVOLVIENDO ELEMENTOS	16
DESENVOLVIENDO ELEMENTOS.....	17
DUPLICANDO ELEMENTOS	17
ELIMINANDO ELEMENTOS	18
ACCEDIENDO A LOS ATRIBUTOS	19
5. TRABAJANDO CON ESTILOS	22
6. INTRODUCCIÓN A LOS EVENTOS	23
7. FIREBUG AVANZADO	25
LA SOLAPA RED	25
LA SOLAPA CONSOLA	26
LA SOLAPA SCRIPT	28
8. EJERCICIOS	30
EJERCICIO 1	30
EJERCICIO 2	31
9. ÍNDICE DE EJEMPLOS Y TABLAS	33
EJEMPLOS	33
TABLAS	33

1. Objetivos

En este segundo tema se pretenden conseguir los siguientes objetivos:

- ✚ Entender el concepto de encadenamiento o chaining.
- ✚ Generar, eliminar y/o modificar elementos del DOM dinámicamente utilizando jQuery.
- ✚ Moverse por el árbol del DOM utilizando jQuery.
- ✚ Manipular los estilos de los elementos utilizando jQuery.
- ✚ Depurar código javascript (jQuery) con Firebug.

2. Encadenamiento o chaining

Con jQuery, es posible seleccionar múltiples conjuntos de elementos y realizar múltiples cosas con ellos, todo dentro de una sola línea de código. Esto es así, porque el resultado de la mayoría de operaciones en un objeto jQuery, es el objeto en sí mismo. Este encadenamiento no solamente ayuda a mantener el código jQuery conciso, sino que también puede mejorar el rendimiento de un script cuando la alternativa, es volver a especificar un selector.

También es posible dividir una sola línea de código en múltiples líneas para mayor legibilidad:

```
$( 'td:contains(Henry)' ) // Encontrar todas las celdas que contienen "Henry"  
.parent()                // Seleccionar su padre  
.addClass(/highlight/)   // Añadirle la clase "highlight"
```

A continuación, veremos los métodos transversales del DOM, cuya utilización, junto con la utilización del encadenamiento, hacen que podamos conseguir cosas muy elaboradas en una sola línea de código.

3. Métodos transversales del DOM (DOM Traversing)

Los selectores jQuery que hemos explorado hasta el momento, nos permiten seleccionar un conjunto de elementos a medida que navegamos por el árbol DOM y filtrar los resultados. Existen muchas ocasiones en las que seleccionar un elemento padre o ancestro es esencial; aquí es donde los métodos transversales DOM de jQuery entran en juego.

Con estos métodos a nuestra disposición, podemos recorrer el DOM con facilidad.

Tenemos tres tipos de métodos transversales:

- ✚ Métodos de filtrado. Nos sirven para filtrar el conjunto de elementos devuelto por un selector.
- ✚ Métodos de recorrido del árbol DOM. Nos sirven para movernos a través de los nodos del árbol DOM.
- ✚ Otros métodos. Nos ayudan a realizar otras tareas.

Métodos de filtrado

La siguiente tabla muestra los diferentes métodos existentes en el DOM transversal para filtrar los elementos devueltos por un selector.

Método	Funcionamiento
.eq(índice)	Reduce el conjunto de elementos seleccionados a un elemento especificado por el índice.
.filter('selector')	Reduce el conjunto de elementos seleccionados a aquellos que son seleccionados por el selector que recibe como parámetro.
.first()	Reduce el conjunto de elementos seleccionados al primero del conjunto.
.has('selector')	Reduce el conjunto de elementos seleccionados a aquellos que tienen un descendiente que se corresponde con el selector pasado como parámetro.
.is('selector')	Devuelve cierto si, al menos, uno de los elementos seleccionados se corresponde con el selector pasado como parámetro.
.last()	Reduce el conjunto de elementos seleccionados al último del conjunto.
.map(callback)	Pasa cada elemento del conjunto a través de una función callback, produciendo un nuevo objeto jQuery que contiene los valores devueltos por dicha función.
.not('selector')	Elimina del conjunto de elementos seleccionados aquellos que coinciden con el selector que le pasamos.
.slice()	Reduce el conjunto de elementos seleccionados a un subconjunto especificado por un rango de índices.

Tabla 1: Métodos de filtrado del DOM transversal

Veamos algunos ejemplos de uso:

- ✚ Poner el color de fondo rojo a los de una lista que, a su vez, contengan otra lista:

```
<!DOCTYPE html>
<html>
```

```
<head>
  <title>Método .is()</title>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <script src="http://code.jquery.com/jquery-1.9.0.min.js"
    type="text/javascript"></script>
  <script type="text/javascript">
    $(function()
    {
      $("li").has("ul").css("background-color", "red");
    });
  </script>
</head>
<body>
  <ul>
    <li>coches</li>
    <li>motos</li>
    <li>accesorios:
      <ul>
        <li>ruedas</li>
        <li>antenas</li>
        <li>radiocassettes</li>
      </ul>
    </li>
  </ul>
</body>
</html>
```

Ejemplo 1: [métodoHas.html](#)

🚦 Verificar si una tabla contiene alguna celda vacía:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Método .is()</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <script src="http://code.jquery.com/jquery-1.9.0.min.js"
      type="text/javascript"></script>
    <script type="text/javascript">
      $(function()
      {
        if ($('table td').is('td:empty'))
          alert("Hay celda vacía");
        else
          alert("No hay celda vacía");
      });
    </script>
  </head>
  <body>
    <table>
      <tr>
        <th>Producto</th><th>Descripción</th><th>Precio</th>
      </tr>
      <tr>
        <td>Estuche de pinturas</td>
        <td>Pinturas de varios colores</td>
        <td>18,99 </td>
      </tr>
      <tr>
        <td>Compás</td>
        <td>De diferentes tamaños</td>
```

```
<td></td>
</tr>
<tr>
  <td>Folios</td>
  <td>Paquete de 500 folios</td>
  <td>5,99 </td>
</tr>
<tr>
  <td>Cartulina</td>
  <td>Tamaño din A2</td>
  <td>2,99 </td>
</tr>
</table>
</body>
</html>
```

Ejemplo 2: [metodols.html](#)

- 🔧 Obtener una lista con los valores de los campos de texto existentes en un formulario:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Método .map()</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <script src="http://code.jquery.com/jquery-1.9.0.min.js"
      type="text/javascript"></script>
    <script type="text/javascript">
      $(function()
      {
        $("p").append($(".input").map(function()
        {
          return $(this).val();
        }).get().join(", "));
      });
    </script>
  </head>
  <body>
    <p><b>Values: </b></p>
    <form>
      <input type="text" name="name" value="John"/>
      <input type="text" name="password" value="password"/>
      <input type="text" name="url" value="http://ejohn.org"/>
    </form>
  </body>
</html>
```

Ejemplo 3: [metodoMap.html](#)

- 🔧 Poner un borde rojo a los párrafos que no tengan aplicado el estilo de clase **sinresaltar**:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Método .not()</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <script src="http://code.jquery.com/jquery-1.9.0.min.js"
      type="text/javascript"></script>
    <script type="text/javascript">
```

```

    $(function()
    {
        $("p").not(".sinresaltar").css("border", "1px solid red");
    });
</script>
</head>
<body>
    <p class="sinresaltar">Esto es un párrafo de texto resaltado</li>
    <p>Esto es un párrafo de texto sin resaltar</li>
</body>
</html>

```

Ejemplo 4: [metodoNot.html](#)

Métodos de recorrido del árbol DOM

La siguiente tabla muestra los diferentes métodos existentes en el DOM transversal para recorrer el árbol DOM a partir de un elemento.

Método	Funcionamiento
.children(['selector'])	Obtiene los hijos de cada elemento en el conjunto seleccionado, opcionalmente filtrados por un selector.
.closest('selector')	Para cada elemento del conjunto, obtiene el primer elemento que se corresponde con el selector pasado, comenzando por el propio elemento y continuando con sus antecesores en el DOM.
.find('selector')	Obtiene los descendientes de cada elemento, filtrados por un selector.
.next(['selector'])	Obtiene el siguiente hermano de cada elemento del conjunto seleccionado, opcionalmente filtrado por un selector.
.nextAll(['selector'])	Obtiene los siguientes hermanos de cada elemento del conjunto seleccionado, opcionalmente filtrados por un selector.
.nextUntil('selector')	Obtiene los siguientes hermanos de cada elemento del conjunto seleccionado, hasta que uno de ellos se corresponda con el selector pasado como parámetro.
.offsetParent()	Obtiene el antecesor más cercano que está posicionado, es decir, que tiene position:relative o absolute.
.parent(['selector'])	Obtiene los padres de cada elemento en el conjunto seleccionado, opcionalmente filtrados por un selector.
.parents(['selector'])	Obtiene los antecesores de cada elemento en el conjunto seleccionado, opcionalmente filtrados por un selector.
.parentsUntil('selector')	Obtiene los antecesores de cada elemento en el conjunto seleccionado, hasta que uno de ellos se corresponda con el selector pasado como parámetro.
.prev(['selector'])	Obtiene el anterior hermano de cada elemento del conjunto seleccionado, opcionalmente filtrado por un selector.
.prevAll(['selector'])	Obtiene los anteriores hermanos de cada elemento del conjunto seleccionado, opcionalmente filtrados por un selector.
.prevUntil('selector')	Obtiene los anteriores hermanos de cada elemento del conjunto seleccionado, hasta que uno de ellos se corresponda con el selector pasado como parámetro.
.siblings(['selector'])	Obtiene los hermanos de cada elemento del conjunto seleccionado, opcionalmente filtrados por un selector.

Tabla 2: métodos de recorrido del árbol DOM

Veamos algunos ejemplos de uso:

🚦 Poner la fuente de color rojo a todos los `` que hayan dentro de un

`<p>`:

`<!DOCTYPE html>`


```
<html>
<head>
  <title>método .find()</title>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <script src="http://code.jquery.com/jquery-1.9.0.min.js"
    type="text/javascript"></script>
  <script type="text/javascript">
    $(document).ready(function()
    {
      $("p").find("span").css("color", "red");
    });
  </script>
</head>
<body>
  <p>Esto es un texto <span>que contiene span</span> y este otro
  <span>también lo contiene</span>
</body>
</html>
```

Ejemplo 5: [metodoFind.html](#)

- ✚ Poner el color de fondo rojo a los siguientes hermanos de un elemento hasta que aparezca un hermano de tipo <dt>:

```
<!DOCTYPE html>
<html>
<head>
  <title>método nextUntil</title>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <script src="http://code.jquery.com/jquery-1.9.0.min.js"
    type="text/javascript"></script>
  <script type="text/javascript">
    $(document).ready(function()
    {
      $("#term-2").nextUntil("dt").css("background-color", "red");
    });
  </script>
</head>
<body>
  <dl>
    <dt id="term-1">término 1</dt>
    <dd>definición 1-a</dd>
    <dd>definición 1-b</dd>
    <dd>definición 1-c</dd>
    <dd>definición 1-d</dd>

    <dt id="term-2">término 2</dt>
    <dd>definición 2-a</dd>
    <dd>definición 2-b</dd>
    <dd>definición 2-c</dd>

    <dt id="term-3">término 3</dt>
    <dd>definición 3-a</dd>
    <dd>definición 3-b</dd>
  </dl>
</body>
</html>
```

Ejemplo 6: [metodoNextUntil.html](#)

- ✚ Poner el color de fondo rojo al padre de un element:

```

<!DOCTYPE html>
<html>
  <head>
    <title>método .parent()</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <script src="http://code.jquery.com/jquery-1.9.0.min.js"
      type="text/javascript"></script>
    <script type="text/javascript">
      $(document).ready(function()
      {
        $(".li.item-a").parent().css("background-color", "red");
      });
    </script>
  </head>
  <body>
    <ul>
      <li>coches</li>
      <li>motos</li>
      <li>accesorios:
        <ul>
          <li class="item-a">ruedas</li>
          <li>antenas</li>
          <li>radiocassettes</li>
        </ul>
      </li>
    </ul>
  </body>
</html>

```

Ejemplo 7: [metodoParent.html](#)

Otros métodos transversales del DOM

Por último, veremos otros métodos que no se pueden clasificar en ninguno de los tipos anteriores.

Método	Funcionamiento
.add('selector')	Añade elementos al conjunto de elementos seleccionado
.andSelf()	Añade el anterior conjunto de elementos existente en la pila al actual conjunto de elementos seleccionados
.end()	Finaliza la operación de filtrado más reciente en el encadenamiento actual y devuelve el conjunto de elementos seleccionados a su anterior estado.

Tabla 3: Otros métodos transversales del DOM

Veamos algunos ejemplos de uso:

- En el siguiente ejemplo se muestran dos casos, el primero le pone el color de fondo amarillo a los párrafos que hay dentro de un <div>, y el segundo le pone el color de fondo amarillo a los párrafos que hay dentro del <div> y también al propio <div>:

```

<!DOCTYPE html>
<html>
  <head>
    <title>método .andself()</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

```

```

<style>
  p, div { margin:5px; padding:5px; }
  .borde { border: 2px solid red; }
  .fondo { background:yellow; }
  .izquierda, .derecha { width: 45%; float: left;}
  .derecha { margin-left:3%; }
</style>
<script src="http://code.jquery.com/jquery-1.9.0.min.js"
  type="text/javascript"></script>
<script type="text/javascript">
  $(function()
  {
    $("div.izquierda,div.derecha").find("div,div > p").addClass("borde");
    // Primer ejemplo
    $("div.antes-andself").find("p").addClass("fondo");
    // Segundo ejemplo
    $("div.despues-andself").find("p").andSelf().addClass("fondo");
  });
</script>
</head>
<body>
  <div class="izquierda">
    <p><strong>Antes <code>andSelf()</code></strong></p>
    <div class="antes-andself">
      <p>Primer párrafo</p>
      <p>Segundo párrafo</p>
    </div>
  </div>
  <div class="derecha">
    <p><strong>Después <code>andSelf()</code></strong></p>
    <div class="despues-andself">
      <p>Primer párrafo</p>
      <p>Segundo párrafo</p>
    </div>
  </div>
</body>
</html>

```

Ejemplo 8: [metodoAndself.html](#)

- ✚ Poner el color de fondo rojo a los de la clase coches y el color de fondo verde a los de la clase motos, cuando sean descendientes de un :

```

<!DOCTYPE html>
<html>
  <head>
    <title>método .end()</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <script src="http://code.jquery.com/jquery-1.9.0.min.js"
      type="text/javascript"></script>
    <script type="text/javascript">
      $(function()
      {
        $('ul').find('li.coches').css('background-color', 'red')
          .end().find('li.motos').css('background-color', 'green');
      });
    </script>
  </head>
  <body>
    <ul>

```

```
<li class="camiones">Camiones</li>
<li class="coches">Coches</li>
<li class="motos">Motos</li>
<li class="furgonetas">Furgonetas</li>
</ul>
</div>
</body>
</html>
```

Ejemplo 9: [metodoEnd.html](#)

4. Manipular el html y el css de nuestra página

Aprender a seleccionar elementos del DOM es el primer paso para entender el funcionamiento de jQuery, pero después de seleccionar un elemento, necesitaremos poder hacer algo con él. Hasta ahora sólo hemos cambiado los estilos de los elementos. Veremos a continuación, qué más podemos hacer.

Manipulando los elementos del DOM

jQuery es muy potente a la hora de añadir elementos, borrarlos, cambiar su contenido, duplicarlos, etc. En la tabla que se muestra a continuación, podemos ver los principales métodos que nos ofrece jQuery para realizar este tipo de operaciones.

Método ¹	Funcionamiento
<code>.html([contenido])</code>	Sin parámetros, obtiene el código html que contiene un elemento. Si le pasamos el nuevo contenido mediante una cadena de texto, este sustituirá al actual.
<code>.text()</code>	Este método nos devolverá el texto combinado que contengan los elementos seleccionados, eliminando de estos las etiquetas html existentes.
<code>.append(contenido)</code> <code>.appendTo(contenido)</code>	El método <code>.append()</code> , añadirá código al final de cada uno de los elementos seleccionados. El método <code>.appendTo()</code> , es parecido al método <code>.append()</code> , la diferencia radica en la colocación del contenido a añadir y el destino de este contenido. En el método <code>.append()</code> la expresión del selector anterior al método, es el contenedor y lo que pasamos como parámetro, es el contenido. Por el contrario, en el método <code>.appendTo()</code> , la expresión del selector anterior, es el contenido y lo que le pasamos como parámetro, es el contenedor.
<code>.prepend(contenido)</code> <code>.prependTo(contenido)</code>	El método <code>.prepend()</code> , funciona igual que el método <code>.append()</code> , pero insertando el contenido al principio de cada elemento. La diferencia entre <code>.prepend()</code> y <code>.prependTo()</code> , es la misma que entre <code>.append()</code> y <code>.appendTo()</code> .
<code>.after(contenido)</code> <code>.insertAfter(contenido)</code>	El método <code>.after()</code> , inserta el código que le pasemos como parámetro después de cada elemento seleccionado. La diferencia entre <code>.after()</code> y <code>.insertAfter()</code> , es la misma que entre <code>.append()</code> y <code>.appendTo()</code> .
<code>.before(contenido)</code> <code>.insertBefore(contenido)</code>	Hará lo mismo que <code>.after()</code> , pero insertará el contenido antes de los elementos, en lugar de después. La diferencia entre <code>.before()</code> e <code>.insertBefore()</code> , es la misma que entre <code>.append()</code> y <code>.appendTo()</code> .
<code>.replaceAll(destino)</code> <code>.replaceWith(origen)</code>	Reemplazan unos elementos por otros. Ambos métodos hacen lo mismo, la diferencia es que <code>.replaceAll()</code> reemplaza el origen por el destino y <code>.replaceWith()</code> lo hará al contrario. La forma de invocar al método sería la siguiente: <code>origen.replaceAll(destino)</code> . Como origen y destino especificaremos un selector u objeto jQuery.
<code>.wrap(envoltorio)</code> <code>.wrapAll()</code> <code>.wrapInner()</code>	Estos métodos envuelven un elemento, de forma que quede contenido dentro de otro. El método <code>.wrap()</code> , envuelve cada uno de los elementos seleccionados con el envoltorio indicado. Si en lugar de esto, quisiéramos envolver todos los elementos seleccionados dentro de un único envoltorio, usaríamos el método <code>.wrapAll()</code> . Si queremos envolver el contenido de los elementos, en lugar de los elementos en sí mismos, usaremos el método <code>.wrapInner()</code> .
<code>.unwrap()</code>	Elimina los padres de los elementos seleccionados. Este método no recibe parámetros.
<code>.clone()</code>	Duplica todos los elementos seleccionados. Después podemos colocarlos donde queramos utilizando otros métodos.

¹ El parámetro contenido de todos estos métodos puede ser: una cadena de texto con html, un objeto del DOM o un objeto jQuery. También podemos pasar varios parámetros de cualquier de estos tipos, separándolos por comas (Ver el ejemplo [metodosBeforeAfter.html](#)).

<code>.remove([selector])</code>	Elimina el conjunto de elementos seleccionados. Si le pasamos un selector como parámetro se hará un filtrado de los objetos a eliminar, de forma que sólo se eliminarán los que cumplan el filtro.
<code>.empty()</code>	Elimina todos los descendientes del conjunto de elementos seleccionado.

Tabla 4: Métodos para manipular los elementos del DOM de la página

A continuación, veremos una serie de ejemplos, que nos muestran técnicas muy sencillas para cambiar los contenidos de nuestra página manipulando los elementos del DOM.

Cambiando el html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Método .html()</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <script src="http://code.jquery.com/jquery-1.9.0.min.js"
      type="text/javascript"></script>
    <script type="text/javascript">
      $(function()
      {
        alert("contenido anterior: " + $('#contenido').html());
        $('#contenido').html('<div>Hola jQuery.</div>');
      });
    </script>
  </head>
  <body>
    <div id="contenido">
      <p>Corro 4 veces a la semana.</p>
      <p>Hago pesas 3 veces a la semana.</p>
    </div>
  </body>
</html>
```

Ejemplo 10: [metodoHtml.html](#)

Al ejecutar el ejemplo, vemos que, en primer lugar, se muestra en una ventana de alert el contenido html del <div> con la clase contenido y, posteriormente, se cambia el contenido del <div> por otro <div> que contiene el texto "Hola jQuery". De este ejemplo, se puede deducir que el método .html() sustituye el contenido existente por el nuevo contenido introducido.

Obteniendo el texto del objeto

```
<!DOCTYPE html>
<html>
  <head>
    <title>Método .text()</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <script src="http://code.jquery.com/jquery-1.9.0.min.js"
      type="text/javascript"></script>
    <script type="text/javascript">
```

```
$(function()  
    $("p.segundo").html($("p.primer").text());  
});  
</script>  
</head>  
<body>  
    <div id="contenido">  
        <p class="primero">texto con contenido <strong>importante</strong></p>  
        <p class="segundo"></p>  
    </div>  
</body>  
</html>
```

[Ejemplo 11: metodoText.html](#)

En el ejemplo anterior, se puede observar que en el segundo párrafo, no se ha copiado la etiqueta ``.

Añadiendo html dentro del elemento

En el siguiente ejemplo se añadirá un párrafo al principio del `<div>` contenido y otro al final:

```
<!DOCTYPE html>  
<html>  
    <head>  
        <title>Métodos .append() .prepend()</title>  
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />  
        <script src="http://code.jquery.com/jquery-1.9.0.min.js"  
            type="text/javascript"></script>  
        <script type="text/javascript">  
            $(function()  
            {  
                $('#contenido').append('<p>juego al futbol 1 vez a la semana</p>');  
                $('#contenido').prepend('<p>juego al pádel 1 vez a la semana</p>');  
            });  
        </script>  
    </head>  
    <body>  
        <div id="contenido">  
            <p>Corro 4 veces a la semana.</p>  
            <p>Hago pesas 3 veces a la semana.</p>  
        </div>  
    </body>  
</html>
```

[Ejemplo 12: metodosAppendPrepend.html](#)

El método `.appendTo()`, es parecido al método `.append()`, la diferencia radica en la colocación del contenido a añadir y el destino de este contenido. En el método `.append()` la expresión del selector anterior al método, es el contenedor y lo que pasamos como parámetro, es el contenido. Por el contrario, en el método `.appendTo()`, la expresión del selector anterior, es el contenido y lo que le pasamos

como parámetro, es el contenedor. Más adelante, veremos un ejemplo de uso de `.appendTo()`. También disponemos de un método `.prependTo()`.

Añadiendo html antes o después de un elemento

Hemos visto en los anteriores ejemplos, que podemos añadir html dentro de los elementos seleccionados, pero ¿qué ocurre cuando lo que queremos hacer es añadir elementos antes o después de estos elementos? Pues que entonces usaremos los métodos `.after()`, `.before()`, `.insertAfter()` o `.insertBefore()`.

Podemos pasarle a estos métodos un objeto jQuery, pero tenemos que tener en cuenta que no se hará una copia del objeto que le pasamos, sino que se moverá.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Métodos .before() .after()</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <script src="http://code.jquery.com/jquery-1.9.0.min.js"
      type="text/javascript"></script>
    <script type="text/javascript">
      $(function()
      {
        var $obj = $("<p>Test 3</p>");
        var parrafo = document.createElement("p");
        parrafo.appendChild(document.createTextNode("Test 4"));
        $(".inner").before("<p>Test</p>", "<p>Test2</p>", $obj, parrafo);
        $('.container').after($('h2'));
      });
    </script>
  </head>
  <body>
    <div class="container">
      <h2>Saludos</h2>
      <div class="inner">Hola</div>
      <div class="inner">Adios</div>
    </div>
  </body>
</html>
```

Ejemplo 13: [metodosBeforeAfter.html](#)

En el ejemplo, se han añadido varios párrafos antes de cada `<div>` de la clase `inner`. Además, el `<h2>` se ha movido de un lugar a otro, porque le hemos pasado al método `.after()` un objeto jQuery con este elemento seleccionado.

Los métodos `.after()` e `.insertAfter()`, realizan la misma tarea. La diferencia entre uno y otro, es la misma que hay entre los métodos `.append()` y `.appendTo()`, es decir, en el método `.after()`, la expresión del selector anterior al método, es el elemento después del cual el contenido es insertado y lo que pasamos como parámetro es el contenido. Por el contrario, en el método `.insertAfter()`, la

expresión del selector anterior, es el contenido y lo que le pasamos como parámetro, es el elemento previo.

Los métodos `.before()` e `.insertBefore()`, funcionan exactamente igual que los anteriores. La diferencia es que en lugar de insertar después insertan antes del elemento.

Reemplazando elementos

```
<!DOCTYPE html>
<html>
  <head>
    <title>Método .replaceAll()</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <script src="http://code.jquery.com/jquery-1.9.0.min.js"
      type="text/javascript"></script>
    <script type="text/javascript">
      $(function()
      {
        $('<h2>Saludos</h2>').replaceAll('.inner');
      });
    </script>
  </head>
  <body>
    <div class="container">
      <h2>Saludos</h2>
      <div class="inner">Hola</div>
      <div class="inner">Adios</div>
    </div>
  </body>
</html>
```

Ejemplo 14: [metodoReplaceAll.html](#)

En el ejemplo vemos que se reemplazan todos los elementos de la clase `inner` por un `<h2>`.

Envolviendo Elementos

En el siguiente ejemplo, vemos el funcionamiento de los métodos `.wrap()`:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Método .wrap()</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <style>
      p { padding:10px; }
      .one { border: green solid 1px; padding:10px; margin: 10px;}
      .all { border: blue solid 1px; padding:10px; margin: 10px;}
      .inner {border: red solid 1px; padding:10px; margin: 10px;}
    </style>
    <script src="http://code.jquery.com/jquery-1.9.0.min.js"
      type="text/javascript"></script>
    <script type="text/javascript">
      $(function()
```

```
{
  $("p").wrapAll('<div class="all"></div>');
  $("p").wrapInner('<div class="inner"></div>');
  $("p").wrap('<div class="one"></div>');
});
</script>
</head>
<body>
  <h2>Saludos</h2>
  <p>Hola</p>
  <p>Adios</p>
</body>
</html>
```

[Ejemplo 15: metodoWrap.html](#)

Desenvolviendo elementos

Evidentemente, también podemos desenvolver elementos, es decir, eliminar los elementos envolventes. Para esta labor, el método a utilizar es `.unwrap()`.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Método .wrap()</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <style>
      .nuevo { border: blue solid 1px;}
      .container {border: red solid 1px;}
    </style>
    <script src="http://code.jquery.com/jquery-1.9.0.min.js"
      type="text/javascript"></script>
    <script type="text/javascript">
      $(function()
      {
        $(".inner").unwrap('<div class="nuevo"></div>');
      });
    </script>
  </head>
  <body>
    <div class="container">
      <h2>Saludos</h2>
      <div class="inner">Hola</div>
      <div class="inner">Adios</div>
    </div>
  </body>
</html>
```

[Ejemplo 16: metodoUnwrap.html](#)

Duplicando elementos

Podemos encontrarnos situaciones, donde necesitemos copiar un elemento del DOM y colocarlo en algún otro lugar. Para esto, jQuery dispone del método `.clone()`.

En el siguiente ejemplo, se duplica el contenido del elemento con id `productos1` y se añade al final del elemento con id `productos2`. Para ello, se utiliza el método `.appendTo()`:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Método .clone()</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <script src="http://code.jquery.com/jquery-1.9.0.min.js"
      type="text/javascript"></script>
    <script type="text/javascript">
      $(function()
      {
        $('#productos1').clone().appendTo('#productos2');
      });
    </script>
  </head>
  <body>
    <div id="productos1">
      <p>Camisas</p>
      <p>Pantalones</p>
    </div>
    <div id="productos2">
    </div>
  </body>
</html>
```

[Ejemplo 17: metodoClone.html](#)

Eliminando elementos

Para eliminar elementos del DOM utilizaremos el método `.remove()`. Este método es muy fácil de usar, sólo tenemos que seleccionar el elemento o elementos a eliminar y llamar al método:

```
$(function() {
  $('#contenido').remove();
});
```

También podemos eliminar un descendiente o hijo del elemento seleccionado, pasándole el selector correspondiente al método `.remove()`. Por ejemplo, para borrar todos los párrafos que contengan el texto "Hola" haremos lo siguiente:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Método .remove()</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <script src="http://code.jquery.com/jquery-1.9.0.min.js"
      type="text/javascript"></script>
    <script type="text/javascript">
      $(function()
      {
        $("p").remove(":contains('Hola')");
      });
    </script>
  </head>
  <body>
    <p>Hola</p>
    <p>Adios</p>
  </body>
</html>
```

```
</head>
<body>
  <div id="contenido">
    <p>Corro 4 veces a la semana.</p>
    <p>Hola Hago pesas 3 veces a la semana.</p>
  </div>
</body>
</html>
```

Ejemplo 18: [metodoRemove.html](#)

Si en lugar de eliminar uno o varios elementos, lo que nos interesa es vaciarlos, es decir, eliminar todos sus hijos, podemos utilizar el método `.empty()`. Este método borrará todos los descendientes de los elementos seleccionados, incluidos los textos.




Accediendo a los atributos

Al igual que tenemos métodos jQuery que nos permiten acceder a los elementos del DOM, también disponemos de otros métodos que nos facilitan la tarea de trabajar con los atributos de los elementos. En la siguiente tabla se muestran estos métodos:

Método	Funcionamiento
<code>.attr()</code>	Obtiene el valor de un atributo para el primer elemento de los seleccionados o establece el valor de un atributo para el conjunto de elementos seleccionados.
<code>.prop()</code>	Obtiene el valor de una propiedad para el primer elemento de los seleccionados o establece el valor de una propiedad para el conjunto de elementos seleccionados.
<code>.removeAttr()</code>	Elimina un atributo de los elementos seleccionados
<code>.removeProp()</code>	Elimina una propiedad de los elementos seleccionados
<code>.val()</code>	Obtiene el valor del primer elemento seleccionado o establece el valor para el conjunto de elementos seleccionados (Se utiliza para elementos de formulario).

Tabla 5: Métodos para acceder a los atributos

El método `.attr()` tiene dos posibles usos:

-  `.attr(nombreAtributo)`: obtiene el valor del atributo en el primer elemento del conjunto seleccionado. Si lo que queremos es obtener el valor para cada uno de los elementos, tendremos que recorrerlos todos utilizando un bucle `.each()`.
-  Devolverá `undefined` para los elementos que no tengan establecido el atributo.
-  `.attr(nombreAtributo, valorAtributo)`: establece el valor del atributo para el conjunto de elementos seleccionados.

El uso de este método para obtener el valor de un atributo tiene dos ventajas:

- ✚ Al llamarse a través de un objeto jQuery, se puede utilizar el encadenamiento.
- ✚ Los valores de algunos atributos no son consistentes entre distintos navegadores. Este método reduce estas inconsistencias.

Ejemplo de uso. HTML:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Método .attr()</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <script src="http://code.jquery.com/jquery-1.9.0.min.js"
      type="text/javascript"></script>
    <script type="text/javascript">
      $(function()
      {
        $("#titulo").text($("#img").attr("alt"));
      });
    </script>
  </head>
  <body>
    <div>
      
    </div>
    <div id="titulo"></div>
  </body>
</html>
```

Ejemplo 19: [metodoAttr.html](#)

El método `.prop()` es muy similar al anterior, pero en lugar de obtener atributos, obtiene o modifica propiedades. Antes de la versión 1.6 de jQuery se utilizaba el mismo método `.attr()` para trabajar con atributos y con propiedades, pero a partir de esta versión aparece este nuevo método.

Como ejemplo, las propiedades `selectedIndex`, `tagName`, `nodeName`, `nodeType`, `ownerDocument`, `defaultChecked`, `anddefaultSelected` deberían ser manejadas mediante este método.

Los métodos `.removeAttr()` y `.removeProp()` eliminan atributos de forma consistente entre distintos navegadores.

El método `.val()` lo usaremos para obtener o establecer el valor de los campos de formulario.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Método .val()</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <script src="http://code.jquery.com/jquery-1.9.0.min.js"
      type="text/javascript"></script>
    <script type="text/javascript">
      $(function()
      {
        var contador = $("#contador").val();
        alert("Contador actual: " + contador);
        contador++;
        $("#contador").val(contador);
      });
    </script>
  </head>
  <body>
    <form id="formulario" method="post" action="login.php">
      <input type="text" name="contador" value="1" id="contador" />
    </form>
  </body>
</html>
Ejemplo 20: metodoVal.html
```

Para select y checkbox podemos utilizar el método .val() de la siguiente forma:

```
// Obtener el valor de un desplegable
$('select.foo option:selected').val();
// Igual que el anterior, pero más fácil
$('select.foo').val();
// Obtener el valor de un checkbox chequeado
$('input:checkbox:checked').val();
// Obtener el valor del radio botón seleccionado en un grupo
$('input:radio[name=bar]:checked').val();
```

5. Trabajando con estilos

Hemos visto anteriormente el método `css()` que añade estilos en línea a los elementos del DOM. Esto no es demasiado correcto porque estamos perdiendo la separación entre la visualización de los elementos (css) y la lógica de negocio de nuestra página (javascript). En lugar de esto, es más correcto utilizar estilos existentes en nuestra hoja de estilos y aplicarlos a los elementos necesarios. En la siguiente tabla podemos ver los métodos de los que dispone jQuery para realizar estas tareas.

Método	Funcionamiento
<code>.css()</code>	Obtiene o añade propiedades css a los elementos seleccionados
<code>.addClass()</code>	Añade una clase css a los elementos seleccionados
<code>.hasClass()</code>	Devuelve cierto si los elementos seleccionados tienen una clase css aplicada
<code>.removeClass()</code>	Elimina una clase css a los elementos seleccionados
<code>.toggleClass()</code>	Añade o elimina una clase css a los elementos seleccionados. Si la tiene la elimina, si no la tiene la añade.

Tabla 6: métodos para trabajar con estilos

A continuación, veremos algunos ejemplos de uso:

- ✚ Añadir borde a los elementos seleccionados:
`$('.container').css('border','1px solid #333');`
- ✚ Añadir una clase a los elementos seleccionados:
`$('.container').addClass('active');`
- ✚ Añadir varias clases a los elementos seleccionados:
`$('.container').addClass('active book');`
- ✚ Eliminar una clase a los elementos seleccionados:
`$('#page-wrap').removeClass('alternate');`
- ✚ Eliminar varias clases a los elementos seleccionados:
`$('#page-wrap').removeClass('alternate main product');`
- ✚ Eliminar todas las clases a los elementos seleccionados:
`$('#page-wrap').removeClass();`
- ✚ Añadir una clase a un elemento si no la tiene y eliminarla si la tiene:
`$('#page-wrap').toggleClass('alternate');`

6. Introducción a los eventos

En posteriores sesiones, hablaremos detalladamente de los eventos en jQuery, pero daremos ahora una pequeña introducción, para poder hacer los ejercicios del tema.



Cualquier aplicación con interfaz gráfica moderna está basada en eventos, y las aplicaciones Web no son una excepción. Todas las aplicaciones basadas en eventos utilizan manejadores de eventos, que no son más que funciones que se ejecutan cuando se produce un suceso o acontecimiento importante (un evento), como por ejemplo, un clic del ratón, la finalización de la carga de la página, etc.

Aunque en jQuery existen varias formas de asociar un manejador de evento a un evento, la forma más sencilla, es a través de los métodos de evento.

Veamos el siguiente ejemplo:

```
<!DOCTYPE html>
<html>
  <head>
    <title>.click()</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <script src="http://code.jquery.com/jquery-1.9.0.min.js"
      type="text/javascript"></script>
    <script type="text/javascript">
      $(function()
      {
        $("a").click(function(event)
        {
          var contador = parseInt($(this).text());
          contador++;
          $(this).html(contador)
          return false;
        });
      });
    </script>
  </head>
  <body>
    <p>Pulsa sobre el número para incrementarlo</p>
    <a href="">0</a>
  </body>
</html>
Ejemplo 21: click.html
```

En el ejemplo, vemos como asociamos un manejador de evento para el evento `click` del enlace. Este manejador de evento incrementa el número que contiene el enlace realizando los siguientes pasos:

-  Obtiene el texto.
-  Lo convierte a entero.

✚ Lo incrementa.

✚ Vuelve a cargar el valor incrementado en el html del enlace.

Cuando pulsamos sobre un enlace, la acción por defecto, es cargar una nueva página. Una de las formas de evitar que el enlace se comporte así, es haciendo que el manejador de evento devuelva `false`.

Como se ve en el ejemplo, para poder acceder al elemento, sobre el cual, se produce el evento, podemos utilizar el objeto `$(this)`. `$(this)` será un objeto jQuery, por lo tanto, podremos llamar a cualquier método a través de él, en el ejemplo se llama a los métodos `.text()` y `.html()` de este objeto.

Podemos observar, que para asociar el manejador de evento primero seleccionamos el/los elemento/s a los que queremos asociárselo, y después, invocamos al método de evento correspondiente, en este caso el método `.click()`. Si el selector utilizado, seleccionara más de un elemento, el manejador de evento se asociaría a todos los elementos seleccionados. Por ejemplo, si hubiéramos tenido más de un enlace en la página, el mismo manejador de evento se habría asociado para todos ellos y, al pulsar sobre cualquiera de ellos, se ejecutaría el manejador de evento asociado.

Los métodos que podemos utilizar para asociar manejadores a los eventos correspondientes, son los siguientes: `.blur()`, `.focus()`, `.select()`, `.submit()`, `.click()`, `.dblclick()`, `.toggle()`, `.focusin()`, `.error()`, `.focusout()`, `.resize()`, `.hover()`, `.scroll()`, `.mouseenter()`, `.mousemove()`, `.mouseout()`, `.mouseover()`, `.mouseup()`, `.mousedown()`.

En posteriores sesiones, seguiremos hablando ampliamente sobre el tema de los eventos.

7. Firebug avanzado

La solapa Red

Si la solapa está deshabilitada, sólo tenemos que pulsar sobre el triángulo gris que aparece cuando nos ponemos sobre ella y seleccionar la opción Habilitado.

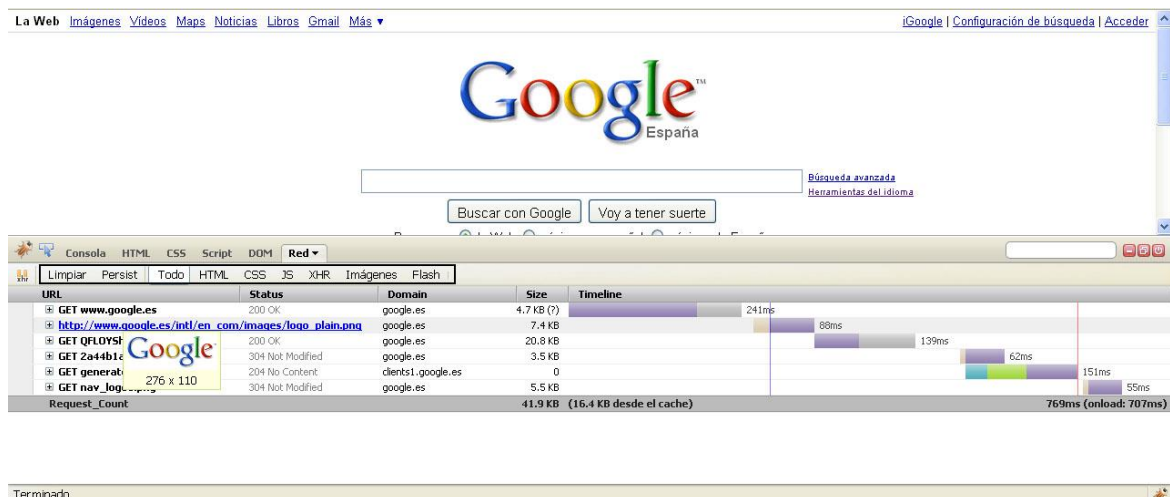
En esta solapa podemos ver todos los recursos que carga la página, en qué orden, cuánto tarda y lo más importante: señala en rojo aquellos que no pudo encontrar (esto quita muchos dolores de cabeza). Además, si nos situamos sobre la petición nos ofrece una vista previa de la imagen y/o datos del recurso.

Con las pestañas a la derecha de “Todo” (en la parte superior de la solapa) podemos filtrar los tipos de peticiones que queremos visualizar:

- “HTML” para documentos HTML.
- “CSS” para hojas de estilo.
- “JS” para archivos javascript.
- “Imágenes” para documentos de imagen.
- “Flash” para objetos Flash.
- “XHR” para capturar las peticiones AJAX, con lo que podemos ver tanto las peticiones que se envían como las respuestas de las mismas. En sesiones posteriores esta funcionalidad nos será de mucha utilidad.

En la parte superior de la solapa tenemos dos botones más:

- “Limpiar” borra las peticiones que aparecen en la ventana.
- “Persist” fija las peticiones mostradas, de modo que si pulsamos el botón actualizar del navegador aparecerán las peticiones de la nueva carga en un grupo a parte sin eliminar las actuales.






La solapa Consola

Esta solapa se usa principalmente para registrar los eventos que suceden durante la ejecución (Logging), de esta forma no tendremos que mostrar alerts o document.write para visualizar el estado de variables o verificar que la ejecución pasa por un punto determinado.

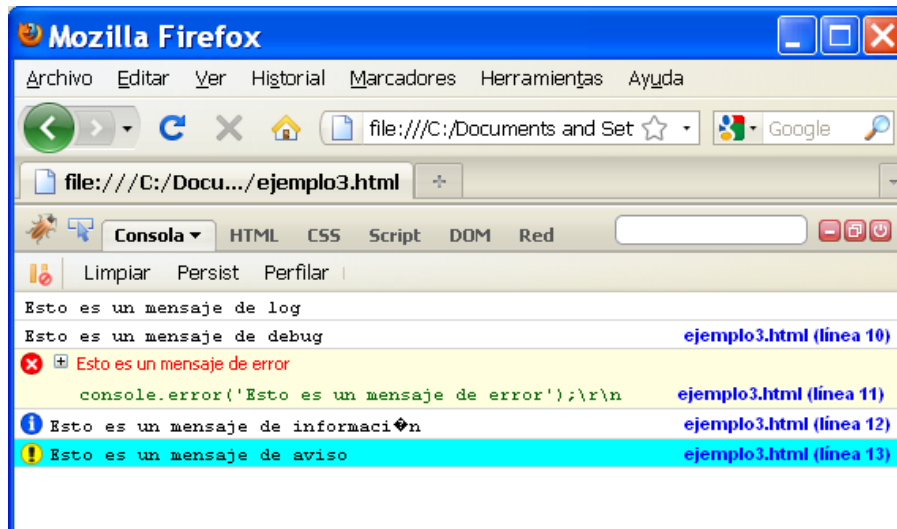
Tipos de Loggings

Podemos mostrar los loggings por la consola utilizando el objeto console de Firebug. Existen 5 tipos de Loggings:

1. console.log: Muestra un mensaje sin icono.
2. console.debug: Muestra un mensaje en la consola, incluyendo un hipervínculo a la línea donde fue llamado.
3. console.error: Muestra un mensaje en la consola con un hipervínculo a la línea donde fue llamado y el icono .
4. console.info: Muestra un mensaje en la consola con un hipervínculo a la línea donde fue llamado y el icono .
5. console.warn: Muestra un mensaje en la consola con un hipervínculo a la línea donde fue llamado y el icono .

Ejemplo: Vamos a crear un nuevo documento html, en el cual, incluiremos el siguiente script:

```
<script language="javascript" type="text/javascript">
<!-- // 
console.log("Esto es un mensaje de log");
console.debug("Esto es un mensaje de debug");
console.error("Esto es un mensaje de error");
console.info("Esto es un mensaje de información");
console.warn("Esto es un mensaje de aviso");
// ]]&gt; --&gt;
&lt;/script&gt;</pre></div><div data-bbox="138 722 862 761" data-label="Text"><p>Si abrimos el documento con Firefox y nos vamos a la solapa Consola veremos los logs que se han registrado.</p></div><div data-bbox="138 924 346 940" data-label="Page-Footer"><hr/>Ponente: Alejandro Amat Reina</div><div data-bbox="785 924 862 940" data-label="Page-Footer">Página 26</div>
```



Si pulsamos sobre uno de los hipervínculos de los logs que se han registrado nos llevará a la solapa script, seleccionando la línea donde se llama a la consola que genera ese mensaje.

Mostrar los valores de las variables

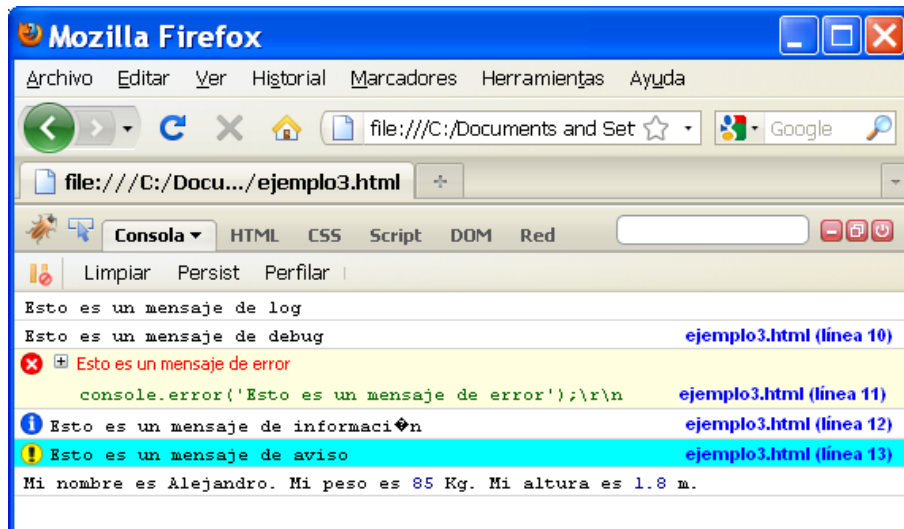
En los mensajes mostrados podemos visualizar los valores de las variables que nos interesen. La forma de mostrar estas variables es similar a como lo hacemos con la función printf de C, introducimos un patrón dentro de la cadena del mensaje que será sustituido por el valor de la variable que le indiquemos. Podemos utilizar los siguientes patrones de formato:

- %s: String.
- %d o %i enteros (todavía no se soportan números formateados).
- %f números en coma flotante.

Ejemplo: en el documento html anterior añadiremos el siguiente script:

```
<script language="javascript" type="text/javascript">
<!-- // <![CDATA[
var nombre = "Alejandro";
var peso = 85;
var altura = 1.80;
console.log("Mi nombre es <strong>%s</strong>. " +
            "Mi peso es <strong>%d</strong> Kg. " +
            "Mi altura es <strong>%f</strong> m.", nombre, edad, altura);
// ]]> -->
</script>
```

Si abrimos el documento con Firefox y nos vamos a la solapa Consola veremos los logs que se han registrado.




La solapa Script

La solapa Script es la que más nos interesa de todas, puesto que nos va a permitir depurar nuestro código javascript. En esta solapa visualizamos el contenido de la página tal y como aparece en el documento. Para insertar un breakpoint² en el código sólo tenemos que hacer clic con el botón izquierdo del ratón a la izquierda del número de línea en la cual queremos que la ejecución se pare. Una vez hemos situado el breakpoint en el lugar deseado, sólo tenemos que pulsar el botón actualizar del navegador y la ejecución del código se parará donde le hemos indicado.

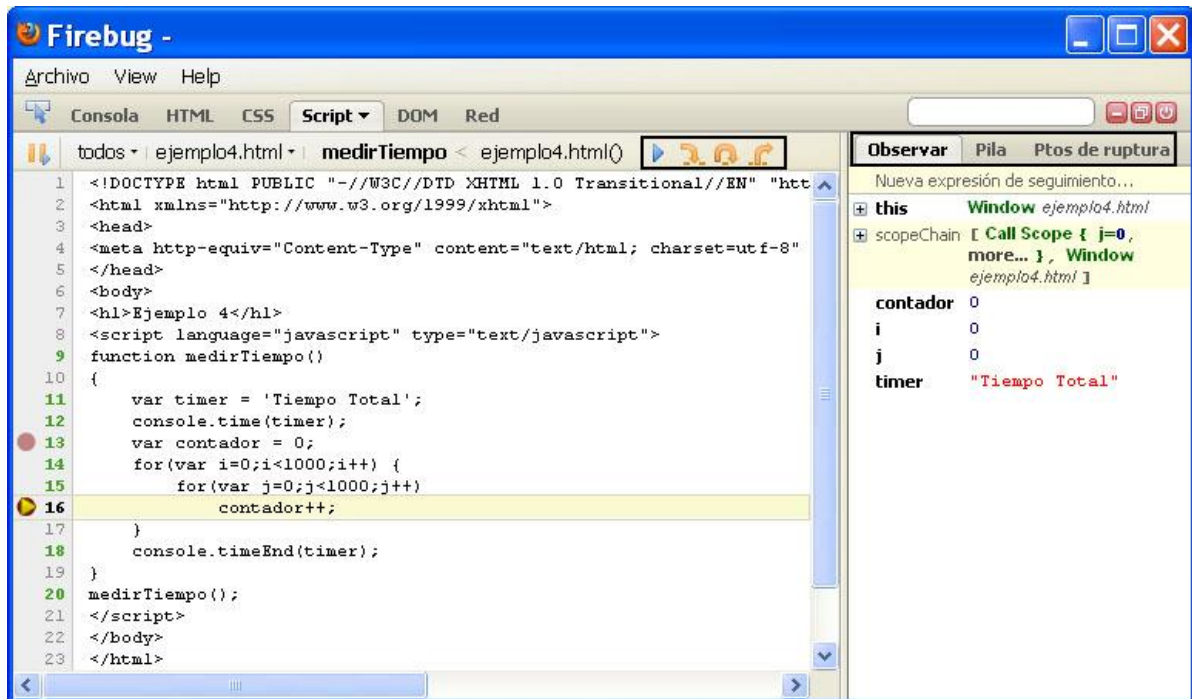
La ventana de depuración está dividida en dos, la parte de la izquierda nos muestra nuestro código y el punto donde se encuentra la ejecución, mientras que en la parte de la derecha tenemos distintas solapas que nos ofrecen información muy útil:

- La solapa “Observar” nos muestra el valor que van tomando las variables a lo largo de la ejecución.
- La solapa “Pila” nos muestra la pila de llamada de funciones que se han realizado hasta llegar al punto en el que nos encontramos.
- La solapa puntos de ruptura nos muestra donde hemos insertado breakpoints, y nos permite habilitarlos o deshabilitarlos.

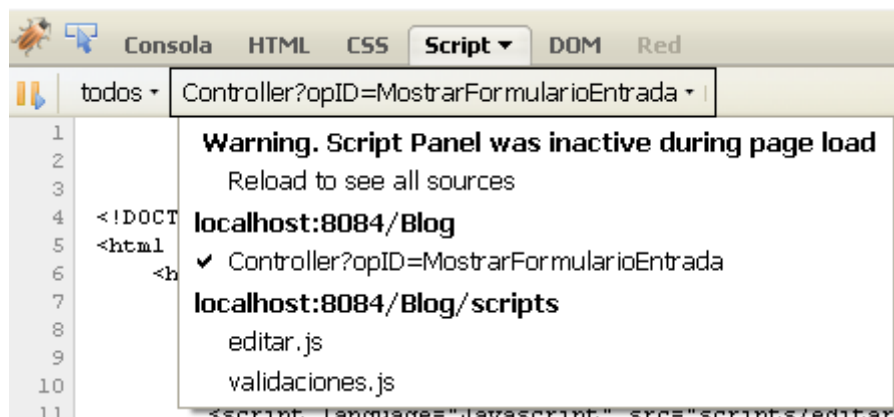
Una vez que la ejecución se ha parado en un breakpoint podemos controlar la ejecución con los botones que aparecen en la parte superior de la ventana , estos botones nos permiten (de izquierda a derecha): continuar con la ejecución hasta el siguiente breakpoint, saltar a la siguiente instrucción (si se tratará de una llamada a

² Breakpoint (punto de ruptura) la ejecución se parará en este punto para que podamos visualizar el contenido de las variables o realizar una ejecución paso a paso

función entraría), saltar a la siguiente instrucción (si se tratará de una llamada a función la saltaría), continuar la ejecución hasta salir de la función actual.



Para depurar un script externo lo seleccionaremos en el desplegable que aparece en la barra principal de la solapa.



8. Ejercicios

En este tema, seguiremos trabajando la aplicación del carro de la compra. En concreto, implementaremos la funcionalidad de añadir y eliminar productos al carrito.

Para evitar problemas, antes de empezar, eliminaremos de nuestro `document.ready` simplificado todos los selectores que se pidieron en el tema anterior.

Ejercicio 1

Al hacer doble click con el ratón (evento `dblclick`) sobre uno de los artículos de nuestra lista, se deben realizar las siguientes acciones:

- ✚ Comprobaremos que quedan artículos disponibles. El stock debe ser mayor que 0, si no es así, no haremos ninguno de los siguientes puntos.
 - ✚ Restaremos 1 al número de artículos disponibles (stock). Si después de disminuir el stock, no quedan más artículos disponibles, le añadiremos al elemento de la clase `stock` del artículo, la clase `agotado`. Esto hará que el stock aparezca tachado (otra forma de saber si hay artículos disponibles, sería comprobar si el elemento tiene aplicada esta clase).
 - ✚ Incrementaremos el número de artículos comprados.
 - ✚ Actualizaremos el precio total de la compra, sumándole el precio del artículo.
 - ✚ Añadiremos el producto al principio del carrito. Para ello, seguiremos los siguientes pasos:
 - Haremos una copia del artículo sobre el que se ha pulsado. Como esta copia la tenemos que introducir en el carrito, para evitar tener un id duplicado, le añadiremos al atributo `id` una "c" delante. Construiremos el id de la siguiente forma:
 - "c" + id
- Cuando introduzcamos dos artículos iguales en el carrito, el id seguirá estando duplicado, pero ya solucionaremos esto más adelante.
- Le añadiremos la clase `icart` a la copia creada.
 - Le ocultaremos el elemento de la clase `stock` (para hacer esto, podemos utilizar el método `.hide()` de jQuery).

- Cambiaremos la propiedad css `cursor` del elemento y de todos sus hijos al valor default (`$(selector).css("cursor", "default")`).
- Le añadiremos al principio un enlace (lo utilizaremos para eliminar el artículo del carrito). Para añadir el enlace usaremos el siguiente código:

```
var $delete = $('<a href="" class="delete"></a>');
```

Esto nos creará un objeto jQuery que podremos añadir, por ejemplo, con el método `.prepend($delete)`.
- Añadiremos la copia creada al principio (método `.prepend()`) del contenedor de artículos comprados del carrito (elemento con id `cart_items`).

De momento, sólo se verán en el carrito los cuatro primeros artículos añadidos, pero más adelante solucionaremos ese problema.

Ejercicio 2

Para eliminar un producto del carrito, se debe pulsar sobre el enlace que se ha creado al introducirlo, por lo tanto, tendremos que asociar un manejador de evento al enlace `delete` que creamos dinámicamente. Esto lo haremos con el método `.click()` visto durante el tema:

```
$delete.click(...);
```

Es recomendable, por claridad, reutilización y para evitar problemas, que se divida el código en funciones.

Al pulsar sobre el enlace, debemos hacer lo siguiente:

- ✚ Obtendremos el id del elemento padre, es decir, el artículo que hay dentro del carrito y, a partir de este, obtendremos el id de la lista de artículos a comprar. Como esto lo hacemos desde el evento click del enlace, podemos acceder a él mediante `$(this)` y al artículo del carrito mediante `$(this).parent()`. Una vez obtenido el id del artículo del carrito que queremos eliminar, podemos obtener el de la lista de artículos con el siguiente código:

```
var id = idPadre.substring(1);
```

Donde `idPadre` será el que hemos obtenido del elemento padre del enlace.

- ✚ Una vez tenemos el id, debemos incrementar el stock disponible del artículo que se elimina del carrito. Si el stock estaba a cero, debemos quitar la clase agotado del mismo.
- ✚ Actualizaremos el total de productos comprados restándole uno.
- ✚ Actualizaremos el precio total de la compra restándole el precio del artículo eliminado.
- ✚ Eliminaremos el artículo del carro de la compra.
- ✚ Como la acción por defecto de un enlace al hacer click sobre él es navegar a otra página, debemos evitar este comportamiento. Para hacer esto, bastará con que el manejador de evento devuelva false (`return false;`), esto evitará que el enlace realice su acción predeterminada.

En el aula virtual, encontraréis un vídeo en el que se explica cómo tiene que quedar nuestra aplicación después de realizar los ejercicios indicados.

9. Índice de ejemplos y tablas

Ejemplos

EJEMPLO 1: MÉTODOHAS.HTML.....	5
EJEMPLO 2: METODOIS.HTML.....	6
EJEMPLO 3: METODOMAP.HTML.....	6
EJEMPLO 4: METODONOT.HTML.....	7
EJEMPLO 5: METODOFIND.HTML.....	8
EJEMPLO 6: METODONEXTUNTIL.HTML.....	8
EJEMPLO 7: METODOPARENT.HTML.....	9
EJEMPLO 8: METODOANDSELF.HTML.....	10
EJEMPLO 9: METODOEND.HTML.....	11
EJEMPLO 10: METODOHTML.HTML.....	13
EJEMPLO 11: METODOTEXT.HTML.....	14
EJEMPLO 12: METODOSAPPENDPREPEND.HTML.....	14
EJEMPLO 13: METODOSBEFOREAFTER.HTML.....	15
EJEMPLO 14: METODOREPLACEALL.HTML.....	16
EJEMPLO 15: METODOWRAP.HTML.....	17
EJEMPLO 16: METODOUNWRAP.HTML.....	17
EJEMPLO 17: METODOCLONE.HTML.....	18
EJEMPLO 18: METODOREMOVE.HTML.....	19
EJEMPLO 19: METODOATTR.HTML.....	20
EJEMPLO 20: METODOVAL.HTML.....	21
EJEMPLO 21: CLICK.HTML.....	23

Tablas

TABLA 1: MÉTODOS DE FILTRADO DEL DOM TRANSVERSAL.....	4
TABLA 2: MÉTODOS DE RECORRIDO DEL ÁRBOL DOM.....	7
TABLA 3: OTROS MÉTODOS TRANSVERSALES DEL DOM.....	9
TABLA 4: MÉTODOS PARA MANIPULAR LOS ELEMENTOS DEL DOM DE LA PÁGINA.....	13
TABLA 5: MÉTODOS PARA ACCEDER A LOS ATRIBUTOS.....	19
TABLA 6: MÉTODOS PARA TRABAJAR CON ESTILOS.....	22